

USER GUIDE

Essential Studio

for EJ2 Angular

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

DateTimePicker	34
Getting started with Angular Datetimepicker component	34
Dependencies.....	34
Setup Angular environment.....	34
Create a new application	34
Installing Syncfusion DateTimePicker package	35
Registering DateTimePicker module.....	35
Adding CSS reference.....	36
Adding DateTimePicker component	36
Running the application	37
Setting the min and max	37
See Also	38
Globalization in Angular Datetimepicker component	38
Right-To-Left	41
Strict mode in Angular Datetimepicker component.....	42
Date time range in Angular Datetimepicker component	44
Date time masking in Angular Datetimepicker component	45
Configure Mask Placeholder	46
Customization in Angular Datetimepicker component	47
Day and Time Cell format.....	47
Adding mandatory asterisk to placeholder and float label.....	48
See Also	49
Accessibility in Angular Datetimepicker component.....	49
WAI-ARIA attributes.....	50
Keyboard Interaction	50
Input Navigation.....	50
Calendar Navigation.....	51
Ensuring accessibility	52
See also	53
Style appearance in Angular Datetimepicker component.....	53
Customizing the appearance of DateTimePicker wrapper element.....	53
Customizing the DateTimePicker icons element	53
Customizing the time picker popup in the DateTimePicker	53
Customizing the Calendar popup of the DateTimePicker	54
Full screen mode support in mobiles and tablets.....	54

How To	56
Json data binding with datetimepicker in Angular Datetimepicker component	56
Two way binding in Angular Datetimepicker component	57
Disable placeholder readonly in Angular Datetimepicker component.....	58
Custom event emitter in Angular Datetimepicker component	58
Custom validation using form validator in Angular Datetimepicker component.....	59
Reactive form in Angular Datetimepicker component	61
Template driven forms in Angular Datetimepicker component.....	62
Customize the datetimepicker day header in Angular Datetimepicker component.....	64
Ej1 api migration in Angular Datetimepicker component	66
DateTime Selection	66
DateTime Format	66
Calendar Views.....	66
Date Range	66
Disabled Dates	66
Customization	67
Accessibility.....	69
Persistence	69
Validation	69
Common.....	70
Globalization	74
Strict Mode	74
Open and Close	74
View Navigation	74
Diagram.....	75
Getting started with Angular Diagram component	75
Setup Angular Environment.....	75
Create an Angular Application	75
Installing Syncfusion Diagram package	76
Registering Diagram Module	76
Adding CSS reference.....	77
Add Diagram component.....	77
Defining Basic Diagram	78
Module Injection.....	78
Flow Diagram	79

Automatic Organization Chart	84
Nodes in Angular Diagram component.....	89
Create node.....	89
Add node through nodes collection.....	90
Add/Remove node at runtime	91
Add node from palette.....	92
Create node through data source.....	92
Draw nodes	92
Position	92
Flip.....	94
Appearance	95
Customize the style of main node on multi-selection.	96
Gradient	96
Linear gradient	97
Radial gradient	98
Shadow.....	99
Customizing shadow	100
Icon.....	101
Customizing expand icon	103
Customizing collapse icon	103
Interaction.....	103
Constraints	103
Custom properties	103
Stack order	103
Data flow	103
See Also	105
Shapes in Angular Diagram component	105
Text	106
Image.....	107
Image alignment	110
HTML.....	112
HTML Node With Template	113
Native	114
SVG content alignment	117
Basic shapes	118

Path	120
Flow Shapes	121
Bpmn shapes in Angular Diagram component	122
Event	124
Gateway	126
Activity	128
Tasks.....	129
Subprocess	132
Event subprocess	133
Transaction subprocess.....	134
Process	136
Loop.....	136
Compensation.....	138
Call.....	139
Adhoc	140
Boundary.....	141
Data	142
Datasource	144
Artifact	145
Text annotation.....	145
Group	147
BPMN flows.....	148
Association	148
Sequence.....	149
Message	151
UmdlDiagram in Angular Diagram component	152
UML Class Diagram Shapes	152
UML Class Relationships	157
How to add UML child at runtime.....	163
Adding UML Nodes in Symbol palette	168
Editing	170
UML Activity diagram.....	171
UML Activity diagram Shapes	171
Connectors in Angular Diagram component	176
Create connector	176

Add connectors through connectors collection.....	176
Add connector at runtime.....	177
Connectors from palette.....	178
Draw connectors.....	178
Update connector at runtime	179
Connect nodes	180
Connections with ports	182
Segments.....	187
Straight.....	187
Orthogonal	190
Avoid overlapping	193
How to customize Orthogonal Segment Thumb Shape.....	195
Bezier	197
Avoid overlapping with bezier	201
Decorator	207
Padding	209
Hit padding.....	211
Flip.....	212
Bridging	213
Corner radius.....	215
Appearance	216
Segment appearance	216
Decorator appearance	218
Interaction.....	219
Automatic line routing.....	219
Constraints	222
Custom properties	223
Stack order	223
Enable Connector Splitting.....	224
See Also	226
Group in Angular Diagram component.....	226
Create group	226
Add group when initializing diagram	226
Add group at runtime	230
Add children To group at runtime	231

Remove children from group at runtime	231
Container.....	232
Canvas	233
Stack.....	233
Difference between a basic group and containers	235
Interaction.....	235
See Also	235
Swim lane in Angular Diagram component	235
Create a swimlane.....	235
Lanes	241
Phase	255
Add swimlane to palette	261
Limitations.....	264
Labels in Angular Diagram component.....	264
Create annotation	265
Add annotations at runtime.....	266
Remove annotation	268
Update annotation at runtime.....	269
Alignment.....	270
Offset.....	270
Horizontal and vertical alignment.....	271
Annotation alignment with respect to segments	274
Margin.....	275
Text align	277
Hyperlink	278
Template Support for Annotation	279
Wrapping.....	280
Text overflow	282
Appearance	283
Interaction.....	285
Edit	286
Read-only annotations.....	287
Drag Limit	288
Multiple annotations	289
Constraints	290

Ports in Angular Diagram component	290
Create port	291
Add ports when initializing nodes.....	292
Add ports at runtime.....	293
Remove ports at runtime	294
Update port at runtime.....	296
Appearance	297
Offset.....	298
Constraints	299
Constraints in Angular Diagram component.....	299
Diagram constraints	299
Node constraints.....	299
Connector constraints.....	300
Port constraints.....	301
Annotation constraints	302
Selector constraints	303
Snap constraints.....	304
Boundary constraints.....	305
Inherit behaviors	306
Bitwise operations	307
Add operation	307
Remove Operation	307
Check operation	307
Interaction in Angular Diagram component	307
Selection.....	307
Single selection	307
Selecting a group.....	308
Multiple selection	308
Select/Unselect elements using program	309
Select entire elements in diagram programmatically.....	309
Drag.....	309
Resize	310
Rotate.....	310
Connection editing.....	311
End point handles	311

Straight segment editing.....	312
Orthogonal thumbs.....	312
Bezier thumbs	314
Drag and drop nodes over other elements.....	314
User handles	314
Alignment.....	315
Offset.....	315
Side.....	315
Horizontal and vertical alignments	315
Margin.....	315
Notification for the mouse button clicked.....	315
Appearance	316
Zoom pan	317
Zoom pan status.....	318
Keyboard	319
See Also	321
Tools in Angular Diagram component	321
Drawing tools	321
Shapes	321
Connectors	323
Text	324
Polygon.....	326
Polyline Connector	327
Tool selection	328
Grid lines in Angular Diagram component.....	329
Customize the gridlines visibility.....	329
Appearance	330
Line intervals.....	331
Snapping.....	332
Snap to lines.....	332
Customization of snap intervals.....	333
Snap to objects.....	334
Page settings in Angular Diagram component.....	336
Page size and appearance.....	336
Set background image.....	338

Multiple page and page breaks.....	340
Boundary constraints.....	342
Scroll settings in Angular Diagram component.....	344
Get current scroll status.....	344
Define scroll status.....	344
Update scroll status	345
AutoScroll.....	345
Autoscroll border	346
Scroll limit	347
Scroll padding.....	348
Scrollable Area	349
UpdateViewport.....	350
Data binding in Angular Diagram component	351
Local data	351
Remote Data	353
CRUD	355
Read DataSource.....	355
How to perform Editing at runtime	356
InsertData.....	356
UpdateData	357
DeleteData	358
See Also	359
Automatic layout in Angular Diagram component.....	359
Layout modes.....	359
Hierarchical layout	360
Radial tree layout.....	362
Organizational Chart	365
Symmetric layout	381
Mind Map layout.....	381
Tree Orientation in layout.....	381
Complex hierarchical tree	384
Customize layout.....	390
Accessibility in Angular Diagram component	405
WAI-ARIA attributes.....	406
Aria-label	406

Keyboard interaction	407
Ensuring accessibility	408
See also	408
Commands in Angular Diagram component.....	408
Align	408
Distribute	411
Sizing	413
Clipboard	415
Grouping	417
Z-Order command.....	418
Zoom	423
Nudge command.....	424
Nudge by using arrow keys	424
BringIntoView	425
BringToCenter	425
FitToPage command	426
Command manager.....	427
Custom command	427
Modify the existing command	429
See Also	430
Undo redo in Angular Diagram component.....	430
Undo and redo	430
Undo/redo through shortcut keys	430
Undo/redo through public APIs	431
canLog	433
History change event	435
Stack limit.....	436
Retain selection.....	437
Virtualization in Angular Diagram component	437
Virtualization in Diagram	437
Serialization in Angular Diagram component	438
Save	438
Load.....	439
Prevent default values	439
Export in Angular Diagram component	439

Exporting options	440
File Name	440
Format.....	440
Margin.....	440
Mode.....	440
Region	441
PageSettings.....	441
Content	442
Custom bounds	442
Export diagram with stretch option.....	443
Print.....	443
Limitations.....	444
Tooltip in Angular Diagram component.....	444
Default tooltip.....	444
Common tooltip for all nodes and connectors	445
Tooltip for a specific node/connector.....	446
Tooltip for Ports	448
Tooltip template content	450
Tooltip alignments	452
Tooltip animation.....	454
User handle in Angular Diagram component	456
Alignment.....	456
Fixed user handles	458
Initialization an fixed user handles	458
Customization	459
Customizing node fixed user handle	462
Customizing connector fixed user handle.....	465
Style in Angular Diagram component	468
Customizing the connector end point handle.....	468
Customizing the connector end point handle when connected.....	468
Customizing the connector end point handle when disabled	468
Customizing the bezier connector handle	468
Customizing the bezier connector line	469
Customizing the resize handle	469
Customizing the selector pivot line.....	469

Customizing the selector border.....	469
Customizing the rotate handle	470
Customizing the symbolpalette while hovering	470
Customizing the symbolpalette when selected	470
Customizing the ruler.....	470
Customizing the ruler overlap.....	470
Customizing the text edit.....	471
Customizing the text edit on selection	471
Ruler in Angular Diagram component	471
Adding Rulers to the Diagram.....	471
Customizing the Ruler	472
Layers in Angular Diagram component.....	473
Visible.....	474
Lock	475
Objects	476
AddInfo.....	477
Context menu in Angular Diagram component	481
Customize context menu	481
Template Support for Context menu.....	485
Context menu events.....	487
Symbol palette in Angular Diagram component.....	490
Create symbol palette.....	490
Add palettes to SymbolPalette	490
Customize the palette header	493
Restrict expansion of the palette panel.....	496
Stretch the symbols into the palette	499
Add/Remove symbols to palette at runtime	500
Customize the size of symbols	500
Symbol preview.....	502
Default settings	504
Tooltip for symbols in symbol palette	508
Palette interaction	512
DragEnter	512
DragLeave	512
DragOver	512

See Also	512
Overview in Angular Diagram component.....	512
Create overview	513
Ej1 api migration in Angular Diagram component.....	517
Background	517
Bridging	518
CommandManager	518
Connectors	521
ContextMenu	534
DataSourceSettings.....	536
DefaultSettings.....	543
DrawType	544
EnableAutoScroll.....	544
EnableContextMenu	545
EnablePersistence	545
EnableRtl	545
GetCustomTool	545
Height.....	546
HistoryManager	546
LabelRenderingMode.....	550
Layout.....	551
Locale	553
Nodes	554
SymbolPalette	576
SelectedItems.....	578
SerializationSettings.....	581
How to load EJ1 diagram in EJ2 diagram	581
Tooltip	582
Dialog	584
Getting started with Angular Dialog component.....	584
Getting Started with Angular CLI	584
Installing Syncfusion Popups package	585
Adding Dialog module.....	586
Adding Dialog component	587
dialog-container {.....	588

Adding CSS reference.....	588
Running the application.....	589
Modal Dialog.....	590
Enable header.....	591
Enable footer.....	592
Draggable.....	594
Positioning.....	595
See Also.....	597
Template in Angular Dialog component.....	598
Header.....	598
Footer.....	598
Content.....	598
See Also.....	600
Animation in Angular Dialog component.....	600
Resize in Angular Dialog component.....	602
Localization in Angular Dialog component.....	603
Loading translations.....	604
Accessibility in Angular Dialog component.....	605
WAI-ARIA attributes.....	606
Keyboard interaction.....	606
See Also.....	608
Ensuring accessibility.....	608
See also.....	608
Render a dialog using utility functions in Angular Dialog component.....	608
Alert dialog.....	609
Confirm dialog.....	611
Close utility dialog.....	612
Style in Angular Dialog component.....	613
Customizing the dialog header.....	613
Customizing the dialog content.....	613
Customizing modal dialog overlay.....	614
Customizing the dialog resize icon.....	614
Customizing the dialog close button.....	614
Customizing the dialog footer button.....	615
How to.....	615

Create nested dialog in Angular Dialog component	615
Position the dialog on center of the page on scrolling in Angular Dialog component	617
Load dialog content using ajax in Angular Dialog component.....	618
Render a dialog without header in Angular Dialog component	619
Show dialog with full screen in Angular Dialog component	620
Display a dialog with custom position in Angular Dialog component	622
Prevent closing of modal dialog in Angular Dialog component.....	623
Prevent the focus on the first element in Angular Dialog component.....	625
Prevent opening of the dialog in Angular Dialog component.....	627
Read all the values from dialog on button click in Angular Dialog component.....	629
Customize the dialog appearance in Angular Dialog component.....	632
Close dialog while click on outside of dialog in Angular Dialog component	634
Add an icons to dialog buttons in Angular Dialog component	637
Use dialog in ng routing in Angular Dialog component	640
Render a dialog using ng content in Angular Dialog component	641
Use template driven forms inside dialog in Angular Dialog component	642
Use reactive forms inside dialog in Angular Dialog component	645
Bind the event to the buttons in footer in Angular Dialog component.....	648
Render a dialog using ng template in Angular Dialog component	650
Add a minimize maximize buttons in Angular Dialog component.....	650
Setting max height to the dialog in Angular Dialog component.....	653
Ej1 api migration in Angular Dialog component	655
Accessibility and Localization.....	655
Header.....	656
Footer.....	657
Content	657
Animation.....	658
Draggable and resizing.....	658
Target.....	659
Position	659
Visibility.....	659
Dialog Mode.....	660
Tooltip.....	660
Control State	660
State Maintenance.....	660

Common.....	660
DocumentEditor.....	663
Overview	663
Key Features.....	663
Supported Web platforms	663
Getting started with Angular Document editor component	664
Dependencies.....	664
Setup your development environment	665
Installing Syncfusion DocumentEditor package.....	665
Configuring system JS	666
Adding CSS reference.....	668
Adding Component	669
Frequently Asked Questions	673
Feature module in Angular Document editor component	674
Import in Angular Document editor component.....	676
Import document from local machine	677
Convert word documents into SFDT	678
Compatibility with Microsoft Word	681
See Also	683
Export in Angular Document editor component	683
SFDT export.....	683
Word export.....	684
Template export.....	685
Text export	686
Export as blob	687
See Also	690
Web services in Angular Document editor component.....	690
Which operations require server-side interaction.....	690
Required Web API structure	691
Customize the expected method name.....	692
Add the custom headers to XMLHttpRequest	693
Modify the XMLHttpRequest before request send	693
Server Deployment	694
Word processor server docker image overview in Angular Document editor component.....	694
Provide your license key for activation.....	695

Provide your license key for activation	696
Provide your license key for activation	697
Provide your license key for activation	697
How to deploy word processor server docker container in azure app service in Angular Document editor component	699
How to deploy word processor server docker container in azure kubernetes service in Angular Document editor component	700
How to deploy Word Processor server in Amazon Kubernetes Service	703
How to publish documenteditor web api application in azure app service from visual studio in Angular Document editor component.....	706
Accessibility in Angular Document editor component	708
Keyboard interaction	709
Ensuring accessibility	709
See also	709
Image in Angular Document editor component.....	710
Image resizing	711
Changing size.....	712
Text wrapping style	712
Positioning the image	712
See Also	712
Shapes in Angular Document editor component	712
Supported shapes	712
Text box Shape	713
Shape Resizer	713
Text wrapping style.....	713
Positioning the shape.....	713
Text wrapping style in Angular Document editor component	714
In-Line with Text.....	714
In Front of Text.....	714
Top and Bottom	714
Behind	714
Square	715
Bookmark in Angular Document editor component	715
Add bookmark.....	715
Select Bookmark	715
Delete Bookmark	716

Get Bookmark from document	716
Get Bookmark from selection	716
Replace bookmark content	716
Show or Hide bookmark.....	717
Bookmark Dialog	717
See Also	718
Link in Angular Document editor component	718
Navigate a hyperlink	718
Copy link.....	720
Add hyperlink	720
Customize screen tip.....	722
Remove hyperlink	722
Hyperlink dialog	723
See Also	724
Table in Angular Document editor component	724
Create a table	724
Insert rows	724
Delete table.....	725
Delete row.....	725
Delete column.....	725
Merge cells.....	726
Positioning the table	726
How to work with tables	726
See Also	729
Table of contents in Angular Document editor component.....	729
Inserting table of contents.....	729
Update or edit table of contents	731
See Also	732
Header footer in Angular Document editor component	732
Go to header footer region	733
Link to previous.....	733
Header and footer distance	734
Close header footer region	734
See Also	734
Text format in Angular Document editor component.....	734

Bold	734
Italic.....	735
Underline property	735
Strikethrough property	735
Superscript property	736
Subscript property	736
Size	737
Color.....	737
Font	737
Highlight color.....	737
Toolbar with options for text formatting.....	738
See Also	741
Paragraph format in Angular Document editor component	741
Indentation.....	741
Special indentation	742
Increase indent	742
Decrease indent	742
Text alignment	742
Line spacing and its type	742
Paragraph spacing.....	742
Pagination properties.....	743
Paragraph Border	743
Show or Hide Paragraph marks.....	744
Toolbar with paragraph formatting options	744
See Also	747
Styles in Angular Document editor component.....	747
Styles definition overview	747
Default style	748
Style hierarchy	748
Defining new styles	749
Applying a style	751
Get Styles	751
Modify an existing style	752
List format in Angular Document editor component	752
Create bullet list	752

Create numbered list	753
Clear list.....	753
Working with lists	753
Editing numbered list.....	755
See Also	755
Table format in Angular Document editor component	755
Cell margins.....	755
Background color	756
Cell spacing.....	756
Cell vertical alignment.....	756
Table alignment	757
Cell width	757
Table width	757
Apply borders.....	757
Working with row formatting	758
See Also	759
Section format in Angular Document editor component	759
Page size.....	759
Page margins.....	759
Header distance	759
Footer distance	760
Columns	760
Breaks.....	760
See Also	761
Comments in Angular Document editor component	761
Add a new comment.....	761
Comment navigation.....	761
Delete comment	761
Delete all comment.....	761
Protect the document in comments only mode.....	761
Mention support in Comments.....	763
Fields in Angular Document editor component.....	764
Adding Fields.....	765
Update fields.....	765
Get field info	765

Set field info	766
See Also	766
Form fields in Angular Document editor component.....	766
Insert form field	766
Get form field names	767
Get form field properties	767
Set form field properties.....	767
Export form field data	768
Import form field data	768
Reset form fields	768
Protect the document in form filling mode	769
Clipboard in Angular Document editor component	770
Copy	770
Cut.....	770
Paste.....	770
Local paste (copy/paste within control)	770
Paste with formatting	772
See Also	773
History in Angular Document editor component	773
Enable or disable history.....	773
Undo and redo	774
Stack size	774
See Also	774
Find and replace in Angular Document editor component	774
Options pane.....	775
Search.....	776
Search results	777
SearchResultsChange event.....	778
Customize find and replace.....	779
Keyboard shortcut in Angular Document editor component	781
Text formatting	781
Paragraph formatting.....	782
Clipboard.....	782
Keyboard shortcut to navigate around the document	782
Keyboard shortcut to extend selection.....	783

Find and Replace	783
Create, Save and Print document	783
Edit Operation	784
Insert special characters	784
Dialog	784
See Also	784
Scrolling zooming in Angular Document editor component	784
Zooming	788
Page Fit Type	788
Zoom option using UI	789
Print in Angular Document editor component	794
Improve print quality	797
Print using window object	798
Page setup	799
Dialog in Angular Document editor component	800
Font Dialog	800
Paragraph dialog	801
Table dialog	802
Bookmark dialog	803
Hyperlink dialog	804
Table of contents dialog	805
Styles Dialog	806
Style dialog	807
List dialog	808
Borders and shading dialog	809
Table options dialog	810
Table properties dialog	811
Page setup dialog	812
See Also	813
Right to left in Angular Document editor component	813
Chart in Angular Document editor component	820
Supported Chart Types	839
Document management in Angular Document editor component	839
Set current user	839
Highlighting the text area	840

Restrict Editing Pane	840
See Also	846
Spell check in Angular Document editor component	846
Features	847
Enable SpellCheck	847
Disable SpellCheck	847
Spell check settings	847
Context menu.....	849
Global local in Angular Document editor component	851
Localization	851
Document Editor	851
Color Picker	856
Notes in Angular Document editor component	856
Insert footnotes	856
Insert endnotes.....	858
Update or edit footnotes and endnotes	859
Collaborative Editing (preview).....	860
Prerequisites	860
How to enable collaborative editing in client side.....	860
How to enable collaborative editing in ASP.NET Core.....	864
How to perform Scaling in Collaborative Editing	871
Collaborative Editing (preview).....	872
Prerequisites	872
How to enable collaborative editing in client side.....	872
How to enable collaborative editing in Java	875
How to perform Scaling in Collaborative Editing	880
View in Angular Document Editor Component.....	882
Web Layout	882
Ruler	882
Heading Navigation Pane.....	883
How To	884
Override the keyboard shortcuts in Angular Document editor component	884
Customize context menu in Angular Document editor component.....	886
Customize tool bar in Angular Document editor component	892
Change document view in Angular Document editor component.....	894

Open default document in Angular Document editor component	896
Default read only in Angular Document editor component	900
Open document by address in Angular Document editor component	905
Deploy document editor component for mobile in Angular Document editor component	907
Disable optimized text measuring in Angular Document editor component	908
Get the selected content in Angular Document editor component	910
Set default format in document editor in Angular Document editor component	913
Show hide spinner in Angular Document editor component	917
Resize document editor in Angular Document editor component	920
Export document as pdf in Angular Document editor component	923
Customize font family drop down in Angular Document editor component	927
Auto save document in document editor in Angular Document editor component	928
Retrieve the bookmark content as text in Angular Document editor component	932
Get current word in Angular Document editor component	936
Insert page number and navigate to page in Angular Document editor component	938
Move selection to specific position in Angular Document editor component	941
Disable header and footer edit in document editor in Angular Document editor component	943
Insert text in current position in Angular Document editor component	947
Change the cursor color in document editor in Angular Document editor component	951
Hide tool bar and properties pane in Angular Document editor component	951
Insert text or image in table programmatically in Angular Document editor component	953
Change the default search highlight color in Angular Document editor component	956
How to optimize the SFDT file	957
How to disable auto focus in Syncfusion Angular Document Editor component	958
How to disable drag and drop in document editor in Angular Document editor component	958
Enable ruler	959
Customize color picker in Angular Document editor component	961
DropDownButton	962
Getting started with Angular Drop down button component	962
Dependencies	962
Setup Angular environment	962
Create an Angular application	962
Installing Syncfusion DropDownButton package	963
Adding DropDownButton module	963
Adding Syncfusion DropDownButton component	964

Adding CSS reference.....	965
Running the application.....	965
Icons in Angular Drop down button component.....	966
DropDownButton icons.....	966
Vertical button.....	969
See Also.....	970
Popup items in Angular Drop down button component.....	970
Icons.....	970
Navigations.....	971
Template.....	972
Separator.....	975
See Also.....	976
Accessibility in Angular Drop down button component.....	976
WAI-ARIA attributes.....	977
Keyboard interaction.....	978
Ensuring accessibility.....	978
See also.....	978
How To.....	978
Change caret icon in Angular Drop down button component.....	978
Create dropdownbutton with rounded corner in Angular Drop down button component.....	979
Create right to left dropdownbutton in Angular Drop down button component.....	980
Customize icon and width in Angular Drop down button component.....	981
Disable a dropdownbutton in Angular Drop down button component.....	982
Group popup items with listview component in Angular Drop down button component.....	983
Hide dropdown arrow in Angular Drop down button component.....	984
Open a dialog on popup item click in Angular Drop down button component.....	985
Position popup open in Angular Drop down button component.....	987
Underline a character in the item text in Angular Drop down button component.....	988
DropDownList.....	989
Getting started with Angular Drop down list component.....	989
Dependencies.....	989
Setup angular environment.....	989
Create a new application.....	989
Installing Syncfusion DropDownList package.....	990
Registering DropDownList module.....	990

Adding CSS reference.....	991
Adding DropDownList component.....	991
Binding data source	992
Running the application	992
Configure the popup list	993
Two-way binding.....	994
See Also	995
Data binding in Angular Drop down list component	995
Binding local data	995
Binding remote data	998
Data binding using Async pipe	999
See Also	1001
Value binding in DropDownList	1001
Primitive Data Types	1001
Object Data Types	1002
Templates in Angular Drop down list component	1003
Item template	1003
Value template.....	1004
Group template.....	1006
Header template	1007
Footer template	1008
No records template	1009
Action failure template	1010
See Also	1011
Virtualization in DropDown List	1011
Binding local data	1012
Binding remote data	1013
Grouping	1014
Filtering with Virtualization.....	1016
Grouping in Angular Drop down list component.....	1017
Customization	1018
See Also	1018
Filtering in Angular Drop down list component.....	1018
Limit the minimum filter character.....	1019
Change the filter type	1021

Case sensitive filtering	1022
Diacritics Filtering.....	1023
See Also	1024
Localization in Angular Drop down list component.....	1024
Loading translations.....	1025
See Also	1026
Style in Angular Drop down list component	1026
Customizing the appearance of wrapper element	1026
Customizing the dropdown icon's color	1026
Customizing the focus color	1026
Customizing the outline theme's focus color	1027
Customizing the disabled component's text color	1027
Customizing the float label element's focusing color	1027
Customizing the color of the placeholder text	1028
Customizing the background color of focus, hover, and active item's	1028
Customizing the appearance of pop-up element	1028
Adding mandatory asterisk to placeholder and float label.....	1028
Accessibility in Angular Drop down list component	1029
WAI-ARIA attributes.....	1030
Keyboard Interaction	1030
Ensuring accessibility	1032
See also	1032
Form support in Angular Drop down list component.....	1032
Template-Driven Forms	1032
Reactive Forms.....	1033
How To	1035
Add item in Angular Drop down list component	1035
Cascading in Angular Drop down list component.....	1036
Clear item in Angular Drop down list component	1038
Close popup in Angular Drop down list component.....	1039
Group header in Angular Drop down list component	1040
Highlight filtering in Angular Drop down list component.....	1041
Icons support in Angular Drop down list component	1042
Incremental search in Angular Drop down list component.....	1043
Modify data in Angular Drop down list component	1043

Remote data bind in Angular Drop down list component	1044
Remove item in Angular Drop down list component	1046
Search on filtering in Angular Drop down list component	1047
Tooltip in Angular Drop down list component.....	1048
Value change in Angular Drop down list component	1050
Value support in Angular Drop down list component	1051
Ej1 api migration in Angular Drop down list component.....	1051
DataBinding.....	1051
Filtering	1052
Template	1053
Virtual Scrolling.....	1053
Applying CSS.....	1054
Sorting	1054
Popup.....	1054
Placeholder	1056
Grouping	1056
Accessibility.....	1056
Miscellaneous	1056
Selection.....	1057
Common.....	1058
Dropdown Tree	1059
Getting started with Angular Drop down tree component	1059
Dependencies.....	1059
Setup angular environment	1060
Create an Angular Application	1060
Installing Syncfusion DropDownTree package.....	1060
Registering Dropdown Tree Module.....	1061
Add Dropdown Tree component	1062
Binding data source	1062
Run the application	1063
Data binding in Angular Drop down tree component	1065
Local data	1065
Remote data.....	1068
Templates in Angular Drop down tree component	1070
Item template	1070

Header template	1071
Footer template	1072
No records template	1073
Action failure template	1074
Custom template to show selected items in input	1075
Checkbox in Angular Drop down tree component	1077
Auto Check	1079
Select All	1080
Localization in Angular Drop down tree component	1081
Accessibility in Angular Dropdown Tree component	1081
WAI-ARIA attributes	1082
Keyboard interaction	1083
Ensuring accessibility	1084
See also	1084
FileManager	1084
Getting started with Angular File manager component	1084
Prerequisites	1084
Setting up angular project	1084
Adding Dependencies	1085
Installing Syncfusion FileManager package	1085
Adding style sheet to the application	1086
Adding File Manager module	1086
Adding Syncfusion component	1087
Run the application	1088
File Download support	1089
File Upload support	1089
Image Preview support	1090
Injecting feature modules	1090
Switching initial view of the File Manager	1092
Maintaining component state on page reload	1093
Rendering component in right-to-left direction	1094
Specifying the current path of the File Manager	1095
User interface in Angular File manager component	1096
Toolbar	1096
Files and folders navigation	1097

View	1098
Context menu.....	1099
File operations in Angular File manager component.....	1099
Folder Upload support	1100
File operation request and response Parameters	1105
File request and response contents.....	1121
Action Buttons	1122
Views in Angular File manager component	1123
Largelcons View	1123
Details View.....	1124
Customization in Angular File manager component	1125
Context menu customization.....	1125
Details view customization	1126
Navigation pane customization	1127
Show/Hide file extension	1128
Show/Hide hidden items.....	1129
Show/Hide thumbnail images in large icons view	1130
Toolbar customization	1131
Upload customization	1132
Tooltip customization	1133
Multiple selection in Angular File manager component.....	1134
Drag and drop in Angular File manager component	1135
File system provider in Angular File manager component	1136
ASP.NET Core file system provider	1137
ASP.NET MVC 5 file system provider	1138
ASP.NET Core Azure cloud file system Provider	1139
ASP.NET MVC 5 Azure cloud file system Provider	1140
ASP.NET Core Amazon S3 cloud file provider	1142
ASP.NET MVC Amazon S3 cloud file provider.....	1143
File Transfer Protocol file system provider	1144
SQL database file system provider.....	1145
NodeJS file system provider.....	1147
Google Drive file system provider.....	1149
Firebase Realtime Database file system provider.....	1150
Localization in Angular File manager component.....	1156

Virtualization in Angular File Manager component.....	1162
Module Injection.....	1162
Enable Virtualization	1163
Limitations for Virtualization	1164
Accessibility in Angular File Manager component.....	1164
WAI-ARIA attributes.....	1165
Keyboard interaction	1166
Ensuring accessibility	1167
See also	1167
Access control in Angular File manager component	1167
Access Rules	1168
Permissions	1169
How To	1171
Adding custom item to context menu in Angular File manager component	1171
Adding custom item to toolbar in Angular File manager component	1172
Enable disable toolbar item in Angular File manager component	1174
Initialize filemanager using systemjs in Angular File manager component.....	1175
Customize custom thumbnail in Angular File manager component	1180
Nested items in Angular File manager component	1180
Create the custom file provider using NodeJS.....	1184
Perform custom sorting in Angular FileManager component	1203
Floating Action Button	1205
Getting started with Angular Floating action button component	1205
Dependencies.....	1205
Setup Angular Environment.....	1205
Create an Angular Application	1205
Installing Syncfusion Buttons package.....	1206
Adding Floating Action Button module.....	1206
Adding Syncfusion Floating Action Button Component.....	1207
Adding CSS reference.....	1207
Running the application	1207
Click event	1208
Icons in Angular Floating action button component	1209
FAB with icon	1209
FAB with icon and text	1209

Styles in Angular Floating action button component	1212
FAB styles	1212
Styles customization	1213
Show text on hover	1213
Positions in Angular Floating action button component	1214
Custom position	1219
Events in Angular Floating action button component	1220
created	1220
onclick	1220
Accessibility in Angular Floating action button component	1221
WAI-ARIA attributes	1222
Keyboard interaction	1223
Ensuring accessibility	1223
See also	1223

DateTimePicker

Getting started with Angular Datetimepicker component

The following section explains the steps required to create a simple DateTimePicker component and also it demonstrates the basic usage of the DateTimePicker.

Dependencies

Install the below required dependency package in order to use the `DateTimePicker` component in your application.

```
`javascript
|-- @syncfusion/ej2-angular-calendars
|-- @syncfusion/ej2-angular-base
|-- @syncfusion/ej2-base
|-- @syncfusion/ej2-calendars
|-- @syncfusion/ej2-lists
|-- @syncfusion/ej2-inputs
|-- @syncfusion/ej2-splitbuttons
|-- @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-datetimepicker
`,`
```

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-datetimepicker --style=scss
`,`
```

Navigate to the created project folder.

```
`bash
cd syncfusion-angular-datetimepicker
`
```

Installing Syncfusion DateTimePicker 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-calendars](#) package to the application.

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

```
`bash
npm install @syncfusion/ej2-angular-calendars@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-calendars:"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 DateTimePicker module

Import DateTimePicker module into Angular application(`src/app/app.module.ts`) from the package `@syncfusion/ej2-angular-calendars`.

```
`javascript
import { NgModule } from "@angular/core";
```

```
import { BrowserModule } from "@angular/platform-browser";
// import the DateTimePickerModule for the DateTimePicker component
import { DateTimePickerModule } from "@syncfusion/ej2-angular-calendars";
import { AppComponent } from "./app.component";
@NgModule({
  //declaration of DateTimePickerModule into NgModule
  imports: [BrowserModule, DateTimePickerModule],
  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-lists/styles/material.css';
@import '../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-calendars/styles/material.css';
@import '../node_modules/@syncfusion/ej2-angular-calendars/styles/material.css';
`
```

If you want to refer the combined component styles, please make use of our [CRG](#) (Custom Resource Generator) in your application.

Adding DateTimePicker component

Modify the template in [src/app/app.component.ts] file to render the Angular DateTime Picker component.

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

```
`javascript
import { Component } from "@angular/core";
```



```
@Component({
  selector: 'app-root',
  template: `<!-- To Render DateTimePicker -->
  <ejs-datetimepicker></ejs-datetimepicker>`
})
export class AppComponent {}
`
```

Running the application

After completing the configuration required to render a basic DateTimePicker, 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 } from '@angular/forms'
import { DateTimePickerModule } from '@syncfusion/ej2-angular-calendars'
import { Component } from '@angular/core';
@Component({
  imports: [
    DateTimePickerModule,
    FormsModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-datetimepicker></ejs-datetimepicker>`
})
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));
```

Setting the min and max

The minimum and maximum date time can be defined with the help of `min` and `max` property. The following example demonstrates to set the `min` and `max` on initializing the DateTimePicker. To know more about range restriction in Angular DateTime Picker, please refer this [page](#).

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule } from '@angular/forms'
import { DateTimePickerModule } from '@syncfusion/ej2-angular-calendars'
import { Component } from '@angular/core';
@Component({
  imports: [
    DateTimePickerModule,
    FormsModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-datetimepicker [min]='minDate' [max]='maxDate'></ejs-datetimepicker>`
})
export class AppComponent {
  public month: number = new Date().getMonth();
  public fullYear: number = new Date().getFullYear();
  public minDate: Date = new Date(this.fullYear, this.month, 22, 12);
  public maxDate: Date = new Date(this.fullYear, this.month, 25, 17);
  constructor() {}
}
```

MAIN.TS

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

If the value of `min` or `max` properties changed through code behind, then you have to update the `value` property to set within the range.

Note: You can refer to our [Angular DateTime Picker](#) feature tour page for its groundbreaking feature representations. You can also explore our [Angular DateTime Picker example](#) that shows how to render the DateTime Picker in Angular.

See Also

- [Render DateTimePicker with specific culture](#)
- [How to achieve validation with DateTimePicker](#)
- [How to achieve two-way binding with DateTimePicker](#)
- [Reactive forms with DateTimePicker](#)
- [Template-driven forms with DateTimePicker](#)

Globalization in Angular Datetimepicker component

Globalization is the combination of internalization and localization. You can adapt the component to various languages by parsing and formatting the date or number [Internationalization](#) and also add culture specific customization and translation to the text [localization](#).

By default, the date format, week, month, time format and meridian names are specific to the American English culture. It utilizes the [Essential JavaScript 2 Internationalization](#) package to parse and format the date object based on the culture by using the official [UNICODE CLDR](#) JSON data. It provides the `loadCldr` method to load culture specific CLDR JSON data. To use a different culture other than English, follow the steps below:

- Install the `CLDR-Data` package by using the following command (installs all the CLDR JSON data). To know more about CLDR-Data refer to the [CLDR-Data](#) link.

```
npm install cldr-data --save
```

Once the package installed, you can find the culture specific JSON data under the location `/node_modules/cldr-data`.

- Now import the installed CLDR JSON data into the `app.component.ts` file.
- Now use the `loadCldr` method to load the culture specific CLDR JSON data from the installed location to `app.component.ts` file.
- DateTimePicker displayed `Sunday` as the first day of week based on default culture ("en-US"). If you want to display the DateTimePicker with loaded culture's first day of week, you need to import `weekdata.json` file from the `cldr-data/supplemental` as given in the code example.

```
`typescript
import { loadCldr } from "@syncfusion/ej2-base";
declare var require: any;
loadCldr(
  require("cldr-data/main/de/numbers.json"),
  require("cldr-data/main/de/ca-gregorian.json"),
  require("cldr-data/supplemental/numberingSystems.json"),
  require("cldr-data/main/de/timeZoneNames.json"),
  require('cldr-data/supplemental/weekdata.json') // To load the culture based first day of week
);
```

The `Localization` library allows you to localize default text content of the DateTimePicker. The DateTimePicker component has static text for **today** feature that can be changed to other cultures (Arabic, Deutsch, French, etc.) by defining the [locale](#) value and translation object.

Locale keywords | Text

today | Name of the button to choose Today date.

placeholder | Hint to describe expected value in input element.

- Before changing to a culture other than English, ensure that locale text for the concerned culture is loaded through load method of L10n class.

```
`typescript
//Load the L10n, loadCldr from ej2-base
import { loadCldr, L10n } from "@syncfusion/ej2-base";
//load the locale object to set the localized placeholder value
L10n.load({
  de: {
    datetimepicker: {
      placeholder: "Wählen Sie ein Datum und eine Uhrzeit aus",
      today:"heute"
    }
  }
});
`
```

- Set the culture by using the locale property.

In the following code example, the DateTimePicker is initialized in German culture with corresponding localized text.

The following example demonstrates the DateTimePicker in German culture.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DateTimePickerModule } from '@syncfusion/ej2-angular-calendars'
import { Component } from '@angular/core';
import { DateTimePickerComponent } from '@syncfusion/ej2-angular-calendars';
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 gregorian from './ca-gregorian.json';
import * as numbers from './numbers.json';
import * as detimeZoneNames from './timeZoneNames.json';
loadCldr(numberingSystems, gregorian, numbers, detimeZoneNames);
@Component({
  imports: [
    DateTimePickerModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-datetimepicker [value]='date' locale='de'></ejs-
datetimepicker>`
```

```

    })
    export class AppComponent {
        public date: Date = new Date("12/11/2017 1:00 AM");
        ngOnInit(): void {
            /*loads the localization text*/
            L10n.load({
                de: {
                    datetimepicker: {
                        placeholder: "Wählen Sie ein Datum und eine Uhrzeit aus",
                        today: "heute"
                    }
                }
            });
        }
        constructor() {}
    }
}

```

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

The DateTimePicker supports RTL (right-to-left) functionality for languages like Arabic and Hebrew to displays the text in the right-to-left direction.

Use `enableRtl` property to set the RTL direction.

The following code example initialize the DateTimePicker component in Arabic culture and also explains how to set the localized text to the placeholder using `load` method of `L10n` class.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DateTimePickerModule } from '@syncfusion/ej2-angular-calendars'
import { Component } from '@angular/core';
import { DateTimePickerComponent } from '@syncfusion/ej2-angular-calendars';
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 gregorian from './ca-gregorian.json';
import * as numbers from './numbers.json';
import * as artimeZoneNames from './timeZoneNames.json';
loadCldr(numberingSystems, gregorian, numbers, artimeZoneNames);
@Component({
    imports: [
        DateTimePickerModule
    ],
    standalone: true,
    selector: 'app-root',
})

```

```

    template: `<ejs-datetimepicker [enableRTL]='true' locale='ar'></ejs-
datetimepicker>`
  })
  export class AppComponent {
    ngOnInit(): void {
      L10n.load({
        'ar': {
          'datetimepicker': {
            placeholder: 'حدد التاريخ والوقت',
            today: "اليوم"
          }
        }
      });
    }
    constructor() {}
  }
}

```

MAIN.TS

```

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

```

Strict mode in Angular Datetimepicker component

The **strictMode** is an act, that allows the user to enter only the valid date and time within the specified min/max range in textbox.

If the input entered is invalid, then the component will stay with the previous value.

Else, if the datetime is out of range, then the component will set the datetime to the min/max value.

The following example demonstrates the DateTimePicker in **strictMode** with min/max range of 5/5/2019 2:00 AM to 5/25/2019 2:00 AM. Here, it allows to enter only the valid date and time within the specified range. If you are trying to enter the out-of-range value as like 5/28/2019, then the value will set to the **max** value as 5/25/2019 2:00 AM. Since the value 28 is greater than to **max** value of 25. Or else if you are trying to enter the invalid date, then the value will stay with the previous value.

The following example demonstrates the DateTimePicker with **strictMode true**.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule } from '@angular/forms'
import { DateTimePickerModule } from '@syncfusion/ej2-angular-calendars'
import { Component, ViewEncapsulation } from '@angular/core';
import { DateTimePickerComponent } from '@syncfusion/ej2-angular-calendars';
@Component({
  imports: [
    DateTimePickerModule,
    FormsModule
  ],
  standalone: true,

```

```

    selector: 'app-root',
    template: `<ejs-datetimepicker [strictMode]='true' [value]='date'
[min]='minDate' [max]='maxDate'></ejs-datetimepicker>`
  })
  export class AppComponent {
    public date: Date = new Date('5/28/2019 2:00 AM');
    public minDate: Date = new Date('5/5/2019 2:00 AM');
    public maxDate: Date = new Date('5/25/2019 2:00 AM');
    constructor() {
    }
  }
}

```

MAIN.TS

```

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

```

By default, the DateTimePicker act in strictMode **false** state, that allows to enter the invalid or out-of-range datetime in textbox.

If the datetime is out-of-range or invalid, then the model value will be set to **out of range** datetime value or **null** respectively with highlighted **error** class to indicates the datetime is out of range or invalid.

The following example demonstrates the **strictMode** as **false**. Here, it allows to enter the valid or invalid value in textbox.

If you are entering the out-of-range or invalid datetime value, then the model value will be set to **out of range** datetime value or **null** respectively with highlighted **error** class to indicates the datetime is out of range or invalid.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule } from '@angular/forms'
import { DateTimePickerModule } from '@syncfusion/ej2-angular-calendars'
import { Component } from '@angular/core';
import { DateTimePickerComponent } from '@syncfusion/ej2-angular-calendars';
@Component({
  imports: [

    DateTimePickerModule,
    FormsModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-datetimepicker placeholder='Select a date and time'
[value]='date' [min]='minDate' [max]='maxDate'></ejs-datetimepicker>`
})
export class AppComponent {
  public date: Date = new Date("5/25/2017 4:00 PM");
  public minDate: Date = new Date("5/5/2017 2:00 PM");
  public maxDate: Date = new Date("5/25/2017 3:00 PM");
}

```

```
constructor() {}
}
```

MAIN.TS

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

If the value of `min` or `max` properties changed through code behind, then you have to update the `value` property to set within the range.

Date time range in Angular Datetimepicker component

DateTimePicker provides an option to select a date and time value within a specified range by using the `min` and `max` properties.

Always the min value has to be lesser than the max value.

When the min and max properties are configured and the selected datetime value is out-of-range or invalid, then the model value will be set to `out of range datetime value` or `null` respectively with highlighted `error` class to indicates the datetime is out of range or invalid.

The value property depends on the min/max with respect to `strictMode` property.

The below example allows selecting a date within the range from 7th to 27th day in a month.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule } from '@angular/forms'
import { DateTimePickerModule } from '@syncfusion/ej2-angular-calendars'
import { Component } from '@angular/core';
@Component({
  imports: [
    DateTimePickerModule,
    FormsModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-datetimepicker [value]='date' [min]='minDate'
[max]='maxDate'></ejs-datetimepicker>`
})
export class AppComponent {
  public today: Date = new Date();
  public currentYear: number = this.today.getFullYear();
  public currentMonth: number = this.today.getMonth();
  public currentDay: number = this.today.getDate();
  public currentHour: number = this.today.getHours();
  public currentMinute: number = this.today.getMinutes();
  public currentSecond: number = this.today.getSeconds();
  public date: Date = new Date(new Date().setDate(14));
  public minDate: Date = new
Date(this.currentYear, this.currentMonth, 7, 0, 0, 0);
```



```
public maxDate: Date = new
Date(this.currentYear, this.currentMonth, 27, this.currentHour, this.currentMinu
te, this.currentSecond);
constructor() {}
}
```

MAIN.TS

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

If the value of `min` or `max` properties changed through code behind, then you have to update the `value` property to set within the range.

Date time masking in Angular Datetimepicker component

The DateTimePicker has built-in support to masking the date value, when `enableMask` property set as `true`.

To use mask support, inject the MaskedDateTime module in the DateTimePicker.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { DateTimePickerModule } from '@syncfusion/ej2-angular-calendars';
import { Component } from '@angular/core';
import { MaskedDateTimeService } from '@syncfusion/ej2-angular-calendars';
@Component({
  imports: [
    DateTimePickerModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `
    <ejs-datetimepicker [enableMask]="enableMaskSupport"></ejs-
datetimepicker>
  `,
  providers: [MaskedDateTimeService],
})
export class AppComponent {
  constructor() {
    public enableMaskSupport: 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));
```

The mask pattern is defined based on the provided date format to the component. If the format is not specified, the mask pattern is formed based on the default format of the current culture.

| **Keys | Actions |**

| --- | --- |

| **Up / Down arrows | To increment and decrement the selected portion of the date and time. |**

| **Left / Right arrows and Tab | To navigate the selection from one portion to next portion |**

The following example demonstrates default and custom format of DateTimePicker component with mask.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { DateTimePickerModule } from '@syncfusion/ej2-angular-calendars';
import { Component } from '@angular/core';
import { MaskedDateTimeService } from '@syncfusion/ej2-angular-calendars';
@Component({
  imports: [
    DateTimePickerModule
  ],
  standalone: true,
  selector: 'app-root',
  templateUrl: './format.html',
  providers: [MaskedDateTimeService],
})
export class AppComponent {
  constructor() {
    public format: string = "dd/MM/yyyy";
    public enableMaskSupport: 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));
```

Configure Mask Placeholder

You can change mask placeholder value through property `maskPlaceholder`. By default , it takes the full name of date and time co-ordinates such as `day`, `month`, `year`, `hour` etc.

While changing to a culture other than `English`, ensure that locale text for the concerned culture is loaded through load method of L10n class for mask placeholder values like below.

```
`typescript
```

```
//Load the L10n from ej2-base
```

```
import { L10n } from '@syncfusion/ej2-base';
```

//load the locale object to set the localized mask placeholder value

```
L10n.load({
  'de': {
    datetimepicker: { day: 'Tag', month: 'Monat', year: 'Jahr', hour: 'Stunde', minute: 'Minute',
      second: 'Sekunden' }
  }
});
`
```

The following example demonstrates default and customized mask placeholder value.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { DateTimePickerModule } from '@syncfusion/ej2-angular-calendars';
import { Component } from '@angular/core';
import { MaskedDateTimeService } from '@syncfusion/ej2-angular-calendars';
@Component({
  imports: [
    DateTimePickerModule
  ],
  standalone: true,
  selector: 'app-root',
  templateUrl: './maskplaceholder.html',
  providers: [MaskedDateTimeService],
})
export class AppComponent {
  constructor() {
    public enableMaskSupport: boolean = true;
    public maskPlaceholderValue: Object = {day: 'd', month: 'M', year: 'y',
      hour: 'h', minute: 'm', second: 's'}
  }
}
```

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 Datetimepicker component

The DateTimePicker is available for UI customization that can be achieved by using available properties and events in the component.

Day and Time Cell format

The DateTimePicker is available for UI customization based on your application requirements.

It can be achieved by using [renderDayCell](#) event that provides an option to customize each day cell on rendering.

The following example disables the weekends of every month by using `renderDayCell` event.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule } from '@angular/forms'
import { DateTimePickerModule } from '@syncfusion/ej2-angular-calendars'
import { Component } from '@angular/core';
import { DateTimePickerComponent } from '@syncfusion/ej2-angular-calendars';
@Component({
  imports: [

    DateTimePickerModule,
    FormsModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-datetimepicker placeholder='Select a date and time'
(renderDayCell)='onRenderCell($event)'></ejs-datetimepicker>`
})
export class AppComponent {
  onRenderCell(args: any) {
    /*Apply selected format to the component*/
    if (args.date.getDay() == 0 || args.date.getDay() == 6) {
      //sets isDisabled to true to disable the date.
      args.isDisabled = true;
    }
  }
  constructor() {}
}
```

MAIN.TS

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

Adding mandatory asterisk to placeholder and float label

You can add a mandatory asterisk(*) to placeholder and float label using `.e-input-group.e-control-wrapper.e-float-input .e-float-text::after` class.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule } from '@angular/forms'
import { DateTimePickerModule } from '@syncfusion/ej2-angular-calendars'
import { Component } from '@angular/core';
@Component({
  imports: [

    DateTimePickerModule,
    FormsModule
  ],
```

```
standalone: true,
  selector: 'app-root',
  template: `<ejs-datetimepicker floatLabelType="auto" placeholder="Enter
date"></ejs-datetimepicker>`
})
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));
```

See Also

- [How to disable the DateTimePicker component](#)
- [How to customize the DateTimePicker day header](#)

Accessibility in Angular Datetimepicker component

The DateTimePicker component followed the accessibility guidelines and standards, including [ADA](#), [Section 508](#), [WCAG 2.2](#) standards, and [WCAG roles](#) that are commonly used to evaluate accessibility.

The accessibility compliance for the DateTimePicker component is outlined below.

| Accessibility Criteria | Compatibility |


| -- | -- |

| [WCAG 2.2 Support](#) |  |
src="https://cdn.syncfusion.com/content/images/documentation/partial.png" alt="Intermediate"> |

| [Section 508 Support](#) |  |
src="https://cdn.syncfusion.com/content/images/documentation/partial.png" alt="Intermediate"> |

| [Screen Reader Support](#) |  |
src="https://cdn.syncfusion.com/content/images/documentation/full.png" alt="Yes"> |

| [Right-To-Left Support](#) |  |
src="https://cdn.syncfusion.com/content/images/documentation/full.png" alt="Yes"> |

| [Color Contrast](#) |  |
alt="Yes"> |

| [Mobile Device Support](#) |  |
src="https://cdn.syncfusion.com/content/images/documentation/full.png" alt="Yes"> |

| [Keyboard Navigation Support](#) |  |
src="https://cdn.syncfusion.com/content/images/documentation/full.png" alt="Yes"> |

| [Accessibility Checker Validation](#) |  |
src="https://cdn.syncfusion.com/content/images/documentation/full.png" alt="Yes"> |

```
| Axe-core Accessibility Validation |  |  
<style>  
.post .post-content img {  
display: inline-block;  
margin: 0.5em 0;  
}  
</style>  
<div> - All  
features of the component meet the requirement.</div>  
<div> - Some features of the component do not meet the requirement.</div>  
<div> - The component does not meet the requirement.</div>
```

WAI-ARIA attributes

The Web accessibility defines a way to make web content and web applications more accessible to disabled people. It especially helps the dynamic content change and advanced user interface controls developed with Ajax, HTML, JavaScript, and related technologies.

DateTimePicker provides built-in compliance with the [WAI-ARIA](#) specifications. WAI-ARIA supports is achieved through the attributes like `aria-expanded`, `aria-disabled`, `aria-activedescendant` applied to the input element.

To know about the accessibility of Calendar refer to the Calendar's [Accessibility](#) section.

It helps to provide information about the widget for assistive technology to the disabled person in screen reader.

- **Aria-expanded:** attributes indicates the state of a collapsible element.
- **Aria-disabled:** attribute indicates the disabled state of this DateTimePicker component.
- **Aria-activedescendent:** attribute helps in managing the current active child of the DateTimePicker component.

Keyboard Interaction

You can use the following keys to interact with the DateTimePicker.

The component implements the keyboard navigation support by following the [WAI-ARIA practices](#).

DateTimePicker supports the below list of shortcut keys.

Input Navigation

Before opening the popup, use the below list of keys to `DateTimePicker` component the popup element.

Press	To do this
-----	-----

Alt + Down Arrow	Open the select popup
------------------	-----------------------

Alt + Down Arrow + Alt + Down Arrow	Toggle between two popup
-------------------------------------	--------------------------

Calendar Navigation

Use the below list of keys to interact with the Calendar after the DatePicker popup has opened.

Press	To do this
-----	-----
Upper Arrow	Focus the previous week date.
Down Arrow	Focus the next week date.
Left Arrow	Focus the previous date.
Right Arrow	Focus the next date.
Home	Focus the first date in the month.
End	Focus the last date in the month.
Page Up	Focus the same date in the previous month.
Page Down	Focus the same date in the next month.
Enter	Select the currently focused date.
Shift + Page Up	Focus the same date in the previous year.
Shift + Page Down	Focus the same date in the previous year.
Control + Upper Arrow	Moves into the inner level of view like month-year, year-decade
Control + Down Arrow	Moves out from the depth level view like decade-year, year-month
Control + Home	Focus the starting date in the current year.
Control + End	Focus the ending date in the current year.

Use the below list of shortcut keys to interact with the TimePicker after the TimePicker Popup has opened.

Press	To do this
-----	-----
Upper Arrow	Navigate and select the previous item.
Down Arrow	Navigate and select the next item.
Left Arrow	Move the cursor towards arrow key pressed direction.
Right Arrow	Move the cursor towards arrow key pressed direction.
Home	Navigate and select the first item.
End	Navigate and select the last item.
Enter	Select the currently focused item and close the popup.

| Alt + Upper Arrow | Close the popup. |

| Alt + Down Arrow | Open the popup. |

| Esc | Close the popup. |

To focus the DateTimePicker component use the alt+t keys.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule } from '@angular/forms'
import { DateTimePickerModule } from '@syncfusion/ej2-angular-calendars'
import { Component, HostListener, ViewChild } from "@angular/core";
import { DateTimePickerComponent } from "@syncfusion/ej2-angular-calendars";
@Component({
  imports: [
    DateTimePickerModule,
    FormsModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-datetimepicker #ejDateTimePicker placeholder='Select a date and time'></ejs-datetimepicker>`
})
export class AppComponent {
  @ViewChild("ejDateTimePicker") ejDateTimePicker?: DateTimePickerComponent;
  @HostListener("document:keyup", ["$event"])
  handleKeyboardEvent(event: KeyboardEvent) {
    if (event.altKey && event.keyCode === 84 /* t */) {
      // press alt+t to focus the component.
      this.ejDateTimePicker?.focusIn();
    }
  }
  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 DateTimePicker component's accessibility levels are ensured through an [accessibility-checker](#) and [axe-core](#) software tools during automated testing.

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

See also

- [Accessibility in Syncfusion Angular components](#)

Style appearance in Angular Datetimepicker component

The following content provides the exact CSS structure that can be used to modify the control's appearance based on the user preference.

Customizing the appearance of DateTimePicker wrapper element

Use the following CSS to customize the appearance of wrapper element.

```
`css
/ To specify height and font size /
.e-input-group input.e-input, .e-input-group.e-control-wrapper input.e-input {
font-size: 20px;
height: 40px;
}
`
```

Customizing the DateTimePicker icons element

Use the following CSS to customize the DateTimePicker icons element

```
`css
/ To specify background color and font size /
.e-datetime-wrapper .e-input-group-icon.e-date-icon, .e-datetime-wrapper .e-input-group-icon.e-time-
icon {
font-size: 16px;
background-color: blanchedalmond;
}
`
```

Customizing the time picker popup in the DateTimePicker

Use the following CSS to customize the time picker popup in the DateTimePicker

```
`css
/ To specify height /
.e-datetimepicker.e-popup {
height: 100px;
}
`
```

Customizing the Calendar popup of the DateTimePicker

Please check the below section, to customize the style and appearance of the Calendar component in the DateTimePicker

[Customizing Calendar's style and appearance](#)

Full screen mode support in mobiles and tablets

The DateTimePicker component's full-screen mode feature enables users to view the component popup element in full-screen mode on mobile devices with improved visibility and a better user experience. It is important to mention that this feature is exclusively available for mobile and tablet devices in both landscape and portrait orientations. To activate the full screen mode within the DateTimePicker component, simply set the [fullScreenMode](#) API value to `true`. This action will extend the calendar and time popup element to occupy the entire screen on mobile devices.

`javascript

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

```
@Component({
```

```
  selector: 'app-root',
```

```
  template: `<!-- To Render datetimepicker -->
```

```
  <ejs-datetimepicker [fullScreenMode]="true"></ejs-datetimepicker>`
```

```
})
```

```
export class AppComponent { }
```

```
,
```

Default Sample

12/15/2017 2:00 PM		
--------------------	-----------------------------------------------------------------------------------	-----------------------------------------------------------------------------------

How To

Json data binding with datetimepicker in Angular Datetimepicker component

In most of the real cases, the model data will be available with JSON format only.

Here we have showcased DateTimePicker component by setting JSON string to value property.

In this JSON, we have used ISO formatted date string which is frequently used date format to get proper date and time value without any misreading.

Also our DateTimePicker component supports the ISO formatted date value, so parsed JSON value can be directly set to DateTimePicker model.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DateTimePickerModule } from '@syncfusion/ej2-angular-calendars'
import { Component } from '@angular/core';
export interface User {
    selectedDate: Date;
}
export interface JSONUser {
    selectedDate: string;
}
@Component({
    imports: [

        DateTimePickerModule //declaration of ej2-angular-calendars module
        into NgModule
    ],
    standalone: true,
    selector: 'app-root',
    template: `
        <!-- To Render DateTimePicker -->
        <ejs-datetimepicker id="datetimepicker" width='245px'
        [(value)]='user.selectedDate' (change)='onChange($event)'></ejs-
        datetimepicker>
        <div class="valuestring">
            <b>User model</b>: <br/>{{user | json }}
            <br/><br/>
            <b>JSON Data</b>: <br/>{{ model_result }}
            <br/><br/>
        </div>`
    })
export class AppComponent {
    public user?: User | any;
    public JSONData: JSONUser = JSON.parse('{ "selectedDate": "2018-12-18T08:56:00+00:00"}');
    public model_result: string = JSON.stringify(this.JSONData);
    public ngOnInit() {
        this.user = this.JSONData;
    }
    onChange(args: any) {
        this.JSONData.selectedDate = args.value;
        this.model_result = JSON.stringify(this.JSONData);
    }
}
```

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 Angular Datetimepicker component

The following example demonstrates how to achieve **two-way binding** by binding the **value** to the first DateTimePicker component by using property binding and binding the model data using **ngModel** by using model binding to the DateTimePicker component. The **value** of the DateTimePicker will get change, when there is any change in the property value or model value.

The two-way binding can also be achieved only by using **property binding** or **model binding** in the DateTimePicker component.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule } from '@angular/forms'
import { DateTimePickerModule } from '@syncfusion/ej2-angular-calendars'
import { Component } from '@angular/core';
@Component({
  imports: [
    DateTimePickerModule,
    FormsModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `
    <!-- two-way binding using the value binding and model binding in
the DateTimePicker --->
    <ejs-datetimepicker id="firstdatetime" #ejDateTimePicker
[ (value) ]='value' width="230px"></ejs-datetimepicker>
    <ejs-datetimepicker id="seconddatetime" #ejDateTimePickers
[ (ngModel) ]='value' width="230px"></ejs-datetimepicker>
  `
})
export class AppComponent {
  value: Date;
  constructor() {
    this.value = new Date("1/1/2019 1:30 PM");
  }
}
```

MAIN.TS

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

Disable placeholder readonly in Angular Datetimepicker component

Property | Purpose

[enabled](#) | The component can be restricted on a page, by setting `enabled` value as `false` which will disable the component completely from all user interactions including in form post action.

[placeholder](#) | Using `placeholder` you can display a short hint about the expected value in the input element.

[readonly](#) | Editing the value in the component can be prevented by setting `readonly` as `true`, but value can be included in the form post action.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule } from '@angular/forms'
import { DateTimePickerModule } from '@syncfusion/ej2-angular-calendars'
import { Component } from '@angular/core';
@Component({
  imports: [
    DateTimePickerModule,
    FormsModule
  ],
  standalone: true,
  selector: 'app-root',
  templateUrl: './template.html'
})
export class AppComponent {
}
```

MAIN.TS

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

Custom event emitter in Angular Datetimepicker component

The **two-way binding** in DateTimePicker can also be achieved using the custom event binding and property binding in the controls present in two different components. To create custom event, we need to create an instance of `event emitter`.

In the following example, **property binding** is used to share the data from the parent component to the child component using `@input` directive and **custom event binding** is used to share the data from the child component to the parent component by using `@output` directive.

APP.COMPONENT.TS

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

```

@Component({
  imports:      [ DateTimePickerModule ],
  standalone: true,
  selector: 'app-root',
  template: `
    <div class="parentelement">
      <div class="datevalue">
        <ejs-datetimepicker id="datetime" #datetime (change)="deposit()"
placeholder="Parent component" floatLabelType="Always" [value]="value"
width="200px"></ejs-datetimepicker>
      </div>
    </div>
    <child [xvalue]="value" (valueChange)="valuecheck($event)"> </child>
  `,
})
export class ParentComponent {
  value: Date;
  Date: any;
  constructor() {
    this.value = new Date("2/1/2020");
  }
  deposit() {
    this.value = this.Date.value as Date;
  }
  valuecheck(args: any) {
    this.value = args;
  }
}

```

MAIN.TS

```

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

```

Custom validation using form validator in Angular Datetimepicker component

The client side validation takes place in the browser to avoid the waiting time to receive the response from sever. It validates the form elements to provide the better feedback messages to correct the every fields before the form submission.

To achieve the client side validation in a DateTimePicker component by using **FormValidator** function. It provides an option to customize the feedback error messages to the corresponding fields to take action to resolve the issue.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { FormsModule } from '@angular/forms'
import { BrowserModule } from '@angular/platform-browser'
import { DateTimePickerModule } from '@syncfusion/ej2-angular-calendars'
import { Component, ViewChild, OnInit } from '@angular/core';
import { DateTimePickerComponent } from '@syncfusion/ej2-angular-calendars';
import { FormValidator, FormValidatorModel } from '@syncfusion/ej2-inputs';
@Component({

```

```

imports: [
    DateTimePickerModule,
    FormsModule
],
standalone: true,
selector: 'app-root',
template: `<form id="form-element" class="form-vertical">
    <ejs-datetimepicker #ejDateTime id='datetimepicker' placeholder='Enter
date and time' width="275px" (blur)="onFocusOut()" (change)="
onChange($event)"></ejs-datetimepicker>
</form>`
})
export class AppComponent implements OnInit {
    @ViewChild('formElement') element: any;
    @ViewChild('ejDateTime') ejDateTime?: DateTimePickerComponent;
    public formObject?: FormValidator;
    ngOnInit() {
        // custom validator function.
        let customFn: (args: {
            [key: string]: string
        }) => boolean = (args: {
            [key: string]: string
        }) => {
            return (((this.ejDateTime as DateTimePickerComponent)
).value).getFullYear() > 1990 && (((this.ejDateTime as
DateTimePickerComponent).value).getFullYear() < 2020);
        };
        let options: FormValidatorModel = {
            rules: {
                'datetimepicker': {
                    required: [true, "Value is required"]
                }
            },
            customPlacement: (inputElement: HTMLElement, errorElement:
HTMLElement) => {
                inputElement?.parentElement?.parentElement?.appendChild(errorElement);
            }
        };
        this.formObject = new FormValidator('#form-element', options);
        this.formObject.addRules('datetimepicker', {
            range: [customFn, "Please select a date between years from 1990
to 2020"]
        });
        this.formObject = new FormValidator('#form-element', options);
    }
    // Form validation takes place when focus() event of datetimepicker is
triggered.
    public onFocusOut(): void {
        this.formObject?.validate("datetimepicker");
    }
    // Custom validation takes place when value is changed.
    public onChange(args: any) {
        if (this.ejDateTime?.value != null)
            this.formObject?.validate("datetimepicker");
    }
}

```



```
}

```

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 in Angular Datetimepicker component

DateTimePicker is a form component and validation is playing vital role in forms to get the valid data.

Here to showcase the DateTimePicker with form validations we have used the reactive form.

- 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, 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. Declare the `formControlName` to DateTimePicker component as usual.

Then, 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 { DateTimePickerModule } from '@syncfusion/ej2-angular-calendars'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component, ViewChild, OnInit, ElementRef, Inject } from '@angular/core';
import { DateTimePickerComponent } from '@syncfusion/ej2-angular-calendars';
import { FormValidator, FormValidatorModel } from '@syncfusion/ej2-inputs';
import { FormBuilder, FormGroup, Validators } from '@angular/forms';
import { FormsModule } from '@angular/forms';
@Component({
  imports: [
    FormsModule, ReactiveFormsModule, DateTimePickerModule, ButtonModule
  ],
  standalone: true,
```

```

    selector: 'app-root',
    templateUrl: './template.html'
  })
  export class AppComponent implements OnInit {
    @ViewChild('ejDateTimePicker') ejDateTimePicker?:
    DateTimePickerComponent;
    public targetElement?: HTMLElement;
    public placeholder: String = 'Select date and time';
    skillForm?: FormGroup;
    build: FormBuilder;
    constructor(@Inject(FormBuilder) private builder: FormBuilder) {
      this.build = builder;
      this.createForm();
    }
    ngOnInit(): void {
      throw new Error('Method not implemented.');
```

```

    }
    createForm() {
      this.skillForm = this.build.group({
        datetimepicker: ['', Validators.required],
        username: ['', Validators.required],
        usermail: ['', Validators.email],
      });
    }
    get username() {
      return this.skillForm?.get('username');
    }
    get datetimepicker() {
      return this.skillForm?.get('datetimepicker');
    }
    get usermail() {
      return this.skillForm?.get('usermail');
    }
    onSubmit() {
      alert("Form Submitted!");
    }
  }
}
```

MAIN.TS

```

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

Template driven forms in Angular Datetimepicker component

The form can be build with Angular template syntax easily along with form specific directives.

This 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>.

- In angular forms mentioning the name is must to process as form elements.
- Mention the `name` attribute to DateTimePicker element which will be used to identify the form element. To register an DateTimePicker element to ngForm, give the ngModel to it so the FormsModule will automatically detect the DateTimePicker as a form element.

After that, the DateTimePicker value will be selected based on the ngModel value.

The following example demonstrates template driven forms with DateTimePicker component.

APP.COMPONENT.TS

```
import { BrowserModule } from '@angular/platform-browser'
import { NgModule } from '@angular/core'
import { FormsModule, ReactiveFormsModule } from '@angular/forms'
import { DateTimePickerModule } from '@syncfusion/ej2-angular-calendars'
import { Component, ViewChild, Inject } from '@angular/core';
import { FormGroup, FormBuilder, Validators } from '@angular/forms';
import { DateTimePickerComponent } from '@syncfusion/ej2-angular-calendars';
class User {
  public datetime?: Date;
  constructor() {
  }
}
@Component({
  imports: [ FormsModule, ReactiveFormsModule, DateTimePickerModule],
  standalone: true,
  selector: 'app-root',
  templateUrl: './template.html'
})
export class DefaultDateTimePickerComponent {
  constructor() {}
  user?: User | any;
  ngOnInit() {
    this.user = new User();
  }
  onSubmit(userForm: any) {
    (userForm.valid) ? alert("submitted") : alert("form is invalid");
  }
}
```

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>
  <form #userForm="ngForm" (ngSubmit)="onSubmit(userForm)">
    <div class="form-group">
      <ejs-datetimepicker id="datetimepicker" name="datetime"
        [(ngModel)]="user.datetime" #date="ngModel" width="300px" required></ejs-
datetimepicker>
```

```

    <div *ngIf="userForm.invalid && (userForm.dirty ||
userForm.touched)" class="alert alert-danger formerror">
      Valid date and time is required
    </div>
    <div *ngIf="userForm.valid && (userForm.dirty || userForm.touched)"
class="alert alert-success formerror">
      <table>
        <tr>
          <td style="width:50%">Selected value: </td>
          <td class="formtext ">{{ user.datetime | date:"dd/MM/yyyy
hh:mm a"}}</td>
        </tr>
      </table>
    </div>
  </div>
  <div class="buttons">
    <button type="submit" class="e-btn e-success">Submit</button>
    <button type="reset" class="e-btn e-warning">Reset</button>
  </div>
</form>
<div class="dataclass">userForm.value: {{userForm.value | json}}</div>
<div class="dataclass">userForm.valid: {{userForm.valid}}</div>
</div>
<style>
  form {
    margin-top: 50px;
  }
</style>

```

Customize the datetimepicker day header in Angular Datetimepicker component

You can change the format of the day that to be displayed in header using [dayHeaderFormat](#) property.

By default, the format is **Short**.

You can find the possible formats on below.

Name	Description

Short	Sets the short format of day name (like Su) in day header.
Narrow	Sets the single character of day name (like S) in day header.
Abbreviated	Sets the min format of day name (like Sun) in day header.
Wide	Sets the long format of day name (like Sunday) in day header.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DateTimePickerModule } from '@syncfusion/ej2-angular-calendars'
import { DropDownListModule } from '@syncfusion/ej2-angular-dropdowns'
import { Component, ViewChild } from '@angular/core';
import { DateTimePickerComponent, DayHeaderFormats } from '@syncfusion/ej2-
angular-calendars';

```

```

import { DropDownListComponent, ChangeEventArgs } from '@syncfusion/ej2-angular-dropdowns';
@Component({
  imports: [

    DropDownListModule,
    DateTimePickerModule
  ],
  standalone: true,
  selector: 'app-root',
  styleUrls: ['./styles.css'],
  template: `
    <div id="container">
      <div id="datetimepicker">
        <ejs-datetimepicker #default dayHeaderFormat='Short'></ejs-
datetimepicker>
      </div>
      <div id="format">
        <label class="custom-input-label">Header Format
Types</label>
        <div id="wrap">
          <ejs-dropdownlist id="dayformat" #select
[dataSource]='formatData' [(value)]='value' [fields]='fields'
[placeholder]='waterMark' (change)='formatHandler($event)'></ejs-
dropdownlist>
        </div>
      </div>
    </div>
  `
})
export class AppComponent {
  @ViewChild('default')
  public datetimepickerObj?: DateTimePickerComponent;
  @ViewChild('select')
  public dayHeaderFormat?: DropDownListComponent;
  // define the JSON of data
  public formatData: Object[] = [
    { Id: 'Short', Label: 'Short' },
    { Id: 'Narrow', Label: 'Narrow' },
    { Id: 'Abbreviated', Label: 'Abbreviated' },
    { Id: 'Wide', Label: 'Wide' },
  ];
  public fields: Object = { text: 'Label', value: 'Id' };
  public waterMark: string = 'Select format type';
  public value: string = 'Short';
  public formatHandler(args: ChangeEventArgs): void {
    (this.datetimepickerObj as DateTimePickerComponent).dayHeaderFormat
= args.value as DayHeaderFormats;
  }
}

```

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 Datetimepicker component

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

DateTime Selection

<!-- markdownlint-disable MD033 -->

Behavior	API in Essential JS 1	API in Essential JS 2
Setting the Value	Property: value	Property: value

DateTime Format

<!-- markdownlint-disable MD033 -->

Behavior	API in Essential JS 1	API in Essential JS 2
Display datetime format	Property: dateTimeFormat	Property: format
Day header format	Property: dayHeaderFormat	Not Applicable

Calendar Views

<!-- markdownlint-disable MD033 -->

Behavior	API in Essential JS 1	API in Essential JS 2
Start view	Property: startLevel	Property: start
Depth view	Property: depthLevel	Property: depth

Date Range

<!-- markdownlint-disable MD033 -->

Behavior	API in Essential JS 1	API in Essential JS 2
Minimum datetime	Property: minDateTime	Property: min
Maximum datetime	Property: maxDateTime	Property: max

Disabled Dates

<!-- markdownlint-disable MD033 -->

Behavior	API in Essential JS 1	API in Essential JS 2
Disabled dates	Not Applicable	Can be achieved by <code>public disabledDatetime(args: any): void { /Date need to be disabled/ if (args.date.getDay() === 0 args.date.getDay() === 6) { args.isDisabled = true; }}</code>

Customization

<!-- markdownlint-disable MD033 -->

Behavior	API in Essential JS 1	API in Essential JS 2
CSS Class	Property: cssClass	Property: cssClass
Show/Hide the today button	Can be achieved by <code>css.e-datetime-popup.e-popup.e-custom-class .e-button-container { display: none !important;}</code>	Property: showTodayButton
Show/Hide the other month dates	Property: showOtherMonths	Can be achieved by <code>css.e-DateTimePicker .e-calendar .e-content tr.e-month-hide, .e-DateTimePicker .e-calendar .e-content td.e-other-month > .e-day { visibility: none;}</code> <code>.e-DateTimePicker .e-calendar .e-content td.e-month-hide, .e-DateTimePicker .e-calendar .e-content td.e-other-month { pointer-events: none; touch-action: none;}</code>
Show/Hide the popup button	Property: showPopupButton	Event: <code>focus@ViewChild("datetimeObj") datetimeObj: DateTimePickerComponent; public onFocus(args:any): void { this.datetimeObj.show();}</code> <code>css.e-control-wrapper .e-input-group-icon.e-date-icon { display: none;}</code>
Enable/Disable the rounded corner	Property: showRoundedCorner	Can be achieved by <code>css.e-control-wrapper.e-custom-style.e-date-wrapper.e-input-group { border-radius: 4px;}</code>
Skip a month	Property: stepMonths	Can be achieved by <code>@ViewChild("datetimeObj") datetimeObj: DateTimePickerComponent; public onOpen(args) { this.datetimeObj.navigateTo('Year', new Date("03/8/2018"));}</code>
Show/Hide the tooltip	Property: showTooltip	Not Applicable
Interval	Property: interval	Property: step

Button text	Property: buttonText Object = { done: "Ok" }	Can be achieved by <code>L10n.load({ 'en': { 'datetimepicker': { today: 'Now' } } });</code>
Enable/Disable the animation	Property: enableAnimation	Not Applicable
FocusIn method	Not Applicable	Method: <code>focusIn()@ViewChild("datetimeObj")</code> datetimeObj: DateTimePickerComponent;public create(args:any): void{ this.datetimeObj.focusIn();}
FocusOut method	Not Applicable	Method: <code>focusOut()@ViewChild("datetimeObj")</code> datetimeObj: DateTimePickerComponent;public create(args:any): void{ this.datetimeObj.focusIn(); this.datetimeObj.focusOut();}
Prevent popup close	Not Applicable	Event: Closepublic onClose(args:any): void{ args.cancel = true;}
Prevent popup open	Not Applicable	Event: Openpublic onOpen(args:any): void{ args.cancel = true;}

Accessibility

<!-- markdownlint-disable MD033 -->

Behavior	API in Essential JS 1	API in Essential JS 2
Enable/Disable the RTL	Property: enableRTL	Property: enableRtl

Persistence

<!-- markdownlint-disable MD033 -->

Behavior	API in Essential JS 1	API in Essential JS 2
Enable/Disable the persistence	Property: enablePersistence	Property: enablePersistence

Validation

<!-- markdownlint-disable MD033 -->

Behavior	API in Essential JS 1	API in Essential JS 2
Validation rules	Property: validationRules:validationRules:Object;constructor(){ this.validationRules = {required:true};}	Can be achieved bylet options: FormValidatorModel = { rules: { 'datetimepicker': { required: [true, "Value is required"] } } }; this.formObject = new FormValidator('#form-element', options);
Validation message	Property: validationMessage:validationRules:Object;validationMessage:Object;constructor(){ this.validationMessage = {required: "Required DateTime value"}; this.validationRules = {required:true};}	Can be achieved bylet options: FormValidatorModel = { rules: { 'datetimepicker': { required: [true, "Value is required"] } }, customPlacement: (inputElement: HTMLElement, errorElement: HTMLElement) => { inputElement.parentElement.parentElement.appendChild(errorElement); } };this.formObject = new FormValidator('#form-element', options);

Common

<!-- markdownlint-disable MD033 -->

Behavior	API in Essential JS 1	API in Essential JS 2
Width	Property: Width	Property: Width
Readonly	Property: ReadOnly	Property: Readonly
Show/Hide the clear button	Not Applicable	Property: showClearButton
Height	Property: Height	Can be achieved by <code>css.e-control-wrapper.e-custom-style.e-date-wrapper.e-input-group { height: 35px;}</code>
Html Attributes	Property: <code>HtmlAttributes</code> Object = <code>{required:"required"}</code>	Not Applicable
Show/Hide the week number	Property: weekNumber	Property: weekNumber
Watermark text	Property: watermarkText	Property: Placeholder
Disable/Enable	Property: enabled	Property: enabled
Enable/Disable the textbox editing	Property: AllowEdit	Property: AllowEdit
zIndex	Can be achieved by <code>css.e-datetime-popup.e-popup.e-custom-class { z-index: 100 !important;}</code>	Property: zIndex
Specify the placeholder text behavior	Not Applicable	Property: floatLabelType
Event callback for each cell creation	Not Applicable	Event: <code>renderDayCell</code> <code>public onRenderCell(args:any):void{ / code block */}</code>
FocusIn event	Event: <code>FocusIn</code> <code>public onFocus(e:any) { /Triggers when the popup gets focus/ } */}</code>	Event: <code>focus</code> <code>public onFocus(args:any):void{ / code block */}</code>
FocusOut event	Event: <code>focusOut</code> <code>public onFocusout(e:any) { /Triggers when the popup gets focusout/ }</code>	Event: <code>blur</code> <code>public onBlur(args:any):void{ / code block */}</code>
Change event	Event: <code>change</code> <code>public onChange(e:any) { /Triggers when the value is changed/ }</code>	Event: <code>change</code> <code>public onChange(args:any):void{ / code block */}</code>

Behavior	API in Essential JS 1	API in Essential JS 2
Width	Property: Width	Property: Width
Readonly	Property: ReadOnly	Property: Readonly
Show/Hide the clear button	Not Applicable	Property: showClearButton
Height	Property: Height	Can be achieved by <code>css.e-control-wrapper.e-custom-style.e-date-wrapper.e-input-group { height: 35px;}</code>
Html Attributes	Property: <code>HtmlAttributes</code> <code>Object = {required:"required"}</code>	Not Applicable
Show/Hide the week number	Property: weekNumber	Property: weekNumber
Watermark text	Property: watermarkText	Property: Placeholder
Disable/Enable	Property: enabled	Property: enabled
Enable/Disable the textbox editing	Property: AllowEdit	Property: AllowEdit
zIndex	Can be achieved by <code>css.e-datetime-popup.e-popup.e-custom-class { z-index: 100 !important;}</code>	Property: zIndex
Specify the placeholder text behavior	Not Applicable	Property: floatLabelType
Event callback for each cell creation	Not Applicable	Event: <code>renderDayCell</code> <code>public onRenderCell(args:any):void{ / code block */}</code>
FocusIn event	Event: <code>FocusIn</code> <code>public onFocus(e:any) { /Triggers when the popup gets focus/ } */}</code>	Event: <code>focus</code> <code>public onFocus(args:any):void{ / code block */}</code>
Created event	Event: <code>create</code> <code>public onCreate(e:any) { /Triggers when the control is created/ }</code>	Event: <code>created</code> <code>public onCreate(args:any):void{ / code block */}</code>
Destroy event	Event: <code>Destroy</code> <code>public onDestroy(e:any) { /Triggers</code>	Event: <code>destroyed</code> <code>@ViewChild("datetimeObj") datetimeObj:</code>

Behavior	API in Essential JS 1	API in Essential JS 2
Width	Property: Width	Property: Width
Readonly	Property: ReadOnly	Property: Readonly
Show/Hide the clear button	Not Applicable	Property: showClearButton
Height	Property: Height	Can be achieved by <code>css.e-control-wrapper.e-custom-style.e-date-wrapper.e-input-group { height: 35px;}</code>
Html Attributes	Property: <code>HtmlAttributes</code> Object = <code>{required:"required"}</code>	Not Applicable
Show/Hide the week number	Property: weekNumber	Property: weekNumber
Watermark text	Property: watermarkText	Property: Placeholder
Disable/Enable	Property: enabled	Property: enabled
Enable/Disable the textbox editing	Property: AllowEdit	Property: AllowEdit
zIndex	Can be achieved by <code>css.e-datetime-popup.e-popup.e-custom-class { z-index: 100 !important;}</code>	Property: zIndex
Specify the placeholder text behavior	Not Applicable	Property: floatLabelType
Event callback for each cell creation	Not Applicable	Event: <code>renderDayCell</code> <code>public onRenderCell(args:any):void{/ code block */}</code>
FocusIn event	Event: <code>FocusIn</code> <code>public onFocus(e:any) { /Triggers when the popup gets focus/ } */}</code>	Event: <code>focus</code> <code>public onFocus(args:any):void{/ code block */}</code>
	<code>when the control is destroyed/ }</code>	<code>DateTimePickerComponent;public onDestroyed(args:any):void{ console.log("destroyed");}public onChange(args:any):void{ this.datetimeObj.destroy();}</code>

Globalization

<!-- markdownlint-disable MD033 -->

Behavior	API in Essential JS 1	API in Essential JS 2
Locale	Property: locale	Property: locale
Specify the start day of week	Property: startDay	Property: firstDayOfWeek

Strict Mode

<!-- markdownlint-disable MD033 -->

Behavior	API in Essential JS 1	API in Essential JS 2
Strict mode	Property: enableStrictMode	Property: strictMode

Open and Close

<!-- markdownlint-disable MD033 -->

Behavior	API in Essential JS 1	API in Essential JS 2
Close	Event: Closepublic onClose(e:any) { /Triggers when the poupup gets closed/ }	Event: closepublic onClose(args:any): void { /Triggers when the poupup gets closed/ }
Hide	Method: hide()public onCreate(e:any) { var datetimeObject = \$("#datetimepicker").data("ejDateTimePicker"); datetimeObject.show(); datetimeObject.hide(); }	Method: hide()@ViewChild("datetimeObj") datetimeObj: DateTimePickerComponent;public onCreate(args:any): void { this.datetimeObj.show(); this.datetimeObj.hide(); }
Open	Event: openpublic onOpen(e:any) { /Triggers when the poupup gets closed/ }	Event: openpublic onOpen(args:any): void { /Triggers when the poupup gets closed/ }
Show	Method: show()public onCreate(e:any) { var datetimeObject = \$("#datetimepicker").data("ejDateTimePicker"); datetimeObject.show(); }	Method: show()@ViewChild("datetimeObj") datetimeObj: DateTimePickerComponent;public onCreate(args:any): void { this.datetimeObj.show(); this.datetimeObj.hide(); }

View Navigation

<!-- markdownlint-disable MD033 -->

Behavior	API in Essential JS 1	API in Essential JS 2
Navigate to specific month	Not Applicable	Method: navigateTo()@ViewChild("datetimeObj") datetimeObj: DateTimePickerComponent;public onOpen(args) { this.datetimeObj.navigateTo('Year', new Date("03/8/2018"));} }
Navigation callback	Not Applicable	Event: navigated@ViewChild("datetimeObj") datetimeObj: DateTimePickerComponent;public onNavigated(args) { / code block */ }
Enable/Disable the drill down	Property: timeDrillDown timeDrillDown: Object = { showMeridian: true , interval: 10 , enabled: true }	Not Applicable

Diagram

Getting started with Angular Diagram component

This section explains you the steps required to create a simple diagram and demonstrate the basic usage of the diagram control.

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 Diagram 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-diagrams](#) package to the application.

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

```
`bash
npm install @syncfusion/ej2-angular-diagrams@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-diagrams:"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 Diagram Module

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

```
`typescript
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
// import the DiagramModule for the Diagram component
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams';
```



```
import { AppComponent } from './app.component';
@NgModule({
  //declaration of ej2-angular-diagrams module into NgModule
  imports: [ BrowserModule, DiagramModule ],
  declarations: [ AppComponent ],
  bootstrap: [ AppComponent ]
})
export class AppModule { }
```

Adding CSS reference

Combined CSS files are available in the Essential JS 2 package root folder. This can be referenced in [src/styles.css] using following code.

```
`css
@import '../node_modules/@syncfusion/ej2-angular-diagrams/styles/material.css';
@import "../node_modules/@syncfusion/ej2-angular-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-angular-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-angular-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-angular-navigations/styles/material.css";
```

Add Diagram component

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

```
`typescript
import { Component, OnInit, ViewEncapsulation } from '@angular/core';
@Component({
  selector: 'app-container',
  // specifies the template string for the Diagram component
  template: <ejs-diagram id='diagram-container'></ejs-diagram>,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent implements OnInit {
  ngOnInit(): void {
  }
}
```

Defining Basic Diagram

The below example shows a basic Diagrams

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component } from '@angular/core';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  // specifies the template string for the diagram component
  template: `<ejs-diagram id="diagram" width="100%" height="580px"></ejs-diagram>`
})
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));
```

Now, run the application by using npm start command. Open the browser with the generated link and you can see an empty diagram.

```
`bash
```

```
npm start
```

Module Injection

The diagram component is divided into individual feature-wise modules. In order to use a particular feature, inject the required module. The following list describes the module names and their description.

- **BpmnDiagramsService** - Inject this provider to add built-in BPMN Shapes to diagrams.
- **ConnectorBridgingService** - Inject this provider to add bridges to connectors.
- **ConnectorEditingService** - Inject this provider to edit the segments for connector.
- **ComplexHierarchicalTreeService** - Inject this provider to complex hierarchical tree like structure.
- **DataBindingService** - Inject this provider to populate nodes from given data source.
- **DiagramContextMenuService** - Inject this provider to manipulate context menu.
- **HierarchicalTreeService** - Inject this provider to use hierarchical tree like structure.

- `LayoutAnimationService` - Inject this provider animation to layouts.
- `MindMapService` - Inject this provider to use mind map.
- `PrintAndExportService` - Inject this provider to print or export the objects.
- `RadialTreeService` - Inject this provider to use Radial tree like structure.
- `SnappingService` - Inject this provider to Snap the objects.
- `SymmetricLayoutService` - Inject this provider to render layout in symmetrical method.
- `UndoRedoService` - Inject this provider to revert and restore the changes.

These modules should be imported and injected into the Diagram component using `Diagram.Inject` method as follows.

```
`javascript
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams';
import { HierarchicalTreeService, MindMapService, RadialTreeService, ComplexHierarchicalTreeService }
from '@syncfusion/ej2-angular-diagrams';
import { DataBindingService, SnappingService, PrintAndExportService, BpmnDiagramsService } from
'@syncfusion/ej2-angular-diagrams';
import { SymmetricLayoutService, ConnectorBridgingService, UndoRedoService,
LayoutAnimationService } from '@syncfusion/ej2-angular-diagrams';
import { DiagramContextMenuService, ConnectorEditingService } from '@syncfusion/ej2-angular-
diagrams';
@NgModule({
imports: [
BrowserModule, DiagramModule
],
declarations: [AppComponent],
bootstrap: [AppComponent],
providers: [ HierarchicalTreeService, MindMapService, RadialTreeService,
ComplexHierarchicalTreeService, DataBindingService, SnappingService, PrintAndExportService,
BpmnDiagramsService, SymmetricLayoutService, ConnectorBridgingService, UndoRedoService,
LayoutAnimationService, DiagramContextMenuService, ConnectorEditingService ]
})
`
```

Flow Diagram

Create and Add Node

Create and add a `node` (JSON data) with specific position, size, label, and shape.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, ViewChild } from "@angular/core";
import { BasicShapeModel } from "@syncfusion/ej2-angular-diagrams";
@Component({
  imports: [
    DiagramModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram id="diagram" width="100%" height="580px"
mode="SVG">
  <e-nodes>
    <e-node id='node1' [height]=60 [width]=100 [offsetX]=300
[offsetY]=80 [shape]='shape'>
      <e-node-annotations>
        <e-node-annotation content='Start'></e-node-annotation>
      </e-node-annotations>
    </e-node>
    <e-node id='node2' [height]=60 [width]=100 [offsetX]=300
[offsetY]=160 [shape]='shape'>
      <e-node-annotations>
        <e-node-annotation content='var i = 0;'></e-node-annotation>
      </e-node-annotations>
    </e-node>
  </e-nodes>
  <e-connectors>
    <e-connector id='connector1' sourceID='node1' targetID='node2'></e-
connector>
  </e-connectors>
</ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild("diagram")
  public shape?: BasicShapeModel;
  ngOnInit(): void {
    this.shape = { type: "Basic", shape: "Rectangle" };
  }
}
```

MAIN.TS

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

Connect two Nodes with a Connector

Add two node to the diagram as shown in the previous example. Connect these nodes by adding a connector using the `connector` property and refer the source and target end by using the `sourceNode` and `targetNode` properties.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, ViewChild } from "@angular/core";
import { BasicShapeModel } from "@syncfusion/ej2-angular-diagrams";
@Component({
  imports: [
    DiagramModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram id="diagram" width="100%" height="580px"
mode="SVG">
  <e-nodes>
    <e-node id='node1' [height]=60 [width]=100 [offsetX]=300
[offsetY]=80 [shape]='shape'>
      <e-node-annotations>
        <e-node-annotation content='Start'></e-node-annotation>
      </e-node-annotations>
    </e-node>
    <e-node id='node2' [height]=60 [width]=100 [offsetX]=300
[offsetY]=160 [shape]='shape'>
      <e-node-annotations>
        <e-node-annotation content='var i = 0;'></e-node-annotation>
      </e-node-annotations>
    </e-node>
  </e-nodes>
  <e-connectors>
    <e-connector id='connector1' sourceID='node1' targetID='node2'></e-
connector>
  </e-connectors>
</ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild("diagram")
  public shape?: BasicShapeModel;
  ngOnInit(): void {
    this.shape = { type: "Basic", shape: "Rectangle" };
  }
}
```

MAIN.TS

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

Default values for all **nodes** and **connectors** can be set using the **getNodeDefaults** and **getConnectorDefaults** properties, respectively. For example, if all nodes have the same width and height, such properties can be moved into **getNodeDefaults**.

[Complete Flow Diagram](#)

Similarly, the required nodes and connectors can be added to form a complete flow diagram.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, ViewChild } from "@angular/core";
import {
  FlowShapeModel,
  NodeModel,
  ConnectorModel,
  OrthogonalSegmentModel
} from "@syncfusion/ej2-angular-diagrams";
@Component({
  imports: [
    DiagramModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram id="diagram" width="100%" height="580px"
[getNodeDefaults]='nodeDefaults' [getConnectorDefaults]='connectorDefaults'>
  <e-nodes>
    <e-node id='node1' [offsetY]=50 [shape]='terminator'>
      <e-node-annotations>
        <e-node-annotation content='Start'></e-node-annotation>
      </e-node-annotations>
    </e-node>
    <e-node id='node2' [offsetY]=140 [shape]='process'>
      <e-node-annotations>
        <e-node-annotation content='var i = 0;'></e-node-annotation>
      </e-node-annotations>
    </e-node>
    <e-node id='node3' [offsetY]=230 [shape]='decision'>
      <e-node-annotations>
        <e-node-annotation content='i < 10?'></e-node-annotation>
      </e-node-annotations>
    </e-node>
    <e-node id='node4' [offsetY]=320 [shape]='preDefinedProcess'>
      <e-node-annotations>
        <e-node-annotation content='print(\"Hello!!\");'></e-node-
annotation>
      </e-node-annotations>
    </e-node>
    <e-node id='node5' [offsetY]=410 [shape]='process'>
      <e-node-annotations>
        <e-node-annotation content='i++;'></e-node-annotation>
      </e-node-annotations>
    </e-node>
  </e-nodes>
```

```

        <e-node id='node6' [offsetY]=500 [shape]='terminator'>
            <e-node-annotations>
                <e-node-annotation content='End'></e-node-annotation>
            </e-node-annotations>
        </e-node>
    </e-nodes>
    <e-connectors>
        <e-connector id='connector1' sourceID='node1' targetID='node2'></e-connector>
        <e-connector id='connector2' sourceID='node2' targetID='node3'></e-connector>
        <e-connector id='connector3' sourceID='node3' targetID='node4'>
            <e-connector-annotations>
                <<e-connector-annotation content='Yes'></e-connector-annotation>
            </e-connector-annotations>
        </e-connector>
        <e-connector id='connector4' sourceID='node3' targetID='node6' [segments]='segment1'>
            <e-connector-annotations>
                <e-connector-annotation content='No'></e-connector-annotation>
            </e-connector-annotations>
        </e-connector>
        <e-connector id='connector5' sourceID='node4' targetID='node5'></e-connector>
        <e-connector id='connector6' sourceID='node5' targetID='node3' [segments]='segment2'></e-connector>
    </e-connectors>
</ejs-diagram>`,
    encapsulation: ViewEncapsulation.None
})
export class AppComponent {
    @ViewChild("diagram")
    public terminator?: FlowShapeModel;
    public process?: FlowShapeModel;
    public decision?: FlowShapeModel;
    public preDefinedProcess?: FlowShapeModel;
    public segment1?: OrthogonalSegmentModel;
    public segment2?: OrthogonalSegmentModel;
    public nodeDefaults(node: NodeModel): NodeModel {
        node.height = 50;
        node.width = 140;
        node.offsetX = 300;
        return node;
    }
    public connectorDefaults(obj: ConnectorModel): ConnectorModel {
        obj.type = "Orthogonal";
        obj.targetDecorator = { shape: "Arrow", width: 10, height: 10 };
        return obj;
    }
    ngOnInit(): void {
        this.terminator = { type: 'Flow', shape: 'Terminator' };
        this.process = { type: 'Flow', shape: 'Process' };
        this.decision = { type: 'Flow', shape: 'Decision' };
        this.preDefinedProcess = { type: 'Flow', shape: 'PreDefinedProcess' };
    }
}

```

```
    this.segment1 = [{ length: 30, direction: "Right" }, { length: 300,
direction: "Bottom" }];
    this.segment2 = [{ length: 30, direction: "Left" }, { length: 200,
direction: "Top" }];
  }
}
```

MAIN.TS

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

Automatic Organization Chart

In the 'Flow Diagram' section, how to create a diagram manually was discussed. This section explains how to create and position the diagram automatically.

Business object (Employee information)

Define Employee Information as JSON data. The following code example shows an employee array whose, **Name** is used as a unique identifier and **ReportingPerson** is used to identify the person to whom an employee report to, in the organization.

`typescript

```
public data: Object[] = [
{
  Name: "Elizabeth",
  Role: "Director"
},
{
  Name: "Christina",
  ReportingPerson: "Elizabeth",
  Role: "Manager"
},
{
  Name: "Yoshi",
  ReportingPerson: "Christina",
  Role: "Lead"
},
{
  Name: "Philip",
  ReportingPerson: "Christina",
```



```

Role: "Lead"
},
{
Name: "Yang",
ReportingPerson: "Elizabeth",
Role: "Manager"
},
{
Name: "Roland",
ReportingPerson: "Yang",
Role: "Lead"
},
{
Name: "Yvonne",
ReportingPerson: "Yang",
Role: "Lead"
}
];
`

```

Map data source

You can configure the above "Employee Information" with diagram, so that the nodes and connectors are automatically generated using the mapping properties. The following code example show how `dataSourceSettings` is used to map ID and parent with property name identifiers for employee information.

```

`typescript
@Component({
selector: "app-container",
template: <ejs-diagram id="diagram" width="100%" height="580px"
[dataSourceSettings]='dataSourceSettings'></ejs-diagram>
})
export class AppComponent {
@ViewChild("diagram")
public dataSourceSettings: DataSourceModel;
public data: Object[] = [
{

```

```
Name: "Elizabeth",  
Role: "Director"  
},  
{  
Name: "Christina",  
ReportingPerson: "Elizabeth",  
Role: "Manager"  
},  
{  
Name: "Yoshi",  
ReportingPerson: "Christina",  
Role: "Lead"  
},  
{  
Name: "Philip",  
ReportingPerson: "Christina",  
Role: "Lead"  
},  
{  
Name: "Yang",  
ReportingPerson: "Elizabeth",  
Role: "Manager"  
},  
{  
Name: "Roland",  
ReportingPerson: "Yang",  
Role: "Lead"  
},  
{  
Name: "Yvonne",  
ReportingPerson: "Yang",  
Role: "Lead"  
}
```

```

];
ngOnInit(): void {
  this.dataSourceSettings = {
    id: "Name",
    parentId: "ReportingPerson",
    dataManager: new DataManager(this.data as JSON[])
  };
}
}
`

```

Visualize employee

The following code examples indicate how to define the default appearance of nodes and connectors. The `setNodeTemplate` is used to update each node based on employee data.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule, HierarchicalTreeService, DataBindingService } from
 '@syncfusion/ej2-angular-diagrams'
import { Component, OnInit, ViewChild } from "@angular/core";
import {
  NodeModel,
  ConnectorModel,
  LayoutModel,
  Diagram,
  DataSourceModel
} from "@syncfusion/ej2-diagrams";
import { DataManager } from "@syncfusion/ej2-data";
import { ShapeStyleModel } from "@syncfusion/ej2-angular-diagrams";
export interface EmployeeInfo {
  Name: string;
  Role: string;
  color: string;
}
@Component({
  imports: [
    DiagramModule
  ],
  providers: [ HierarchicalTreeService, DataBindingService ],
  standalone: true,
  selector: "app-container",
  template: `

```

```

public data: Object[] = [
  {
    Name: "Elizabeth",
    Role: "Director"
  },
  {
    Name: "Christina",
    ReportingPerson: "Elizabeth",
    Role: "Manager"
  },
  {
    Name: "Yoshi",
    ReportingPerson: "Christina",
    Role: "Lead"
  },
  {
    Name: "Philip",
    ReportingPerson: "Christina",
    Role: "Lead"
  },
  {
    Name: "Yang",
    ReportingPerson: "Elizabeth",
    Role: "Manager"
  },
  {
    Name: "Roland",
    ReportingPerson: "Yang",
    Role: "Lead"
  },
  {
    Name: "Yvonne",
    ReportingPerson: "Yang",
    Role: "Lead"
  }
];

public nodeDefaults(node: NodeModel): NodeModel {
  let codes: Object = {
    Director: "rgb(0, 139,139)",
    Manager: "rgb(30, 30,113)",
    Lead: "rgb(0, 100,0)"
  };
  node.width = 70;
  node.height = 30;
  node.annotations = [
    { content: (node.data as EmployeeInfo).Name, style: { color: "white" } }
  ];
  ((node as NodeModel).style as ShapeStyleModel).fill = (codes as
any)[(node.data as EmployeeInfo).Role] as string;
  return node;
}

public connectorDefaults(connector: ConnectorModel): ConnectorModel {
  connector.type = "Orthogonal";
  connector.cornerRadius = 7;
  return connector;
}

```

```
ngOnInit(): void {
  this.layout = {
    type: "OrganizationalChart"
  };
  this.dataSourceSettings = {
    id: "Name",
    parentId: "ReportingPerson",
    dataManager: new DataManager(this.data as JSON[])
  };
}
```

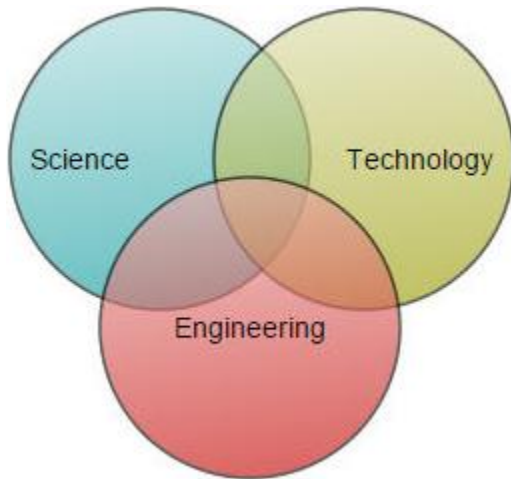
MAIN.TS

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

Note: Please note that project generated through angular CLI project will always the changes made into application and compiled it automatically. We don't need to run "npm start" command for each changes made into the application.

Nodes in Angular Diagram component

Nodes are graphical objects used to visually represent the geometrical information, process flow, internal business procedure, entity, or any other kind of data.



[Link to the Video](#)

<!-- markdownlint-disable MD033 -->

Create node

A node can be created and added to the diagram, either programmatically or interactively. Nodes are stacked on the diagram area from bottom to top in the order they are added.

To create a node easily and to know about different types of node shapes in a Angular Diagram, refer to the below video link.

Add node through nodes collection

To create a node, define the [node](#) object and add that to nodes collection of the diagram model. The following code example illustrates how to add a node to the diagram.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import { DiagramComponent, Diagram, NodeModel, ShapeStyleModel } from
 '@syncfusion/ej2-angular-diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%"
height="580px" [getNodeDefaults] ='getNodeDefaults'>
  <e-nodes>
    <e-node id='node1' [offsetX]=150 [offsetY]=150></e-node>
  </e-nodes>
</ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram?: DiagramComponent;
  public getNodeDefaults(node: NodeModel): NodeModel {
    node.height = 100;
    node.width = 100;
    ((node as NodeModel).style as ShapeStyleModel).fill = "#6BA5D7";
    ((node as NodeModel).style as ShapeStyleModel).strokeColor =
    "White";
    return node;
  }
}
```

MAIN.TS

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

Note: Node id should not begin with numbers(should begin with a letter).Node Id should be unique for all the shapes and connectors.

Add/Remove node at runtime

- Nodes can be added at runtime by using public method, add and can be removed at runtime by using public method, remove. On adding node at runtime, the nodes collection is changed and the [collectionChange](#) event will trigger.
- The node's ID property is used to define the name of the node and its further used to find the node at runtime and do any customization.

The following code illustrates how to add a node.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import { DiagramComponent, Diagram, NodeModel } from '@syncfusion/ej2-
angular-diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%"
height="580px" (created)='created($event)'>
</ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram?: DiagramComponent;
  public node: NodeModel = {
    // Position of the node
    offsetX: 250,
    offsetY: 250,
    // Size of the node
    width: 100,
    height: 100,
    style: {
      fill: '#6BA5D7',
      strokeColor: 'white'
    },
  };
  public created(args: Object): void {
    //Add Node
    (this.diagram as Diagram).add(this.node);
  }
}
```

MAIN.TS

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

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

Add node from palette

Nodes can be predefined and added to the palette, and can be dropped into the diagram when needed. For more information about adding nodes from symbol palette, refer to [Symbol Palette](#).

- Once you drag a node/connector from the palette to the diagram, the following events can be used to do your customization.
- When a symbol is dragged into diagram from symbol palette, the [dragEnter](#) event gets triggered.
- When a symbol is dragged over diagram, the [dragOver](#) event gets triggered.
- When a symbol is dragged and dropped from symbol palette to diagram area, the [drop](#) event gets triggered.
- When a symbol is dragged outside of the diagram, the [dragLeave](#) event gets triggered.

Create node through data source

Nodes can be generated automatically with the information provided through data source. The default properties for these nodes are fetched from default settings. For more information about data source, refer to Data Binding.

Draw nodes

Nodes can be interactively drawn by clicking and dragging the diagram surface by using [NodeDrawingTool](#). For more information about drawing nodes, refer to Draw Nodes.

Position

- Position of a node is controlled by using its [offsetX](#) and [offsetY](#) properties. By default, these offset properties represent the distance between the origin of the diagram's page and node's center point.
- You may expect this offset values to represent the distance between page origin and node's top-left corner instead of center. The Pivot property helps to solve this problem. Default value of node's [pivot](#) point is (0.5, 0.5), that means center of the node.
- The size of the node can be controlled by using its [width](#) and

[height](#) properties.

- Rotation of a node is controlled by using its [rotateAngle](#) property.

The following table illustrates how pivot relates offset values with node boundaries.

Pivot	Offset
-----	-----
(0.5,0.5)	offsetX and offsetY values are considered as the node's center point.
(0,0)	offsetX and offsetY values are considered as the top-left corner of the node.
(1,1)	offsetX and offsetY values are considered as the bottom-right corner of the node.

The following code illustrates how to change the **pivot** value.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, OnInit, ViewChild } from
 '@angular/core';
import { DiagramComponent, Diagram, NodeModel, PointModel, ShapeStyleModel }
from '@syncfusion/ej2-angular-diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%"
height="580px" [getNodeDefaults] ='getNodeDefaults'
(created)='created($event)'>
    <e-nodes>
      <e-node id='node1' [offsetX]=150 [offsetY]=150
[pivot]='pivot'></e-node>
    </e-nodes>
  </ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram?: DiagramComponent;
  public pivot?: PointModel;
  public getNodeDefaults(node: NodeModel): NodeModel {
    node.height = 100;
    node.width = 100;
    ((node as NodeModel).style as ShapeStyleModel).fill = "#6BA5D7";
    ((node as NodeModel).style as ShapeStyleModel).strokeColor =
    "White";
    return node;
  }
  ngOnInit(): void {
    this.pivot = { x: 0, y: 0 };
  }
  public created(args: Object): void {
    //Add Node
    (this.diagram as Diagram).select([(this.diagram as
Diagram).nodes[0]]);
  }
}
```

MAIN.TS

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

Flip

The diagram Provides support to flip the node. [flip](#) is performed to give the mirrored image of the original element.

The flip types are as follows:

- HorizontalFlip - [Horizontal](#) is used to change the element in horizontal direction.
- VerticalFlip - [Vertical](#) is used to change the element in vertical direction
- Both - [Both](#) which involves both vertical and horizontal changes of the element.

The following code illustrates how to provide the mirror image of the original element.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, OnInit, ViewChild } from '@angular/core';
import { DiagramComponent, Diagram, NodeModel, PointModel, FlipDirection, ShapeStyleModel, BasicShapeModel } from '@syncfusion/ej2-angular-diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%" height="580px" [getNodeDefaults] = 'getNodeDefaults' (created)='created($event)'>
    <e-nodes>
      <e-node id='node1' [offsetX]=150 [offsetY]=150 [shape]='shape'></e-node>
    </e-nodes>
  </ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram?: DiagramComponent;
  public getNodeDefaults(node: NodeModel): NodeModel {
    node.height = 100;
    node.width = 100;
    ((node as NodeModel).style as ShapeStyleModel).fill = "#6BA5D7";
    ((node as NodeModel).style as ShapeStyleModel).strokeColor = "White";
    // Flip the node in Horizontal Direction
    node.flip = 'Horizontal';
    return node;
  }
  public shape: BasicShapeModel = {
    type: 'Basic',
    shape: 'RightTriangle',
```

```

    } as BasicShapeModel;
    public created(args: Object): void {
        //Add Node
        (this.diagram as Diagram).select([(this.diagram as
Diagram).nodes[0]]);
    }
}

```

MAIN.TS

```

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

```

Note: The flip is also applicable for group and BPMN shapes.

Appearance

- The appearance of a node can be customized by changing its [fill](#) color, [borderColor](#), [borderWidth](#), [strokeDashArray](#), [opacity](#), and [shadow](#).
- The [visible](#) property of the node enables or disables the visibility of the node.

The following code illustrates how to customize the appearance of the shape.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, OnInit, ViewChild } from
 '@angular/core';
import { DiagramComponent, Diagram, NodeModel, ShapeStyleModel } from
 '@syncfusion/ej2-angular-diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%"
height="580px" [getNodeDefaults] ='getNodeDefaults'>
    <e-nodes>
      <e-node id='node1' [offsetX]=150 [offsetY]=150
[borderColor]='borderColor' [borderWidth]='borderWidth' [style]='style'></e-
node>
    </e-nodes>
  </ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram?: DiagramComponent;
  public style?: ShapeStyleModel;

```

```
public borderColor?: string;
public borderWidth?: number;
public getNodeDefaults(node: NodeModel): NodeModel {
    node.height = 100;
    node.width = 100;
    return node;
}
ngOnInit(): void {
    this.style = { fill: '#6BA5D7', strokeDashArray: '5,5' };
    this.borderColor = 'red';
    this.borderWidth = 2;
}
}
```

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 style of main node on multi-selection.

The style of the main node can be customized by using the className `e-diagram-first-selection-indicator`.

Use the following CSS to customize the style of main node on multiple selection.

```
`css
.e-diagram-first-selection-indicator{
stroke-width: 5px;
stroke: red;
stroke-dasharray: 1,1;
}
`
```

Gradient

The [gradient](#) property of the node allows you to define and apply the gradient effect to that node.

The gradient stop property defines the color and a position, where the previous color transition ends and a new color transition starts.

The gradient stop's opacity property defines the transparency level of the region.

There are two types of gradients as follows:

- Linear gradient
- Radial gradient

Linear gradient

- [LinearGradient](#) defines a smooth transition between a set of colors (so-called stops) on a line.
- A linear gradient's x1, y1, x2, y2 properties are used to define the position (relative to the node) of the rectangular region that needs to be painted.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, OnInit, ViewChild } from '@angular/core';
import { DiagramComponent, Diagram, NodeModel, ShapeStyleModel,
LinearGradientModel } from '@syncfusion/ej2-angular-diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%"
height="580px" [getNodeDefaults] ='getNodeDefaults'>
    <e-nodes>
      <e-node id='node1' [offsetX]=150 [offsetY]=150
[style]='style'></e-node>
    </e-nodes>
  </ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram?: DiagramComponent;
  public style?: ShapeStyleModel;
  public linearGradient: LinearGradientModel = {
    //Start point of linear gradient
    x1: 0,
    y1: 0,
    //End point of linear gradient
    x2: 50,
    y2: 50,
    //Sets an array of stop objects
    stops: [
      {
        color: 'white',
        offset: 0
      },
      {
        color: '#6BA5D7',
        offset: 100
      }
    ],
    type: 'Linear'
  };
  public getNodeDefaults(node: NodeModel): NodeModel {
```

```

        node.height = 100;
        node.width = 100;
        return node;
    }
    ngOnInit(): void {
        this.style = { gradient: this.linearGradient };
    }
}

```

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

- [RadialGradient](#) defines a smooth transition between stops on a circle.
- A radial gradient's cx, cy, fx, fy properties are used to define the position (relative to the node) of the outermost or the innermost circle of the radial gradient.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, OnInit, ViewChild } from '@angular/core';
import { DiagramComponent, Diagram, NodeModel, ShapeStyleModel, RadialGradientModel } from '@syncfusion/ej2-angular-diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%" height="580px" [getNodeDefaults] ='getNodeDefaults'>
    <e-nodes>
      <e-node id='node1' [offsetX]=150 [offsetY]=150 [style]='style'></e-node>
    </e-nodes>
  </ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram?: DiagramComponent;
  public style?: ShapeStyleModel;
  public radialGradient: RadialGradientModel = {
    //Center point of outer circle
    cx: 50,

```

```

        cy: 50,
        //Center point of inner circle
        fx: 25,
        fy: 25,
        //Radius of a radial gradient
        r: 50,
        //Sets an array of stop objects
        stops: [{
            color: 'white',
            offset: 0
        },
        {
            color: '#6BA5D7',
            offset: 100
        }
        ],
        type: 'Radial'
    };
    public getNodeDefaults(node: NodeModel): NodeModel {
        node.height = 100;
        node.width = 100;
        return node;
    }
    ngOnInit(): void {
        this.style = { gradient: this.radialGradient };
    }
}

```

MAIN.TS

```

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

```

Shadow

Diagram provides support to add [shadow](#) effect to a node that is disabled, by default. It can be enabled with the constraints property of the node. The following code illustrates how to drop shadow.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, OnInit, ViewChild } from '@angular/core';
import { DiagramComponent, Diagram, NodeModel, NodeConstraints, ShapeStyleModel } from '@syncfusion/ej2-angular-diagrams';
@Component({
    imports: [
        DiagramModule
    ],
    providers: [ ],
    standalone: true,
    selector: "app-container",

```

```

    template: `<ejs-diagram #diagram id="diagram" width="100%"
height="580px" [getNodeDefaults] ='getNodeDefaults'>
    <e-nodes>
        <e-node id='node1' [offsetX]=150 [offsetY]=150
[constraints]='constraints'></e-node>
    </e-nodes>
</ejs-diagram>`,
    encapsulation: ViewEncapsulation.None
})
export class AppComponent {
    @ViewChild("diagram")
    public diagram?: DiagramComponent;
    public getNodeDefaults(node: NodeModel): NodeModel {
        node.height = 100;
        node.width = 100;
        ((node as NodeModel).style as ShapeStyleModel).fill = "#6BA5D7";
        ((node as NodeModel).style as ShapeStyleModel).strokeColor =
"White";
        return node;
    }
    public constraints?: NodeConstraints;
    ngOnInit(): void {
        this.constraints = NodeConstraints.Default | NodeConstraints.Shadow;
    }
}

```

MAIN.TS

```

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

```

Customizing shadow

The angle, distance, and opacity of the shadow can be customized with the shadow property of the node. The following code example illustrates how to customize shadow.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, OnInit, ViewChild } from
'@angular/core';
import { DiagramComponent, Diagram, NodeModel, NodeConstraints, ShadowModel,
ShapeStyleModel } from '@syncfusion/ej2-angular-diagrams';
@Component({
    imports: [
        DiagramModule
    ],
    providers: [ ],
    standalone: true,
    selector: "app-container",
    template: `<ejs-diagram #diagram id="diagram" width="100%"
height="580px" [getNodeDefaults] ='getNodeDefaults'>

```



```

    <e-nodes>
      <e-node id='node1' [offsetX]=150 [offsetY]=150
[constraints]='constraints' [shadow]='shadow'></e-node>
    </e-nodes>
  </ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram?: DiagramComponent;
  public getNodeDefaults(node: NodeModel): NodeModel {
    node.height = 100;
    node.width = 100;
    ((node as NodeModel).style as ShapeStyleModel).fill = "#6BA5D7";
    ((node as NodeModel).style as ShapeStyleModel).strokeColor =
    "White";
    return node;
  }
  public constraints?: NodeConstraints;
  public shadow?: ShadowModel;
  ngOnInit(): void {
    this.constraints = NodeConstraints.Default | NodeConstraints.Shadow;
    this.shadow = {
      angle: 50,
      opacity: 0.8,
      distance: 9
    }
  }
}

```

MAIN.TS

```

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

```

Icon

Diagram provides support to describe the state of the node. i.e., the node is expanded or collapsed state.

Note: Icon can be created only when the node has outEdges.

- To explore the properties of expand and collapse icon, refer to [expandIcon](#) and [collapseIcon](#).
- The expandIcon's and collapseIcon's shape properties allow to define the shape of the icon.

The following code example illustrates how to create an icon of various shapes.

APP.COMPONENT.TS

```

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

```

```

import { DiagramComponent, Diagram, NodeModel, ShapeModel, ShapeStyleModel }
from '@syncfusion/ej2-angular-diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%"
height="580px" [getNodeDefaults] ='getNodeDefaults'>
    <e-nodes>
      <e-node id='node1' [offsetX]=150 [offsetY]=150
[expandIcon]='expandIcon' [collapseIcon]='collapseIcon'>
        <e-node-annotations>
          <e-node-annotation content="Node1">
            </e-node-annotation>
          </e-node-annotations>
        </e-node>
      <e-node id='node2' [offsetX]=350 [offsetY]=150>
        <e-node-annotations>
          <e-node-annotation content="Custom Template">
            </e-node-annotation>
          </e-node-annotations>
        </e-node>
      </e-nodes>
      <e-connectors>
        <e-connector id='connector' type='Straight' sourceID='node1'
targetID='node2'>
          </e-connector>
        </e-connectors>
      </ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram?: DiagramComponent;
  public getNodeDefaults(node: NodeModel): NodeModel {
    node.height = 100;
    node.width = 100;
    ((node as NodeModel).style as ShapeStyleModel).fill = "#6BA5D7";
    ((node as NodeModel).style as ShapeStyleModel).strokeColor =
    "White";
    return node;
  }
  public expandIcon?: ShapeModel;
  public collapseIcon?: ShapeModel;
  ngOnInit(): void {
    this.expandIcon = {
      shape: 'ArrowDown',
      width: 10,
      height: 10
    } as ShapeModel;
    this.collapseIcon = {
      shape: 'ArrowUp',
      width: 10,
      height: 10
    } as ShapeModel;
  }
}

```

```
    } as ShapeModel;  
  }  
}
```

MAIN.TS

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

Customizing expand icon

- Set the `borderColor`, `borderWidth`, and background color for an `expandIcon` using `borderColor`, `borderWidth`, and `fill` properties.
- Set a size for an `expandIcon` by using `width` and `height` properties.
- The expand icon can be aligned relative to the node boundaries. It has `margin`, `offset`, `horizontalAlignment`, and `verticalAlignment` settings. It is quite tricky, when all four alignments are used together but gives you more control over alignment.
- The [iconColor](#) property can be used to set the `strokeColor` of the Icon.

Customizing collapse icon

- Set the [borderColor](#), [borderWidth](#), background color for an `collapseIcon` using `borderColor`, `borderWidth`, and [fill](#) properties.
- Set a size for `collapseIcon` by using [width](#) and [height](#) properties.
- Like expand icon, collapse icon also can be aligned relative to the node boundaries. It has `margin`, `offset`, `horizontalAlignment`, and `verticalAlignment` settings. It is quite tricky, when all four alignments are used together but gives you more control over alignment.
- The [iconColor](#) property can be used to set the `strokeColor` of the Icon.

Interaction

Diagram provides support to drag, resize, or rotate the node interactively. For more information about editing a node at runtime, refer to [Edit Nodes](#).

Constraints

The constraints property of the node allows you to enable/disable certain features. For more information about node constraints, refer to [Node Constraints](#).

Custom properties

The [addInfo](#) property of the node allows to maintain additional information to the node.

Stack order

The nodes `z-order` property specifies the stack order of the node. A node with greater stack order is always in front of a node with a lower stack order.

Data flow

Node has the `InEdges` and `OutEdges` read-only property. In this property, you can find what are all the connectors that are connected to the node, and then you can find these connectors by using the [getObject](#) method in the diagram.

```
`typescript
@Component({
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%" height="580px" [getNodeDefaults]
    ='getNodeDefaults' (created)='created($event)'>
    <e-nodes>
    <e-node id='node1' [offsetX]=450 [offsetY]=100 [expandIcon]='expandIcon'
    [collapseIcon]='collapseIcon'>
    <e-node-annotations>
    <e-node-annotation content="Node1">
    </e-node-annotation>
    </e-node-annotations>
    </e-node>
    <e-node id='node2' [offsetX]=350 [offsetY]=200>
    <e-node-annotations>
    <e-node-annotation content="Node2">
    </e-node-annotation>
    </e-node-annotations>
    </e-node>
    <e-node id='node3' [offsetX]=450 [offsetY]=200>
    <e-node-annotations>
    <e-node-annotation content="Node3">
    </e-node-annotation>
    </e-node-annotations>
    </e-node>
    <e-node id='node4' [offsetX]=550 [offsetY]=200>
    <e-node-annotations>
    <e-node-annotation content="Node4">
    </e-node-annotation>
    </e-node-annotations>
    </e-node>
    </e-nodes>
    <e-connectors>
    <e-connector id='connector1' type='Orthogonal' sourceID='node1' targetID='node2'>
```

```
</e-connector>
<e-connector id='connector2' type='Orthogonal' sourceID='node1' targetID='node3'>
</e-connector>
<e-connector id='connector3' type='Orthogonal' sourceID='node1' targetID='node4'>
</e-connector>
</e-connectors>
</ejs-diagram>`,
encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram: DiagramComponent;
  public getNodeDefaults(node: NodeModel): NodeModel {
    node.height = 80;
    node.width = 50;
    node.style.fill = "#6BA5D7";
    node.style.strokeColor = "White";
    return node;
  }
  public created(args: Object): void {
    //Add Node
    this.diagram.getObject('connector1');
  }
}
```

See Also

- [How to add annotations to the node](#)
- [How to add ports to the node](#)
- [How to enable/disable the behavior of the node](#)
- [How to add nodes to the symbol palette](#)
- [How to edit the node visual interface](#)
- [How to create diagram nodes using drawing tools](#)

Shapes in Angular Diagram component

Diagram provides support to add different kind of nodes. They are as follows:

- Text node
- Image node
- HTML node
- Native node
- Basic shapes
- Flow shapes

<!-- markdownlint-disable MD033 -->

<!-- markdownlint-disable MD010 -->

Text

Texts can be added to the diagram as [text](#) nodes. The shape property of the node allows you to set the type of node and for text nodes, it should be set as **text**. In addition, define the content object that is used to define the text to be added and style is used to customize the appearance of that text. The following code illustrates how to create a text node.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import { Diagram, NodeModel, TextModel, TextStyleModel } from
 '@syncfusion/ej2-diagrams';
import { DiagramComponent, ShapeStyleModel } from '@syncfusion/ej2-angular-
diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%"
height="580px" [getNodeDefaults] ='getNodeDefaults'>
  <e-nodes>
    <e-node id='node1' [offsetX]=150 [offsetY]=150
[shape]='shape'></e-node>
  </e-nodes>
</ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild('diagram')
  public diagram?: DiagramComponent;
  public shape: TextModel = {
    type: 'Text',
    content: 'Text Element'
  };
  public getNodeDefaults(node: NodeModel): NodeModel {
    node.height = 100;
    node.width = 100;
    ((node as NodeModel).style as ShapeStyleModel).fill = 'none';
    ((node as NodeModel).style as ShapeStyleModel).strokeColor =
'none';
```

```

        (node.style as TextStyleModel).color = 'black';
        (node.style as TextStyleModel).textAlign = 'Center';
        return node;
    }
}

```

MAIN.TS

```

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

```

Image

Diagram allows to add images as [image](#) nodes. The shape property of node allows you to set the type of node and for image nodes, it should be set as **image**. In addition, the source property of shape enables you to set the image source.

The following code illustrates how an image node is created.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import { Diagram, NodeModel, ImageElement, TextStyleModel } from
 '@syncfusion/ej2-diagrams';
import { DiagramComponent, ShapeStyleModel } from '@syncfusion/ej2-angular-
diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%"
height="580px" [getNodeDefaults] ='getNodeDefaults'>
    <e-nodes>
      <e-node id='node1' [offsetX]=150 [offsetY]=150
[shape]='shape'></e-node>
    </e-nodes>
  </ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild('diagram')
  public diagram?: DiagramComponent;
  public shape: ImageElement = {
    type: 'Image',
    source:
'https://ej2.syncfusion.com/demos/src/diagram/employees/image16.png'
  } as unknown as ImageElement;
  public getNodeDefaults(node: NodeModel): NodeModel {

```

```

    node.height = 100;
    node.width = 100;
    ((node as NodeModel).style as ShapeStyleModel).fill = 'none';
    ((node as NodeModel).style as ShapeStyleModel).strokeColor =
'none';
    return node;
  }
}

```

MAIN.TS

```

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

```

Base64 Encoded Image Into The Image Node:

The following code illustrates how add Base64 image into image node.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import { Diagram, NodeModel, ImageElement, TextStyleModel } from
 '@syncfusion/ej2-diagrams';
import { DiagramComponent, ShapeStyleModel } from '@syncfusion/ej2-angular-
diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%"
height="580px" [getNodeDefaults] = 'getNodeDefaults'>
    <e-nodes>
      <e-node id='node1' [offsetX]=150 [offsetY]=150
[shape]='shape'></e-node>
    </e-nodes>
  </ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild('diagram')
  public diagram?: DiagramComponent;
  public shape: ImageElement = {
    type: 'Image',
    source:
'data:image/gif;base64,R0lGODlhPQBEAPeoAJosM//AwO/AwHVVZ/z595kzAP/s7P+goOXMv
8+fhw/v739/f+8PD98fH/8mJl+fn/9ZWb8/PzWlWv///6wWGbImAPgTEMImIN9gUFCEm/gDALULD
N8PAD6atYdCTX9gUNKlj8wZAKUsAOzZz+UMAOsJAP/Z2ccMDA8PD/95eX5NWvsJCOVNQPtfX/8zM
8+QePLl38MGBr8JCP+zs9myn/8GBqwpAP/GxgwJCPny78lzlYlgjAJ8vAP9fX/+MjMUcAN8zM/9wc
'

```



```

M8ZGcATEL+QePdZWf/29uc/P9cmJu9MTDImIN+/r7+/vz8/P8VNQGNugV8AAF9fX8swMNgtAF1DO
ICAgPNSUnNWSMQ5MBAQEJE3QPIGAM9AQMGcG9vb6MhJsEdGM8vLx8fH98AANIWAMuQeL8fABkTE
PPQ00M5OSYdGF15jo+Pj/+pqcsTE78wMFNGQLYmID4dGPvd3UBAQJmTkP+8vH9QUK+vr8ZWSHpzc
JMmILdwcLOGcHRQUHxwcK9PT9DQ00/v70w5MLypoG8wKOuwsP/g4P/Q0IcwKEswKML8aJ9fX2xjd
OtGRs/Pz+Dg4GImIP8gIH0sKEAwKKmTiKZ8aB/f39Ws1+Lft8dgUE9PT5x5aHBwcP+AgP+WltdgY
MyZfyywz78AAAAAAD///8AAP9mZv///wAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
CH5BAEAAKgalAAAAA9AEQAAj/AFEJHEiwoMGDCBMqXMiwoCAbBww4nEhxoYkUpzJGrMixogkfG
UNq1NixJEIDB0SqHGmyJSojM1bKZOmyop0gM3Oe2liTISKMOoPy7GnwY9CjIYcSRYm0aVKSLEm6n
fq05QycVLPUhDrxBlCtYJUqNAq2bNWEBj6ZXRUyxZyDRtqwnXvkhACDV+euTeJm1Ki7A73qNWtFi
F+/gA95Gly2CJLDhwEHMOUAAuOpLYDEgBxZ4GRT1C1fDnpkM+fOqD6DDj1aZpITp0dtGCDhr+fVu
Cu3zlg49ijaokTZTo27uG7Gjn2P+hI8+PDPERoUB318bWbfAJ5sUNFCuGRTYUqV/3ogfXp1rWlMc
6awJjiAAd2fm4ogXjz56aypOoIde4OE5u/F9x199dlXnnGiHZWEYbGpsAEA3QXYnHwEFlIKAgswg
J8LPeiUXGwedCAKABACCN+EA1pYIIYaFlcDhytd5lsGAJbo3onOpajiihl092KHGaUXGwWjUBChj
SPiWJu00/LYIm4v1tXfE6J4gCSJEZ7YgRYUNrkji9P55sF/ogxw5ZkSqIDaZBV6aSGYq/lGZplnd
kckZ98xoICbTcIJGAZcNmduUc210hs35nCyJ58fgmIKX5RQGOZowxaZwYA+JaoKQwsWgijBV4C6
SiTUmpphMspJx9unX4KaimjDv9aaXOEbteBqmuuxgEhOLX6Kqx+yXqqBANsgCtit4FWQAEkrNbpq
7HSOmtwag5w57GrmlJBASEU18ADjUYb3ADTinIttsgSB1oJffa63bduimuqKB1keqWUhoCSK374w
bujvOSu4QG6UvxBrydcpKsav++Ca6G8A6Pr1x2kVMYHwsVxUALDq/krrrhPSOzXG1lUTIoffqGR7
Goi2MAxbv602kEG56I7CS1RsEFKFVYovDJoIRTg7sugNRDGqCJzJgcKE0ywc0ELm6KBCCJo8DIPF
eCWNGcyqNFE06ToAfV0HBRgxsvLThHn1oddQMrXj5DyAQgjEHSAJMWZwS3HPxT/QMbabi/iBCliM
LEJKX2EEekomBAUCxRi42VDADxyTYDVogV+wSChqmKxEKCDAYFDFj40mwbY7bDGDhtrntQYOigeC
hUmclK3QTnAUfEgGFgAwT88hKA6aCRIXhxnQ1yg3BCayK44EWdkUQcBBYEQChFXfCB776aQsG0BI
lQgQgE8qO26X1h8cEUep8ngRBnOy74E9QgRgEAC8SvOfQkh7FDBDmS43PmGoIiKUUEGkMEC/PJHg
xw0xH74yx/3XnaYRJgMB8obxQW6kL9QYEJ0FIFgByfIL7/IQAlvQwEpnAC7DtLNJCKUoO/w45c44
GwCXiAFB/OXAATQryUxdN4LfFiwgjCNYg+kYMIEFkCKDs6PKAIJouyGWMS1FSKJOMRB/BoIxYJIU
XFUXNwoIkEKPAGCBZSQHQ1A2EWDFDEUVLYAdj5AchSIQW6gu10bE/JG2VnZCGfo4R4d0sdQoBAHh
PjhIB94v/wRoRKQWGRHgrhGSQJxCS+0pCZbEhAAOW=='
    } as unknown as ImageElement;
    public getNodeDefaults(node: NodeModel): NodeModel {
        node.height = 100;
        node.width = 100;
        ((node as NodeModel).style as ShapeStyleModel).fill = 'none';
        ((node as NodeModel).style as ShapeStyleModel).strokeColor =
'none';
        return node;
    }
}

```

MAIN.TS

```

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

```

Note: Deploy your HTML file in the web application and export the diagram (image node) or else the image node will not be exported in the Chrome and Firefox due to security issues. Refer to the following link.

Link 1: <http://asked.online/draw-images-on-canvas-locally-using-chrome/2546077/>

Link 2: <http://stackoverflow.com/questions/4761711/local-image-in-canvas-in-chrome>

Image alignment

Stretch and align the image content anywhere but within the node boundary.

The scale property of the node allows to stretch the image as you desired (either to maintain proportion or to stretch). By default, the [scale](#) property of the node is set as **meet**.

The following code illustrates how to scale or stretch the content of the image node.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import { Diagram, NodeModel, ImageElement, TextStyleModel } from
 '@syncfusion/ej2-diagrams';
import { DiagramComponent, ShapeStyleModel } from '@syncfusion/ej2-angular-
diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%"
height="580px" [getNodeDefaults] = 'getNodeDefaults'>
    <e-nodes>
      <e-node id='node1' [offsetX]=150 [offsetY]=150
[shape]='shape'></e-node>
    </e-nodes>
  </ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild('diagram')
  public diagram?: DiagramComponent;
  public shape: ImageElement = {
    type: 'Image',
    source:
'https://ej2.syncfusion.com/demos/src/diagram/employees/image16.png',
    scale: 'None'
  } as unknown as ImageElement;
  public getNodeDefaults(node: NodeModel): NodeModel {
    node.height = 100;
    node.width = 100;
    ((node as NodeModel).style as ShapeStyleModel).fill = 'none';
    ((node as NodeModel).style as ShapeStyleModel).strokeColor =
'none';
    return node;
  }
}
```

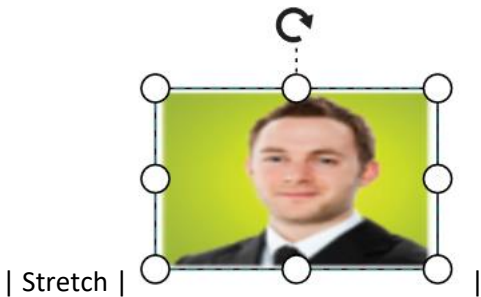
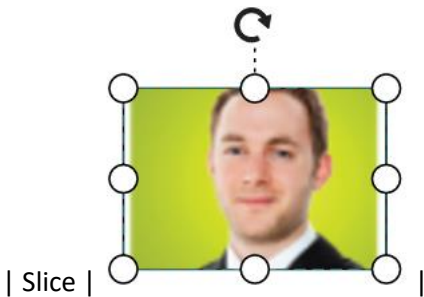
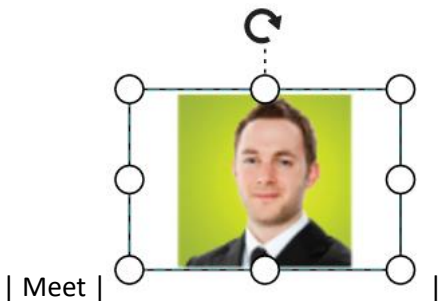
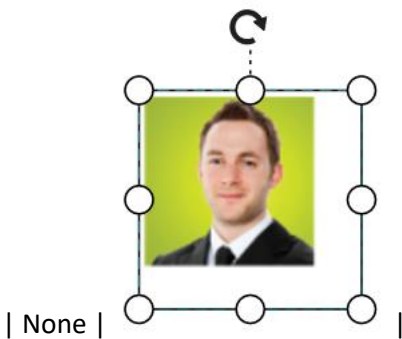
MAIN.TS

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

The following table illustrates all the possible scale options for the image node.

| Values | Images |

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



HTML

Html elements can be embedded in the diagram through [Html](#) type node. The shape property of node allows you to set the type of node and to create a HTML node it should be set as `HTML`. The following code illustrates how an Html node is created.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import { Diagram, NodeModel, HtmlModel, TextStyleModel } from
 '@syncfusion/ej2-diagrams';
import { DiagramComponent, ShapeStyleModel } from '@syncfusion/ej2-angular-
diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%"
height="580px" [getNodeDefaults] ='getNodeDefaults'>
  <e-nodes>
    <e-node id='node1' [offsetX]=150 [offsetY]=150
[shape]='shape'></e-node>
  </e-nodes>
</ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild('diagram')
  public diagram?: DiagramComponent;
  public shape: HtmlModel = {
    content: '<div
style="background:#6BA5D7;height:100%;width:100%;"><button type="button"
style="width:100px"> Button</button></div>',
    type: 'HTML'
  };
  public getNodeDefaults(node: NodeModel): NodeModel {
    node.height = 100;
    node.width = 100;
    ((node as NodeModel).style as ShapeStyleModel).fill = 'none';
    ((node as NodeModel).style as ShapeStyleModel).strokeColor =
'none';
    return node;
  }
}
```

MAIN.TS

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

Note: HTML node cannot be exported to image format, like JPEG, PNG, and BMP. It is by design, while exporting the diagram is drawn in a canvas. Further, this canvas is exported into image formats. Currently, drawing in a canvas equivalent from all possible HTML is not feasible. Hence, this limitation.

HTML Node With Template

Html elements can be embedded in the diagram through [Html](#) type node. The shape property of node allows you to set the type of node. The following code illustrates how an Html node is created with template.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import { Diagram, NodeModel, HtmlModel, TextStyleModel } from
 '@syncfusion/ej2-diagrams';
import { DiagramComponent } from '@syncfusion/ej2-angular-diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%"
height="580px" >
  <ng-template #nodeTemplate let-data >
    <ng-container *ngIf="data.id == 'Node'">
      <input type = "button" value= {{data.id}} >
    </ng-container>
  </ng-template>
  <e-nodes>
    <e-node id='Node' [offsetX]=150 [offsetY]=150 [height]=100
[width]=100 [shape]='shape'></e-node>
  </e-nodes>
</ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild('diagram')
  public diagram?: DiagramComponent;
  public shape: HtmlModel = {
    type: 'HTML'
  };
}
```

MAIN.TS

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

Native

Diagram provides support to embed SVG element into a node. The shape property of node allows you to set the type of node. To create a [native](#) node, it should be set as **native**. The following code illustrates how a native node is created.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import { Diagram, NodeModel, NativeModel, TextStyleModel } from
 '@syncfusion/ej2-diagrams';
import { DiagramComponent, ShapeStyleModel } from '@syncfusion/ej2-angular-
diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%"
height="580px" [getNodeDefaults] ='getNodeDefaults'>
    <e-nodes>
      <e-node id='node1' [offsetX]=150 [offsetY]=150
[shape]='shape'></e-node>
    </e-nodes>
  </ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild('diagram')
  public diagram?: DiagramComponent;
  public shape: NativeModel = {
    content: '<g xmlns="http://www.w3.org/2000/svg"> <g
transform="translate(1 1)"><g>    <path style="fill:#61443C;"
d="M61.979,435.057c2.645-0.512,5.291-0.853,7.936-1.109c-2.01,1.33-4.472,
1.791-6.827,1.28 C62.726,435.13,62.354,435.072,61.979,435.057z"/><path
style="fill:#61443C;"d="M502.469,502.471h-25.6c0.163-30.757-20.173-57.861-
49.749-66.304 c-5.784-1.581-11.753-2.385-17.749-2.389c-2.425-0.028-
4.849,0.114-7.253,0.427c1.831-7.63,2.747-15.45,2.731-23.296 c0.377-47.729-
34.52-88.418-81.749-95.317c4.274-0.545,8.577-0.83,12.885-
0.853c25.285,0.211,49.448,10.466,67.167,28.504
c17.719,18.039,27.539,42.382,27.297,67.666c0.017,7.846-0.9,15.666-
2.731,23.296c2.405-0.312,4.829-0.455,7.253-0.427
C472.572,434.123,502.783,464.869,502.469,502.471z"/>    </g>    <path
style="fill:#8B685A;" d="M476.869,502.471H7.536c-0.191-32.558,22.574-
60.747,54.443-67.413
c0.375,0.015,0.747,0.072,1.109,0.171c2.355,0.511,4.817,0.05,6.827-
1.28c1.707-0.085,3.413-0.171,5.12-0.171
c4.59,0,9.166,0.486,13.653,1.451c2.324,0.559,4.775,0.147,6.787-1.141c2.013-
1.288,3.414-3.341,3.879-5.685    c7.68-39.706,39.605-70.228,79.616-
76.117c4.325-0.616,8.687-0.929,13.056-0.939c13.281-
0.016,26.409,2.837,38.485,8.363
c3.917,1.823,7.708,3.904,11.349,6.229c2.039,1.304,4.527,1.705,6.872,1.106c2.
345-0.598,4.337-2.142,5.502-4.264    c14.373-25.502,39.733-42.923,68.693-
```

```
47.189h0.171c47.229,6.899,82.127,47.588,81.749,95.317c0.017,7.846-
0.9,15.666-2.731,23.296      c2.405-0.312,4.829-0.455,7.253-
0.427c5.996,0.005,11.965,0.808,17.749,2.389C456.696,444.61,477.033,471.713,4
76.869,502.471      L476.869,502.471z"/>      <path style="fill:#66993E;"
d="M502.469,7.537c0,0-6.997,264.96-192.512,252.245c-20.217-1.549-40.166-
5.59-59.392-12.032      c-1.365-0.341-2.731-0.853-4.096-1.28c0,0-0.597-2.219-
1.451-6.144c-6.656-34.048-25.088-198.997,231.765-230.144
C485.061,9.159,493.595,8.22,502.469,7.537z"/>      <path
style="fill:#9ACA5C;" d="M476.784,10.183c-1.28,26.197-16.213,238.165-
166.827,249.6      c-20.217-1.549-40.166-5.59-59.392-12.032c-1.365-0.341-
2.731-0.853-4.096-1.28c0,0-0.597-2.219-1.451-6.144
C238.363,206.279,219.931,41.329,476.784,10.183z"/>      <path
style="fill:#66993E;" d="M206.192,246.727c-0.768,3.925-1.365,6.144-
1.365,6.144c-1.365,0.427-2.731,0.939-4.096,1.28      c-21.505,7.427-
44.293,10.417-66.987,8.789C21.104,252.103,8.816,94.236,7.621,71.452c-0.085-
1.792-0.085-2.731-0.085-2.731
C222.747,86.129,211.653,216.689,206.192,246.727z"/>      <path
style="fill:#9ACA5C;" d="M180.336,246.727c-0.768,3.925-1.365,6.144-
1.365,6.144c-1.365,0.427-2.731,0.939-4.096,1.28      c-13.351,4.412-
27.142,7.359-41.131,8.789C21.104,252.103,8.816,94.236,7.621,71.452
C195.952,96.881,185.541,217.969,180.336,246.727z"/>      </g>      <g>
      <path d="M162.136,426.671c3.451-0.001,6.562-2.08,7.882-5.268s0.591-
6.858-1.849-9.2981-8.533-8.533      c-3.341-3.281-8.701-3.256-12.012,0.054c-
3.311,3.311-3.335,8.671-0.054,12.01218.533,8.533
C157.701,425.773,159.872,426.673,162.136,426.671L162.136,426.671z"/>
      <path d="M292.636,398.57c3.341,3.281,8.701,3.256,12.012-0.054c3.311-
3.311,3.335-8.671,0.054-12.0121-8.533-8.533      c-3.341-3.281-8.701-3.256-
12.012,0.054s-3.335,8.671-0.054,12.012L292.636,398.57z"/>      <path
d="M296.169,454.771c-3.341-3.281-8.701-3.256-12.012,0.054c-3.311,3.311-
3.335,8.671-0.054,12.01218.533,8.533      c3.341,3.281,8.701,3.256,12.012-
0.054c3.311-3.311,3.335-8.671,0.054-12.012L296.169,454.771z"/>
      <path d="M386.503,475.37c3.341,3.281,8.701,3.256,12.012-0.054c3.311-
3.311,3.335-8.671,0.054-12.0121-8.533-8.533      c-3.341-3.281-8.701-3.256-
12.012,0.054c-3.311,3.311-3.335,8.671-0.054,12.012L386.503,475.37z"/>
      <path d="M204.803,409.604c2.264,0.003,4.435-0.897,6.033-
2.518.533-8.533c3.281-3.341,3.256-8.701-0.054-12.012      c-3.311-3.311-8.671-
3.335-12.012-0.0541-8.533,8.533c-2.44,2.44-3.169,6.11-1.849,9.298
C198.241,407.524,201.352,409.603,204.803,409.604z"/>      <path
d="M332.803,443.737c2.264,0.003,4.435-0.897,6.033-2.518.533-8.533c3.281-
3.341,3.256-8.701-0.054-12.012      c-3.311-3.311-8.671-3.335-12.012-0.0541-
8.533,8.533c-2.44,2.44-3.169,6.11-1.849,9.298
C326.241,441.658,329.352,443.737,332.803,443.737z"/>      <path
d="M341.336,366.937c2.264,0.003,4.435-0.897,6.033-2.518.533-8.533c3.281-
3.341,3.256-8.701-0.054-12.012      c-3.311-3.311-8.671-3.335-12.012-0.0541-
8.533,8.533c-2.44,2.44-3.169,6.11-1.849,9.298
C334.774,364.858,337.885,366.937,341.336,366.937z"/>      <path
d="M164.636,454.7711-8.533,8.533c-2.188,2.149-3.055,5.307-
2.27,8.271c0.785,2.965,3.1,5.28,6.065,6.065      c2.965,0.785,6.122-
0.082,8.271-2.2718.533-8.533c3.281-3.341,3.256-8.701-0.054-12.012
C173.337,451.515,167.977,451.49,164.636,454.771L164.636,454.771z"/>
      <path d="M232.903,429.1711-8.533,8.533c-2.188,2.149-3.055,5.307-
2.27,8.271c0.785,2.965,3.1,5.28,6.065,6.065      c2.965,0.785,6.122-
0.082,8.271-2.2718.533-8.533c3.281-3.341,3.256-8.701-0.054-12.012
C241.604,425.915,236.243,425.89,232.903,429.171L232.903,429.171z"/>
      <path d="M384.003,409.604c2.264,0.003,4.435-0.897,6.033-2.518.533-
8.533c3.281-3.341,3.256-8.701-0.054-12.012      c-3.311-3.311-8.671-3.335-
12.012-0.0541-8.533,8.533c-2.44,2.44-3.169,6.11-1.849,9.298
```

```

C377.441,407.524,380.552,409.603,384.003,409.604z"/>      <path
d="M70.77,463.304l-8.533,8.533c-2.188,2.149-3.055,5.307-
2.27,8.271s3.1,5.28,6.065,6.065      c2.965,0.785,6.122-0.082,8.271-
2.2718.533-8.533c3.281-3.341,3.256-8.701-0.054-12.012
C79.47,460.048,74.11,460.024,70.77,463.304L70.77,463.304z"/>      <path
d="M121.97,446.238l-8.533,8.533c-2.188,2.149-3.055,5.307-
2.27,8.271c0.785,2.965,3.1,5.28,6.065,6.065      c2.965,0.785,6.122-
0.082,8.271-2.2718.533-8.533c3.281-3.341,3.256-8.701-0.054-12.012
C130.67,442.981,125.31,442.957,121.97,446.238L121.97,446.238z"/>
      <path d="M202.302,420.638c-1.6-1.601-3.77-2.5-6.033-2.5c-2.263,0-
4.433,0.899-6.033,2.51-8.533,8.533      c-2.178,2.151-3.037,5.304-
2.251,8.262c0.786,2.958,3.097,5.269,6.055,6.055c2.958,0.786,6.111-
0.073,8.262-2.25118.533-8.533      c1.601-1.6,2.5-3.77,2.5-
6.033C204.802,424.408,203.903,422.237,202.302,420.638L202.302,420.638z"/>
      <path d="M210.836,463.304c-3.341-3.281-8.701-3.256-
12.012,0.054c-3.311,3.311-3.335,8.671-0.054,12.012l8.533,8.533
c2.149,2.188,5.307,3.055,8.271,2.27c2.965-0.785,5.28-3.1,6.065-6.065c0.785-
2.965-0.082-6.122-2.27-8.271L210.836,463.304z"/>      <path
d="M343.836,454.771l-8.533,8.533c-2.188,2.149-3.055,5.307-
2.27,8.271c0.785,2.965,3.1,5.28,6.065,6.065      c2.965,0.785,6.122-
0.082,8.271-2.2718.533-8.533c3.281-3.341,3.256-8.701-0.054-12.012
C352.537,451.515,347.177,451.49,343.836,454.771L343.836,454.771z"/>
      <path d="M429.17,483.904c3.341,3.281,8.701,3.256,12.012-0.054s3.335-
8.671,0.054-12.012l-8.533-8.533      c-3.341-3.281-8.701-3.256-12.012,0.054c-
3.311,3.311-3.335,8.671-0.054,12.012L429.17,483.904z"/>      <path
d="M341.336,401.071c2.264,0.003,4.435-0.897,6.033-2.518.533-8.533c3.281-
3.341,3.256-8.701-0.054-12.012      s-8.671-3.335-12.012-0.054l-8.533,8.533c-
2.44,2.441-3.169,6.11-
1.849,9.298C334.774,398.991,337.885,401.07,341.336,401.071z"/>
      <path d="M273.069,435.204c2.264,0.003,4.435-0.897,6.033-2.518.533-
8.533c3.281-3.341,3.256-8.701-0.054-12.012      s-8.671-3.335-12.012-0.054l-
8.533,8.533c-2.44,2.44-3.169,6.11-
1.849,9.298C266.508,433.124,269.618,435.203,273.069,435.204z"/>
      <path
d="M253.318,258.138c22.738,7.382,46.448,11.338,70.351,11.737c31.602,0.543,62
.581-8.828,88.583-26.796      c94.225-65.725,99.567-227.462,99.75-
234.317c0.059-2.421-0.91-4.754-2.667-6.421c-1.751-1.679-4.141-2.52-6.558-
2.308      C387.311,9.396,307.586,44.542,265.819,104.5c-28.443,42.151-
38.198,94.184-26.956,143.776c-3.411,8.366-6.04,17.03-7.852,25.881      c-
4.581-7.691-9.996-14.854-16.147-21.358c8.023-38.158,0.241-77.939-21.57-
110.261C160.753,95.829,98.828,68.458,9.228,61.196      c-2.417-0.214-
4.808,0.628-6.558,2.308c-1.757,1.667-2.726,4-
2.667,6.421c0.142,5.321,4.292,130.929,77.717,182.142
c20.358,14.081,44.617,21.428,69.367,21.008c18.624-0.309,37.097-3.388,54.814-
9.138c11.69,12.508,20.523,27.407,25.889,43.665
c0.149,15.133,2.158,30.19,5.982,44.832c-12.842-5.666-26.723-8.595-40.759-
8.6c-49.449,0.497-91.788,35.567-101.483,84.058      c-5.094-1.093-10.29-1.641-
15.5-1.638c-42.295,0.38-76.303,34.921-76.025,77.217c-
0.001,2.263,0.898,4.434,2.499,6.035
c1.6,1.6,3.771,2.499,6.035,2.499h494.933c2.263,0.001,4.434-0.898,6.035-
2.499c1.6-1.6,2.499-3.771,2.499-6.035      c0.249-41.103-31.914-75.112-72.967-
77.154c0.65-4.78,0.975-9.598,0.975-14.421c0.914-45.674-28.469-86.455-72.083-
100.045      c-43.615-13.59-90.962,3.282-
116.154,41.391C242.252,322.17,242.793,288.884,253.318,258.138L253.318,258.13
8z M87.519,238.092      c-55.35-38.567-67.358-129.25-69.833-
158.996c78.8,7.921,133.092,32.454,161.458,72.992
c15.333,22.503,22.859,49.414,21.423,76.606c-23.253-35.362-77.83-105.726-

```



```

162.473-140.577c-2.82-1.165-6.048-0.736-8.466,1.125      s-3.658,4.873-
3.252,7.897c0.406,3.024,2.395,5.602,5.218,6.761c89.261,36.751,144.772,117.77
6,161.392,144.874      C150.795,260.908,115.29,257.451,87.519,238.092z
M279.969,114.046c37.6-53.788,109.708-86.113,214.408-96.138      c-2.65,35.375-
17.158,159.05-91.892,211.175c-37.438,26.116-85.311,30.57-142.305,13.433
c19.284-32.09,92.484-142.574,212.405-191.954c2.819-1.161,4.805-3.738,5.209-
6.76c0.404-3.022-0.835-6.031-3.25-7.892      c-2.415-1.861-5.64-2.292-8.459-
1.131C351.388,82.01,279.465,179.805,252.231,222.711
C248.573,184.367,258.381,145.945,279.969,114.046L279.969,114.046z
M262.694,368.017c15.097-26.883,43.468-43.587,74.3-43.746
c47.906,0.521,86.353,39.717,85.95,87.625c-0.001,7.188-0.857,14.351-
2.55,21.337c-0.67,2.763,0.08,5.677,1.999,7.774
c1.919,2.097,4.757,3.1,7.568,2.676c1.994-0.272,4.005-0.393,6.017-
0.362c29.59,0.283,54.467,22.284,58.367,51.617H17.661      c3.899-
29.333,28.777-51.334,58.367-51.617c4-
0.004,7.989,0.416,11.9,1.254c4.622,0.985,9.447,0.098,13.417-2.467      c3.858-
2.519,6.531-6.493,7.408-11.017c7.793-40.473,43.043-69.838,84.258-
70.192c16.045-0.002,31.757,4.582,45.283,13.212
c4.01,2.561,8.897,3.358,13.512,2.205C256.422,375.165,260.36,372.163,262.694,
368.017L262.694,368.017z"/>      </g></g>',
    type: 'Native'
  };
  public getNodeDefaults(node: NodeModel): NodeModel {
    node.height = 100;
    node.width = 100;
    ((node as NodeModel).style as ShapeStyleModel).fill = 'none';
    ((node as NodeModel).style as ShapeStyleModel).strokeColor =
'none';
    return node;
  }
}

```

MAIN.TS

```

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

```

Note: Like HTML node, the native node also cannot be exported to image format. Fill color of native node can be overridden by the inline style or fill of the SVG element specified in the template.

SVG content alignment

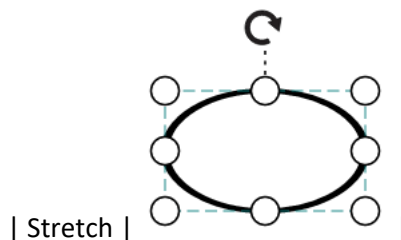
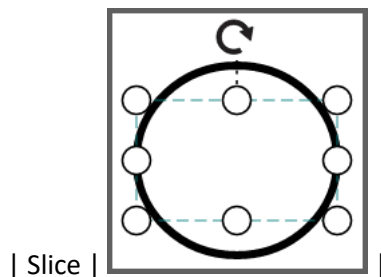
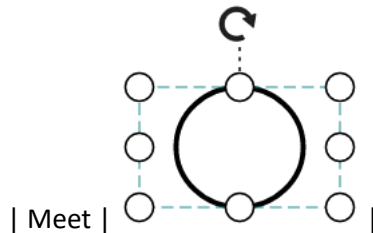
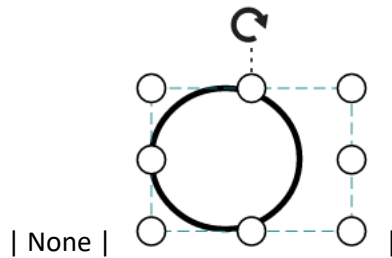
Stretch and align the svg content anywhere but within the node boundary.

The scale property of the node allows to stretch the svg content as you desired (either to maintain proportion or to stretch). By default, the `scale` property of node is set as **meet**.

The following tables illustrates all the possible scale options for the node.

| Values | Images |

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



Basic shapes

- The [Basic](#) shapes are common shapes that are used to represent the geometrical information visually. To create basic shapes, the type of the shape should be set as **basic**. Its shape property can be set with any one of the built-in shape.
- To render a rounded rectangle, you need to set the type as basic and shape as rectangle. Set the [cornerRadius](#) property to specify the radius of rounded rectangle.

The following code example illustrates how to create a basic shape.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import { Diagram, NodeModel, BasicShapeModel , TextStyleModel } from
 '@syncfusion/ej2-diagrams';
import { DiagramComponent, ShapeStyleModel } from '@syncfusion/ej2-angular-
diagrams';
```

```

@Component({
  imports: [
    DiagramModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%"
height="580px" [getNodeDefaults] ='getNodeDefaults'>
  <e-nodes>
    <e-node id='node1' [offsetX]=150 [offsetY]=150
[shape]='shape'></e-node>
  </e-nodes>
</ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild('diagram')
  public diagram?: DiagramComponent;
  public shape: BasicShapeModel = {
    type: 'Basic',
    shape: 'Rectangle',
    cornerRadius: 10
  };
  public getNodeDefaults(node: NodeModel): NodeModel {
    node.height = 100;
    node.width = 100;
    ((node as NodeModel).style as ShapeStyleModel).fill = 'none';
    return node;
  }
}

```

MAIN.TS

```

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

```

Note: By default, the **shape** property of the node is set as **basic**.

The default property for shape is **Rectangle**.

Note: When the **shape** is not set for a basic shape, it is considered as a **rectangle**.

The list of basic shapes are as follows.



Path

The [Path](#) node is a commonly used basic shape that allows visually to represent the geometrical information. To create a path node, specify the shape as **path**. The path property of node allows you to define the path to be drawn. The following code illustrates how a path node is created.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import { Diagram, NodeModel, PathModel, TextStyleModel } from
 '@syncfusion/ej2-diagrams';
import { DiagramComponent, ShapeStyleModel } from '@syncfusion/ej2-angular-
diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%"
height="580px" [getNodeDefaults] ='getNodeDefaults'>
  <e-nodes>
    <e-node id='node1' [offsetX]=150 [offsetY]=150
[shape]='shape'></e-node>
  </e-nodes>
</ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild('diagram')
  public diagram?: DiagramComponent;
  public shape: PathModel = {
    type: 'Path',
    data: 'M35.2441,25 L22.7161,49.9937 L22.7161,0.00657536 L35.2441,25
z M22.7167,25 L-0.00131226,25 M35.2441,49.6337 L35.2441,0.368951 M35.2441,25
L49.9981,25'
  };
  public getNodeDefaults(node: NodeModel): NodeModel {
    node.height = 100;
    node.width = 100;
    ((node as NodeModel).style as ShapeStyleModel).fill = 'none';
    return node;
  }
}
```

MAIN.TS

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

Flow Shapes

The [flow](#) shapes are used to represent the process flow. It is used for analyzing, designing, and managing for documentation process. To create a flow shape, specify the shape type as **flow**. Flow shapes and by default, it is considered as **process**. The following code example illustrates how to create a flow shape.

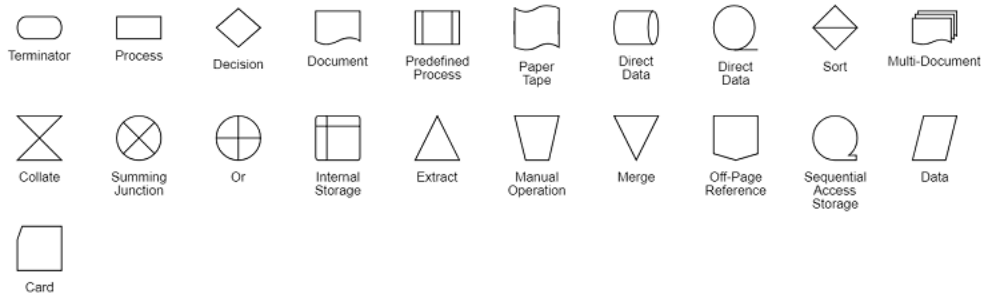
APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import { Diagram, NodeModel, FlowShapeModel, TextStyleModel } from '@syncfusion/ej2-diagrams';
import { DiagramComponent, ShapeStyleModel } from '@syncfusion/ej2-angular-diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%" height="580px" [getNodeDefaults] ='getNodeDefaults'>
    <e-nodes>
      <e-node id='node1' [offsetX]=150 [offsetY]=150
[shape]='shape'></e-node>
    </e-nodes>
  </ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild('diagram')
  public diagram?: DiagramComponent;
  public shape: FlowShapeModel = {
    type: 'Flow',
    shape: 'Document'
  };
  public getNodeDefaults(node: NodeModel): NodeModel {
    node.height = 100;
    node.width = 100;
    ((node as NodeModel).style as ShapeStyleModel).fill = 'none';
    return node;
  }
}
```

MAIN.TS

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

The list of flow shapes are as follows.



Bpmn shapes in Angular Diagram component

BPMN shapes are used to represent the internal business procedure in a graphical notation and enable you to communicate the procedures in a standard manner. To create a BPMN shape, in the node property shape, type should be set as “bpmn” and its shape should be set as any one of the built-in shapes. The following code example illustrates how to create a simple business process.

Note: If you want to use BPMN shapes in diagram, you need to inject BpmnDiagrams in the diagram.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule, BpmnDiagramsService } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import { DiagramComponent, Diagram, BpmnDiagrams, NodeModel, BpmnShapeModel } from '@syncfusion/ej2-angular-diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [BpmnDiagramsService],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%" height="580px" [getNodeDefaults] = 'getNodeDefaults'>
    <e-nodes>
      <e-node id='node1' [offsetX]=150 [offsetY]=150
[shape]='shape'></e-node>
    </e-nodes>
  </ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram?: DiagramComponent;
  public shape: BpmnShapeModel = {
    type: 'Bpmn',
    shape: 'Event',
    // set the event type as End
    event: {
      event: 'End'
    }
  }
  public getNodeDefaults(node: NodeModel): NodeModel {
    node.height = 100;
  }
}
```

```
node.width = 100;  
return node;  
}  
}
```

MAIN.TS

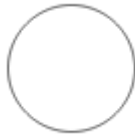
```
import { bootstrapApplication } from '@angular/platform-browser';  
import { AppComponent } from './app.component';  
import 'zone.js';  
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Note : The default value for the property **shape** is “event”.

The list of BPMN shapes are as follows:

| Shape | Image |

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



| Event |



| Gateway |



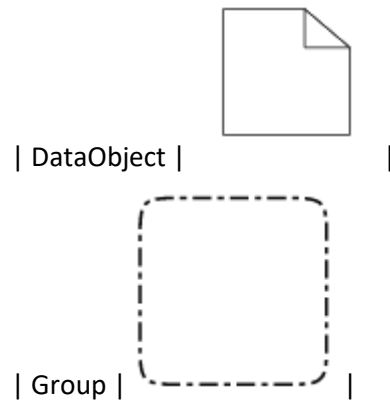
| Task |



| Message |



| DataSource |



The BPMN shapes and its types are explained as follows.

<!-- markdownlint-disable MD033 -->

Event

An [event](#) is notated with a circle and it represents an event in a business process. The type of events are as follows:

- Start
- End
- Intermediate

The event property of the node allows you to define the type of the event. The default value of the event is **Start**. The following code example illustrates how to create a BPMN event.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule, BpmnDiagramsService } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import { DiagramComponent, Diagram, NodeModel, BpmnShapeModel } from '@syncfusion/ej2-angular-diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [BpmnDiagramsService],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%" height="580px" [getNodeDefaults] ='getNodeDefaults'>
    <e-nodes>
      <e-node id='node1' [offsetX]=150 [offsetY]=150 [shape]='shape'></e-node>
    </e-nodes>
  </ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild("diagram")
```



```

public diagram?: DiagramComponent;
public shape: BpmnShapeModel = {
  type: 'Bpmn',
  shape: 'Event',
  // set the event type as End
  event: {
    event: 'End',
    trigger: 'None'
  }
}
public getNodeDefaults(node: NodeModel): NodeModel {
  node.height = 100;
  node.width = 100;
  return node;
}
}

```

MAIN.TS





















```

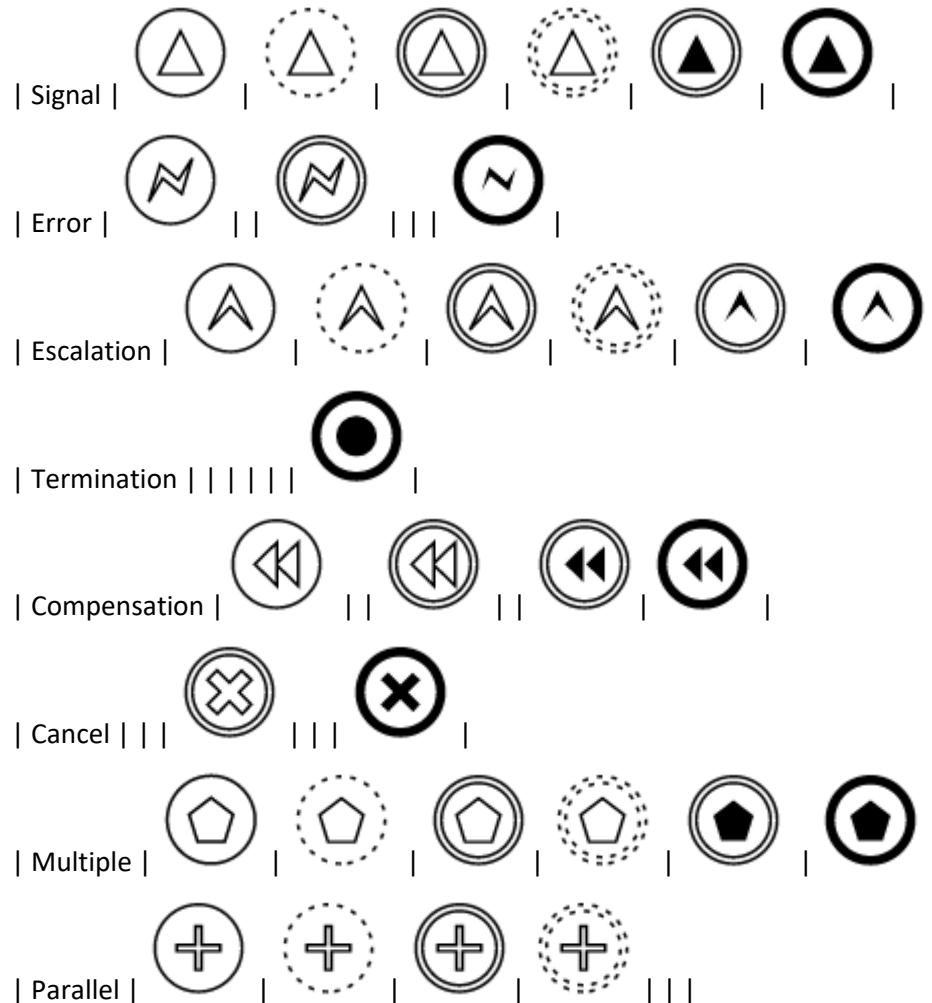
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Event triggers are notated as icons inside the circle and they represent the specific details of the process. The [trigger](#) property of the node allows you to set the type of trigger and by default, it is set as **none**. The following table illustrates the type of event triggers.

| Triggers | Start | Non-Interrupting Start | Intermediate | Non-Interrupting Intermediate | Throwing Intermediate | End |

----- ----- ----- ----- ----- ----- -----						
None						
Message						
Timer						
Conditional						
Link						



Gateway

Gateway is used to control the flow of a process and it is represented as a diamond shape. To create a gateway, the shape property of the node should be set as "gateway" and the [gateway](#) property can be set with any of the appropriate gateways. The following code example illustrates how to create a BPMN Gateway.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule, BpmnDiagramsService } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import { DiagramComponent, Diagram, NodeModel, BpmnShapeModel } from '@syncfusion/ej2-angular-diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [BpmnDiagramsService],
  standalone: true,
  selector: "app-container",
```

```

    template: `<ejs-diagram #diagram id="diagram" width="100%"
height="580px" [getNodeDefaults] ='getNodeDefaults'>
    <e-nodes>
        <e-node id='node1' [offsetX]=150 [offsetY]=150
[shape]='shape'></e-node>
    </e-nodes>
</ejs-diagram>`,
    encapsulation: ViewEncapsulation.None
  })
  export class AppComponent {
    @ViewChild("diagram")
    public diagram?: DiagramComponent;
    public shape: BpmnShapeModel = {
      type: 'Bpmn',
      shape: 'Event',
      // Sets type of the gateway as None
      gateway: {
        type: 'None'
      }
    }
    public getNodeDefaults(node: NodeModel): NodeModel {
      node.height = 100;
      node.width = 100;
      return node;
    }
  }

```

MAIN.TS

```

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

```

Note: By default, the gateway will be set as **none**.

There are several types of gateways as tabulated:

| Shape | Image |

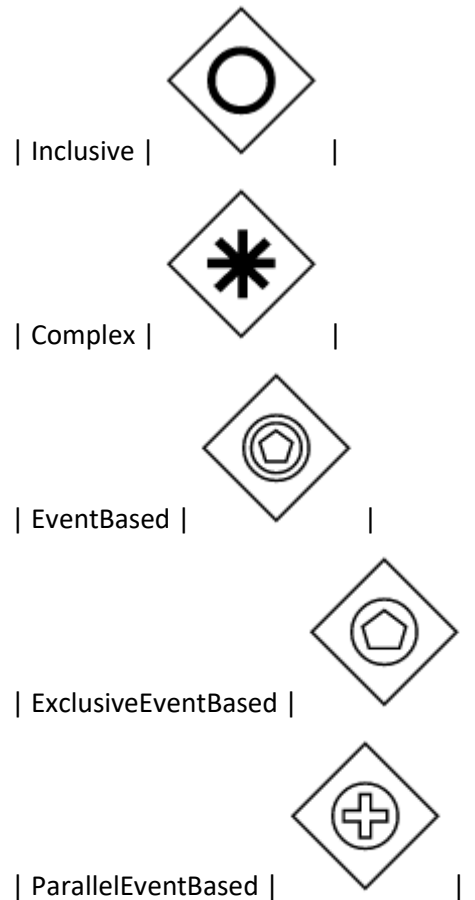
| ----- | ----- |



| Exclusive |



| Parallel |



Activity

The [activity](#) is the task that is performed in a business process. It is represented by a rounded rectangle.

There are two types of activities. They are listed as follows:

- **Task:** Occurs within a process and it is not broken down to a finer level of detail.
- **Subprocess:** Occurs within a process and it is broken down to a finer level of detail.

To create a BPMN activity, set the shape as **activity**. You also need to set the type of the BPMN activity by using the activity property of the node. By default, the type of the activity is set as **task**. The following code example illustrates how to create an activity.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule, BpmnDiagramsService } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import { DiagramComponent, Diagram, NodeModel, BpmnShapeModel } from '@syncfusion/ej2-angular-diagrams';
@Component({
  imports: [
    DiagramModule
  ],
```

```

providers: [BpmnDiagramsService],
standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%"
height="580px" [getNodeDefaults] ='getNodeDefaults'>
    <e-nodes>
      <e-node id='node1' [offsetX]=150 [offsetY]=150
[shape]='shape'></e-node>
    </e-nodes>
  </ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram?: DiagramComponent;
  public shape: BpmnShapeModel = {
    type: 'Bpmn',
    shape: 'Activity',
    //Sets activity as Task
    activity: {
      activity: 'Task'
    },
  }
  public getNodeDefaults(node: NodeModel): NodeModel {
    node.height = 100;
    node.width = 100;
    return node;
  }
}

```

MAIN.TS

```

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

```

The different activities of BPMN process are listed as follows.

Tasks

The [task](#) property of the node allows you to define the type of task such as sending, receiving, user based task, etc. By default, the [type](#) property of task is set as **none**. The following code illustrates how to create different types of BPMN tasks.

The events property of tasks allow to represent these results as an event attached to the task.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule, BpmnDiagramsService } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import { DiagramComponent, Diagram, NodeModel, BpmnShapeModel } from
 '@syncfusion/ej2-angular-diagrams';
@Component({

```

```

imports: [
    DiagramModule
],
providers: [BpmnDiagramsService],
standalone: true,
selector: "app-container",
template: `<ejs-diagram #diagram id="diagram" width="100%"
height="580px" [getNodeDefaults] ='getNodeDefaults'>
    <e-nodes>
        <e-node id='node1' [offsetX]=150 [offsetY]=150
[shape]='shape'></e-node>
    </e-nodes>
</ejs-diagram>`,
encapsulation: ViewEncapsulation.None
})
export class AppComponent {
    @ViewChild("diagram")
    public diagram?: DiagramComponent;
    public shape: BpmnShapeModel = {
        type: 'Bpmn',
        shape: 'Event',
        //Sets activity as Task
        activity: {
            activity: 'Task',
            //Sets the type of the task as Send
            task: {
                type: 'Send'
            }
        },
    }
    public getNodeDefaults(node: NodeModel): NodeModel {
        node.height = 100;
        node.width = 100;
        return node;
    }
}

```

MAIN.TS

```

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

```

The various types of BPMN tasks are tabulated as follows.

Shape	Image
-----	-----



| Service |



| Send |



| Receive |



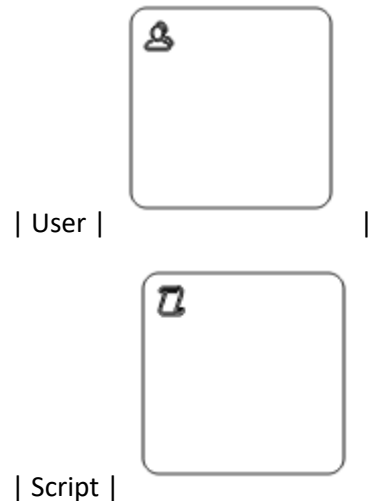
| Instantiating Receive |



| Manual |



| Business Rule |



Subprocess

A [sub-process](#) is a group of tasks, which is used to hide or reveal details of additional levels using the [collapsed](#) property.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule, BpmnDiagramsService } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import { DiagramComponent, Diagram, NodeModel, BpmnShapeModel } from '@syncfusion/ej2-angular-diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [BpmnDiagramsService],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%" height="580px" [getNodeDefaults] = 'getNodeDefaults'>
    <e-nodes>
      <e-node id='node1' [offsetX]=150 [offsetY]=150
[shape]='shape'></e-node>
    </e-nodes>
  </ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram?: DiagramComponent;
  public shape: BpmnShapeModel = {
    type: 'Bpmn',
    shape: 'Event',
    //Sets activity as Task
    activity: {
      activity: 'SubProcess',
      subProcess: {
```



```

        collapsed: true
    },
    },
    },
    public getNodeDefaults(node: NodeModel): NodeModel {
        node.height = 100;
        node.width = 100;
        return node;
    }
}

```

MAIN.TS

```

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

```

The different types of subprocess are as follows:

- Event subprocess
- Transaction

Event subprocess

A subprocess is defined as an event subprocess, when it is triggered by an event. An event subprocess is placed within another subprocess which is not part of the normal flow of its parent process. You can set event to a subprocess with the [event](#) and [trigger](#) property of the subprocess. The [type](#) property of subprocess allows you to define the type of subprocess whether it should be event subprocess or transaction subprocess.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule, BpmnDiagramsService } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import { DiagramComponent, Diagram, NodeModel, BpmnShapeModel } from '@syncfusion/ej2-angular-diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [BpmnDiagramsService],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%"
height="580px" [getNodeDefaults] = 'getNodeDefaults'>
  <e-nodes>
    <e-node id='node1' [offsetX]=150 [offsetY]=150
[shape]='shape'></e-node>
  </e-nodes>
</ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})

```

```

  })
  export class AppComponent {
    @ViewChild("diagram")
    public diagram?: DiagramComponent;
    public shape: BpmnShapeModel = {
      type: 'Bpmn',
      shape: 'Event',
      //Sets activity as Task
      activity: {
        activity: 'SubProcess',
        //Sets the collapsed as true and type as Event
        subprocess: {
          collapsed: true,
          type: 'Event',
          //Sets event as Start and trigger as Message
          event: {
            event: 'Start',
            trigger: 'Message'
          }
        }
      }
    } as BpmnShapeModel;
    public getNodeDefaults(node: NodeModel): NodeModel {
      node.height = 100;
      node.width = 100;
      return node;
    }
  }
}

```

MAIN.TS

```

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

```

Transaction subprocess

- [transaction](#) is a set of activities that logically belong together, in which all contained activities must complete their parts of the transaction; otherwise the process is undone. The execution result of a transaction is one of Successful Completion, Unsuccessful Completion (Cancel), and Hazard (Exception). The `events` property of subprocess allows to represent these results as an event attached to the subprocess.
- The event object allows you to define the type of event by which the subprocess will be triggered. The name of the event can be defined to identify the event at runtime.
- The event's `offset` property is used to set the fraction/ratio (relative to parent) that defines the position of the event shape.
- The `trigger` property defines the type of the event trigger.
- You can also use define ports and labels to subprocess events by using event's `ports` and `labels` properties.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule, BpmnDiagramsService } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import { DiagramComponent, Diagram, NodeModel, BpmnShapeModel } from '@syncfusion/ej2-angular-diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [BpmnDiagramsService],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%"
height="580px" [getNodeDefaults] = 'getNodeDefaults'>
  <e-nodes>
    <e-node id='node1' [offsetX]=150 [offsetY]=150
[shape]='shape'></e-node>
  </e-nodes>
</ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram?: DiagramComponent;
  public shape: BpmnShapeModel = {
    type: 'Bpmn',
    shape: 'Event',
    //Sets activity as Task
    activity: {
      activity: 'SubProcess',
      //Sets the collapsed as true and type as Event
      subProcess: {
        collapsed: true,
        type: 'Transition',
        //Sets event as Intermediate and trigger as Cancel
        event: [{
          event: 'Intermediate',
          trigger: 'Cancel',
          offset: {
            x: 0.25,
            y: 1
          }
        },
        {
          event: 'Intermediate',
          trigger: 'Error',
          offset: {
            x: 0.25,
            y: 1
          }
        }
      ],
    }
  ],
  } as any
},
}

```

```

    public getNodeDefaults(node: NodeModel): NodeModel {
        node.height = 100;
        node.width = 100;
        return node;
    }
}

```

MAIN.TS

```

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

```

Process

Processes is an array collection that defines the children values for BPMN subprocess.

Loop

[Loop](#) is a task that is internally being looped. The loop property of task allows you to define the type of loop. The default value for loop is **none**.

You can define the loop property in subprocess BPMN shape as shown in the following code.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule, BpmnDiagramsService } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import { DiagramComponent, Diagram, NodeModel, BpmnShapeModel } from '@syncfusion/ej2-angular-diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [BpmnDiagramsService],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%" height="580px" [getNodeDefaults] = 'getNodeDefaults'>
    <e-nodes>
      <e-node id='node1' [offsetX]=150 [offsetY]=150
[shape]='shape'></e-node>
    </e-nodes>
  </ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram?: DiagramComponent;
  public shape: BpmnShapeModel = {
    type: 'Bpmn',
    shape: 'Activity',
    //Sets activity as Task
    activity: {

```

```
        activity: 'Task',
        //Sets loop of the task as Standard
        task: {
            loop: 'Standard'
        }
    },
}
public getNodeDefaults(node: NodeModel): NodeModel {
    node.height = 100;
    node.width = 100;
    return node;
}
}
```

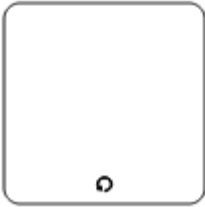
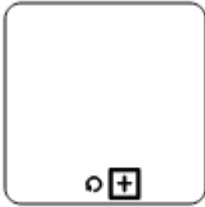



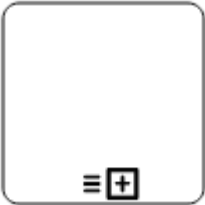
MAIN.TS

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

The following table contains various types of BPMN loops.

| Loops | Task | Subprocess |

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

		
Standard		
		
SequenceMultiInstance		
		
ParallelMultiInstance		

Compensation

[Compensation](#) is triggered, when operation is partially failed and enabled it with the compensation property of the task and the subprocess.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule, BpmnDiagramsService } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import { DiagramComponent, Diagram, NodeModel, BpmnShapeModel } from '@syncfusion/ej2-angular-diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [BpmnDiagramsService],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%" height="580px" [getNodeDefaults] ='getNodeDefaults'>
    <e-nodes>
      <e-node id='node1' [offsetX]=150 [offsetY]=150
[shape]='shape'></e-node>
    </e-nodes>
  </ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram?: DiagramComponent;
  public shape: BpmnShapeModel = {
    type: 'Bpmn',
    shape: 'Activity',
    //Sets activity as Task
    activity: {
      activity: 'Task',
      //Sets compensation of the task as true
      task: {
        compensation: true
      }
    },
  },
  public getNodeDefaults(node: NodeModel): NodeModel {
    node.height = 100;
    node.width = 100;
    return node;
  }
}
```

MAIN.TS

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

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

Call

A [call](#) activity is a global subprocess that is reused at various points of the business flow and set it with the call property of the task.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule, BpmnDiagramsService } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import { DiagramComponent, Diagram, NodeModel, BpmnShapeModel } from
 '@syncfusion/ej2-angular-diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [BpmnDiagramsService],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%"
height="580px" [getNodeDefaults] ='getNodeDefaults'>
  <e-nodes>
    <e-node id='node1' [offsetX]=150 [offsetY]=150
[shape]='shape'></e-node>
  </e-nodes>
</ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram?: DiagramComponent;
  public shape: BpmnShapeModel = {
    type: 'Bpmn',
    shape: 'Activity',
    //Sets activity as Task
    activity: {
      activity: 'Task',
      //Sets the call of the task as true
      task: {
        call: true
      }
    },
  },
}
  public getNodeDefaults(node: NodeModel): NodeModel {
    node.height = 100;
    node.width = 100;
    return node;
  }
}
```

MAIN.TS

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

Adhoc

An adhoc subprocess is a group of tasks that are executed in any order or skipped in order to fulfill the end condition and set it with the [adhoc](#) property of subprocess.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule, BpmnDiagramsService } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import { DiagramComponent, Diagram, NodeModel, BpmnShapeModel } from
 '@syncfusion/ej2-angular-diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [BpmnDiagramsService],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%"
height="580px" [getNodeDefaults] = 'getNodeDefaults'>
  <e-nodes>
    <e-node id='node1' [offsetX]=150 [offsetY]=150
[shape]='shape'></e-node>
  </e-nodes>
</ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram?: DiagramComponent;
  public shape: BpmnShapeModel = {
    type: 'Bpmn',
    shape: 'Activity',
    //Sets activity as Task
    activity: {
      activity: 'SubProcess',
      //Sets collapsed as true and adhoc as true
      subprocess: {
        collapsed: true,
        adhoc: true
      }
    },
  },
  public getNodeDefaults(node: NodeModel): NodeModel {
    node.height = 100;
    node.width = 100;
    return node;
  }
}
```


MAIN.TS

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

Boundary

Boundary represents the type of task that is being processed. The [boundary](#) property of subprocess allows you to define the type of boundary. By default, it is set as **default**.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule, BpmnDiagramsService } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import { DiagramComponent, Diagram, NodeModel, BpmnShapeModel } from '@syncfusion/ej2-angular-diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [BpmnDiagramsService],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%"
height="580px" [getNodeDefaults] ='getNodeDefaults'>
  <e-nodes>
    <e-node id='node1' [offsetX]=150 [offsetY]=150
[shape]='shape'></e-node>
  </e-nodes>
</ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram?: DiagramComponent;
  public shape: BpmnShapeModel = {
    type: 'Bpmn',
    shape: 'Activity',
    //Sets activity as Task
    activity: {
      activity: 'SubProcess',
      //Sets collapsed as true and boundary as Call
      subprocess: {
        collapsed: true,
        boundary: 'Call'
      }
    }
  },
}
  public getNodeDefaults(node: NodeModel): NodeModel {
    node.height = 100;
```

```

    node.width = 100;
    return node;
  }
}

```

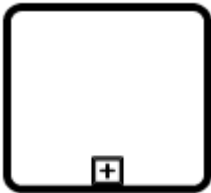


MAIN.TS

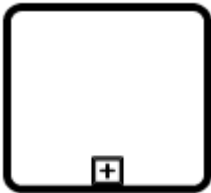
```

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

```

The following table contains various types of BPMN boundaries.

Boundary	Image
-----	-----
Call	
Event	
Default	

**Data**

A data object represents information flowing through the process, such as data placed into the process, data resulting from the process, data that needs to be collected, or data that must be stored. To define a [data object](#), set the shape as **DataObject** and the type property defines whether data is an input or an output. You can create multiple instances of data object with the collection property of data.

APP.COMPONENT.TS

```

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

```

```

import { DiagramModule, BpmnDiagramsService } from '@syncfusion/ej2-angular-diagrams';
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import { DiagramComponent, Diagram, NodeModel, BpmnShapeModel } from '@syncfusion/ej2-angular-diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [BpmnDiagramsService],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%" height="580px" [getNodeDefaults] = 'getNodeDefaults'>
    <e-nodes>
      <e-node id='node1' [offsetX]=150 [offsetY]=150
[shape]='shape'></e-node>
    </e-nodes>
  </ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram?: DiagramComponent;
  public shape: BpmnShapeModel = {
    type: 'Bpmn',
    shape: 'DataObject',
    //Sets collection as true and type as Input
    dataObject: {
      collection: true,
      type: 'Input'
    }
  }
  public getNodeDefaults(node: NodeModel): NodeModel {
    node.height = 100;
    node.width = 100;
    return node;
  }
}

```

MAIN.TS

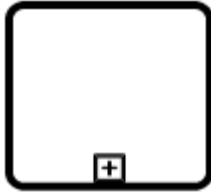
```

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

```

The following table contains various representation of BPMN data object.

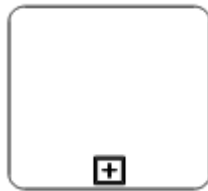
Boundary	Image	
-----	-----	



| Call |



| Event |



| Default |

Datasource

Datasource is used to store or access data associated with a business process. To create a datasource, set the shape as **datasource**. The following code example illustrates how to create a datasource.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule, BpmnDiagramsService } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import { DiagramComponent, Diagram, NodeModel, BpmnShapeModel } from '@syncfusion/ej2-angular-diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [BpmnDiagramsService],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%" height="580px" [getNodeDefaults] ='getNodeDefaults'>
    <e-nodes>
      <e-node id='node1' [offsetX]=150 [offsetY]=150 [shape]='shape'></e-node>
    </e-nodes>
  </ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
```

```
@ViewChild("diagram")
public diagram?: DiagramComponent;
public shape: BpmnShapeModel = {
  type: 'Bpmn',
  shape: 'DataSource'
}
public getNodeDefaults(node: NodeModel): NodeModel {
  node.height = 100;
  node.width = 100;
  return node;
}
}
```

MAIN.TS

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

Artifact

Artifact is used to show additional information about a process in order to make it easier to understand. There are two types of artifacts in BPMN.

- Text annotation
- Group

Text annotation

- A BPMN object can be associated with a text annotation which does not affect the flow but gives details about objects within a flow.
- A TextAnnotation points to or references another BPMN shape, which we call the **textAnnotationTarget** of the textAnnotation. When a target shape is moved or deleted, any TextAnnotations attached to the shape will be moved or deleted too. Thus, the TextAnnotations remain with their target shapes though you can reposition the TextAnnotation to any offset from its target. The **textAnnotationTarget** property of the BpmnTextAnnotation is used to connect an annotation element to the BPMN Node.
- The annotation element can be switched from a BPMN node to another BPMN node simply by dragging the source end of the annotation connector into the other BPMN node.
- By default, the TextAnnotation shape has a connection.
- The **textAnnotationDirection** property is used to set the shape direction of the text annotation.
- By default, the **textAnnotationDirection** is set to a Auto.
- To set the size for text annotation, use the **width** and **height** properties of the node.
- The **offsetX** and **offsetY** properties are used to set the distance between the BPMN node and the TextAnnotation.

APP.COMPONENT.TS

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

```

import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule, BpmnDiagramsService } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import { DiagramComponent, Diagram, NodeModel, BpmnShapeModel } from '@syncfusion/ej2-angular-diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [BpmnDiagramsService],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%"
height="700px" >
  <e-nodes>
    <e-node id='event1' [offsetX]=400 [offsetY]=200 [width]="70"
[height]="70" [shape]='shape'></e-node>
    <e-node id='textNode1' [offsetX]=400 [offsetY]=400 [width]="70"
[height]="70" [shape]='shape1'>
      <e-node-annotations>
        <e-node-annotation content='textNode1'></e-node-
annotation>
      </e-node-annotations>
    </e-node>
    <e-node id='textNode2' [offsetX]=600 [offsetY]=400 [width]="70"
[height]="70" [shape]='shape2'>
      <e-node-annotations>
        <e-node-annotation content='textNode2'></e-node-
annotation>
      </e-node-annotations>
    </e-node>
  </e-nodes>
</ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram?: DiagramComponent;
  public shape: BpmnShapeModel = {
    type: 'Bpmn',
    shape: 'Event',
    event: { event: 'Start', trigger: 'None' },
  }
  //Node with target
  public shape1: BpmnShapeModel = {
    type: 'Bpmn',
    shape: 'TextAnnotation',
    textAnnotation:{
textAnnotationDirection:'Auto',textAnnotationTarget:'event1'
    }
  }
  //Node without target
  public shape2: BpmnShapeModel = {
    type: 'Bpmn',
    shape: 'TextAnnotation',
    textAnnotation:{
textAnnotationDirection:'Auto',textAnnotationTarget:''
  }

```

```
}
}
```

MAIN.TS

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

Group

A group is used to frame a part of the diagram, shows that elements included in it are logically belong together and does not have any other semantics other than organizing elements. To create a group, the shape property of the node should be set as **group**. The following code example illustrates how to create a BPMN group.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule, BpmnDiagramsService } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import { DiagramComponent, Diagram, NodeModel, BpmnShapeModel } from '@syncfusion/ej2-angular-diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [BpmnDiagramsService],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%" height="580px" [getNodeDefaults] = 'getNodeDefaults'>
    <e-nodes>
      <e-node id='node1' [offsetX]=150 [offsetY]=150
[shape]='shape'></e-node>
    </e-nodes>
  </ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram?: DiagramComponent;
  public shape: BpmnShapeModel = {
    type: 'Bpmn',
    shape: 'Group',
  }
  public getNodeDefaults(node: NodeModel): NodeModel {
    node.height = 100;
    node.width = 100;
    return node;
  }
}
```

MAIN.TS

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

BPMN flows

[BPMN Flows](#) are lines that connects BPMN flow objects.

Association

[BPMN Association](#) flow is used to link flow objects with its corresponding text or artifact. An association is represented as a dotted graphical line with opened arrow. The types of association are as follows:

- Directional
- BiDirectional
- Default

The association property allows you to define the type of association. The following code example illustrates how to create an association.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule, BpmnDiagramsService } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, OnInit, ViewChild } from '@angular/core';
import { DiagramComponent, BpmnFlowModel, PointModel } from '@syncfusion/ej2-angular-diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [BpmnDiagramsService],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram id="diagram" width="100%" height="580px">
    <e-connectors>
      <e-connector id='connector' type='Orthogonal'
[sourcePoint]='sourcePoint' [targetPoint]='targetPoint' [shape]='shape'>
    </e-connector>
    </e-connectors>
  </ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild("diagram")
  public sourcePoint?: PointModel;
  public targetPoint?: PointModel;
  public shape?: BpmnFlowModel;
  ngOnInit(): void {
```



```

    this.sourcePoint = { x: 100, y: 100 };
    this.targetPoint = { x: 200, y: 200 };
    this.shape = {
      type: 'Bpmn',
      flow: 'Association',
      association: 'BiDirectional'
    };
  }
}

```

MAIN.TS

```

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

```

The following table demonstrates the visual representation of association flows.

Association	Image
-----	-----
Default	----->
Directional	----->
BiDirectional	-----<----->

The default value for the property `association` is **default**.

Sequence

A [Sequence](#) flow shows the order in which the activities are performed in a BPMN process and is represented by a solid graphical line. The types of sequence are as follows:

- Normal
- Conditional
- Default

The sequence property allows you to define the type of sequence. The following code example illustrates how to create a sequence flow.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { DiagramModule, BpmnDiagramsService } from '@syncfusion/ej2-angular-diagrams';
import { Component, ViewEncapsulation, OnInit, ViewChild } from '@angular/core';
import { DiagramComponent, BpmnFlowModel, PointModel } from '@syncfusion/ej2-angular-diagrams';
@Component({

```

```

imports: [
    DiagramModule
],
providers: [BpmnDiagramsService],
standalone: true,
selector: "app-container",
template: `<ejs-diagram id="diagram" width="100%" height="580px"
[constraints]='diagramConstraints'>
    <e-connectors>
        <e-connector id='connector' type='Orthogonal'
[sourcePoint]='sourcePoint' [targetPoint]='targetPoint'
[constraints]='connectorConstraints' [shape]='shape'>
        </e-connector>
    </e-connectors>
</ejs-diagram>`,
encapsulation: ViewEncapsulation.None
})
export class AppComponent {
    @ViewChild("diagram")
    public sourcePoint?: PointModel;
    public targetPoint?: PointModel;
    public shape?: BpmnFlowModel;
    diagramConstraints: any;
    connectorConstraints: any;
    ngOnInit(): void {
        this.sourcePoint = { x: 100, y: 100 };
        this.targetPoint = { x: 200, y: 200 };
        this.shape = {
            type: 'Bpmn',
            flow: 'Sequence',
            sequence: 'Conditional'
        };
    }
}

```

MAIN.TS

```

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

```

The following table contains various representation of sequence flows.

Sequence Image	
----- -----	
Default	-----
Directional	----->
BiDirectional	◀----->

Note: The default value for the property `sequence` is **normal**.

Message

A [Message](#) flow shows the flow of messages between two participants and is represented by dashed line. The types of message are as follows:

- InitiatingMessage
- NonInitiatingMessage
- Default

The message property allows you to define the type of message. The following code example illustrates how to define a message flow.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule, BpmnDiagramsService } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, OnInit, ViewChild } from '@angular/core';
import { DiagramComponent, Diagram, NodeModel, BpmnFlowModel, DiagramConstraints, ConnectorConstraints, PointModel } from '@syncfusion/ej2-angular-diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [BpmnDiagramsService],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram id="diagram" width="100%" height="580px"
[constraints]='diagramConstraints'>
    <e-connectors>
      <e-connector id='connector' type='Orthogonal'
[sourcePoint]='sourcePoint' [targetPoint]='targetPoint'
[constraints]='connectorConstraints' [shape]='shape'>
      </e-connector>
    </e-connectors>
  </ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild("diagram")
  public diagramConstraints?: DiagramConstraints;
  public connectorConstraints?: ConnectorConstraints;
  public sourcePoint?: PointModel;
  public targetPoint?: PointModel;
  public shape?: BpmnFlowModel;
  ngOnInit(): void {
    this.sourcePoint = { x: 100, y: 100 };
    this.targetPoint = { x: 200, y: 200 };
    this.shape = {
      type: 'Bpmn',
      flow: 'Message',
    }
  }
}
```

```

        message: 'InitiatingMessage'
    };
}
}

```

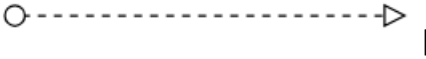


MAIN.TS

```

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

```

The following table contains various representation of message flows.

Message	Image
Default	
InitiatingMessage	
NonInitiatingMessage	

Note: The default value for the property `message` is **default**.

UmdlDiagram in Angular Diagram component

UML Class Diagram

A class diagram visually depicts the static structure of an application and is extensively employed in modeling object-oriented systems. It holds a unique position in UML diagrams, as it directly aligns with object-oriented languages. The diagram also facilitates automatic generation of class diagram shapes based on business logic, streamlining the translation from conceptual models to practical implementation.

UML Class Diagram Shapes

The UML class diagram shapes are explained as follows.

Class

- A class defines a group of objects that share common specifications, features, constraints, and semantics. To create a class object, the classifier should be defined using the [class](#) notation. This notation serves as a foundational element in object-oriented programming, encapsulating the essential characteristics and behavior that objects belonging to the class will exhibit.
- Also, define the [name](#), [attributes](#), and [methods](#) of the class using the class property of node.
- The attribute's [name](#), [type](#), and [scope](#) properties allow you to define the name, data type, and visibility of the attribute.
- The method's [name](#), [parameters](#), [type](#), and [scope](#) properties allow you to define the name, parameter, return type, and visibility of the methods.
- The method parameters object properties allow you to define the name and type of the parameter.

- The following code example illustrates how to create a class.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule, DiagramContextMenuService } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, ViewChild } from "@angular/core";
import { DiagramComponent } from '@syncfusion/ej2-angular-diagrams';
import { NodeModel, UmlClassifierShapeModel } from '@syncfusion/ej2-diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [DiagramContextMenuService],
  standalone: true,
  selector: "app-container",
  // specifies the template string for the diagram component
  template: `<ejs-diagram id="diagram" width="100%" height="580px"
[nodes]='nodes'></ejs-diagram>`
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram?: DiagramComponent;
  public nodes: NodeModel[] = [
    {
      id: "Patient",
      //Position of the node
      offsetX: 200,
      offsetY: 200,
      style: {
        fill: '#26A0DA',
      },
      shape: {
        type: "UmlClassifier",
        //Define class object
        classShape: {
          name: "Patient",
          //Define class attributes
          attributes: [{ name: "accepted", type: "Date" }],
          //Define class methods
          methods: [{ name: "getHistory", type: "getHistory" }]
        },
        classifier: "Class"
      } as UmlClassifierShapeModel
    }
  ];
}
```

MAIN.TS

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

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

Interface

- An interface is a specific type of classifier that signifies a declaration of a cohesive set of public features and obligations. When creating an interface, it involves defining the classifier property using the [interface](#) notation. This essential concept in object-oriented programming outlines a contract for classes to adhere to, specifying the required methods and behaviors without delving into the implementation details.
- Also, define the [name](#), [attributes](#), and [methods](#) of the interface using the interface property of the node.
- The attribute's name, type, and scope properties allow you to define the name, data type, and visibility of the attribute.
- The method's name, parameter, type, and scope properties allow you to define the name, parameter, return type, and visibility of the methods.
- The method parameter object properties of name and type allows you to define the name and type of the parameter.
- The following code example illustrates how to create an interface.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule, DiagramContextMenuService } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, ViewChild } from "@angular/core";
import { DiagramComponent } from '@syncfusion/ej2-angular-diagrams';
import { NodeModel, UmlClassifierShapeModel } from '@syncfusion/ej2-diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [DiagramContextMenuService],
  standalone: true,
  selector: "app-container",
  // specifies the template string for the diagram component
  template: `<ejs-diagram id="diagram" width="100%" height="580px"
[nodes]='nodes'></ejs-diagram>`
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram?: DiagramComponent;
  public nodes: NodeModel[] = [
    {
      id: "Patient",
      //Position of the node
      offsetX: 200,
      offsetY: 200,
      style: {
        fill: '#26A0DA',
      },
      shape: {
```

```

type: "UmlClassifier",
//Define interface object
interfaceShape: {
  name: "Patient",
  //Define interface attributes
  attributes: [{ name: "owner", type: "String[*]" }],
  //Define interface methods
  methods: [
    {
      name: "deposit",
      parameters: [
        {
          name: "amount",
          type: "Dollars"
        }
      ]
    }
  ]
},
classifier: "Interface"
} as UmlClassifierShapeModel
}
];
}

```

MAIN.TS

```

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

```

Enumeration

- To establish an enumeration, designate the classifier property of the node as [enumeration](#). Additionally, define the name and enumerate the members of the enumeration using the appropriate enumeration property of the node. This process encapsulates a set of distinct values within the enumeration, allowing for a clear representation of specific, named constants within a system.
- You can set a name for the enumeration members collection using the name property of members collection.
- The following code example illustrates how to create an enumeration.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule, DiagramContextMenuService } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, ViewChild } from "@angular/core";
import { DiagramComponent } from '@syncfusion/ej2-angular-diagrams';
import { NodeModel, UmlClassifierShapeModel } from '@syncfusion/ej2-diagrams';

```

```

@Component({
  imports: [
    DiagramModule
  ],
  providers: [DiagramContextMenuService],
  standalone: true,
  selector: "app-container",
  // specifies the template string for the diagram component
  template: `<ejs-diagram id="diagram" width="100%" height="580px"
[nodes]='nodes'></ejs-diagram>`
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram?: DiagramComponent;
  public nodes: NodeModel[] = [
    {
      id: "Patient",
      offsetX: 200,
      offsetY: 200,
      style: {
        fill: '#26A0DA',
      },
      shape: {
        type: "UmlClassifier",
        //Define enumeration object
        enumerationShape: {
          name: "AccountType",
          //set the members of enumeration
          members: [
            {
              name: "Checking Account",
            },
            {
              name: "Savings Account"
            },
            {
              name: "Credit Account"
            }
          ]
        },
        classifier: "Enumeration"
      } as UmlClassifierShapeModel
    }
  ];
}

```

MAIN.TS

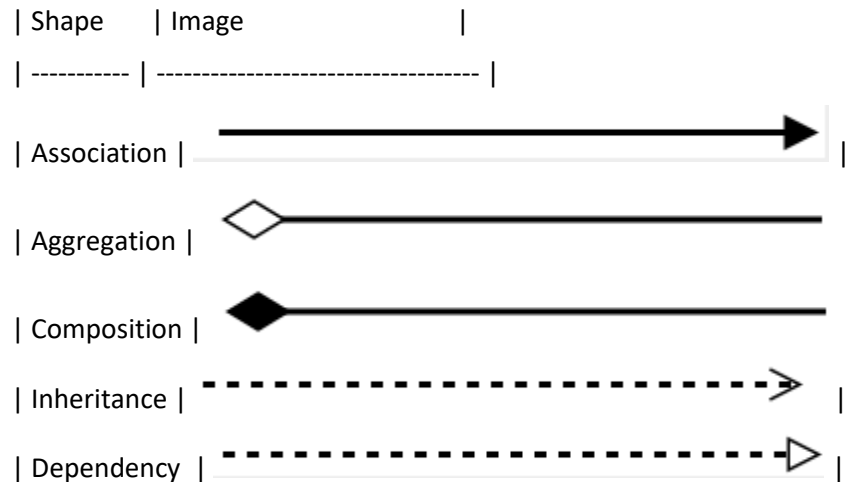
```

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

```


UML Class Relationships

- A class may be involved in one or more relationships with other classes. A relationship can be one of the following types:



Association

Association is basically a set of links that connects elements of an UML model. The type of association is as follows.

1. Directional
2. BiDirectional

The association property allows you to define the type of association. The default value of association is "Directional". The following code example illustrates how to create an association.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule, DiagramContextMenuService } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, ViewChild } from "@angular/core";
import { DiagramComponent } from '@syncfusion/ej2-angular-diagrams';
import { ConnectorModel } from '@syncfusion/ej2-diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [DiagramContextMenuService],
  standalone: true,
  selector: "app-container",
  // specifies the template string for the diagram component
  template: `<ejs-diagram id="diagram" width="100%" height="580px"
[connectors]='connectors'></ejs-diagram>`
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram?: DiagramComponent;
```

```

public connectors: ConnectorModel[] = [
  {
    id: "connector",
    //Define connector start and end points
    sourcePoint: { x: 100, y: 100 },
    targetPoint: { x: 300, y: 300 },
    type: "Straight",
    shape: {
      type: "UmlClassifier",
      relationship: "Association",
      //Define type of association
      association: "BiDirectional"
    }
  }
];
}

```

MAIN.TS

```

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

```

Aggregation

Aggregation is a binary association between a property and one or more composite objects which group together a set of instances.

Aggregation is decorated with a hollow diamond. To create an aggregation shape, define the relationship as “aggregation”.

The following code example illustrates how to create an aggregation.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule, DiagramContextMenuService } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import { DiagramComponent } from '@syncfusion/ej2-angular-diagrams';
import { ConnectorModel } from '@syncfusion/ej2-diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [DiagramContextMenuService],
  standalone: true,
  selector: "app-container",
  // specifies the template string for the diagram component
  template: `<ejs-diagram id="diagram" width="100%" height="580px"
[connectors]='connectors'></ejs-diagram>`
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram?: DiagramComponent;
}

```

```

    public connectors: ConnectorModel[] = [
    {
        id: "connector",
        //Define connector start and end points
        sourcePoint: { x: 100, y: 100 },
        targetPoint: { x: 300, y: 300 },
        type: "Straight",
        shape: {
            type: "UmlClassifier",
            //Set an relationship for connector
            relationship: "Aggregation"
        }
    }
    ];
}

```

MAIN.TS

```

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

```

Composition

Composition is a “strong” form of “aggregation”. Composition is decorated with a black diamond. To create a composition shape, define the relationship property of connector as “composition”.

The following code example illustrates how to create a composition.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule, DiagramContextMenuService } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, ViewChild } from "@angular/core";
import { DiagramComponent } from '@syncfusion/ej2-angular-diagrams';
import { ConnectorModel } from '@syncfusion/ej2-diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [DiagramContextMenuService],
  standalone: true,
  selector: "app-container",
  // specifies the template string for the diagram component
  template: `<ejs-diagram id="diagram" width="100%" height="580px"
[connectors]='connectors'></ejs-diagram>`
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram?: DiagramComponent;
  public connectors: ConnectorModel[] = [
    {
      id: "connector",
      //Define connector start and end points

```

```

    sourcePoint: { x: 100, y: 100 },
    targetPoint: { x: 300, y: 300 },
    type: "Straight",
    shape: {
      type: "UmlClassifier",
      //Set an relationship for connector
      relationship: "Composition"
    }
  }
];
}

```

MAIN.TS

```

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

```

Dependency

Dependency is a directed relationship, which is used to show that some UML elements needs or depends on other model elements for specifications. Dependency is shown as dashed line with opened arrow. To create a dependency, define the relationship property of connector as “dependency”.

The following code example illustrates how to create an dependency.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule, DiagramContextMenuService } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, ViewChild } from "@angular/core";
import { DiagramComponent } from '@syncfusion/ej2-angular-diagrams';
import { ConnectorModel } from '@syncfusion/ej2-diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [DiagramContextMenuService],
  standalone: true,
  selector: "app-container",
  // specifies the template string for the diagram component
  template: `<ejs-diagram id="diagram" width="100%" height="580px"
[connectors]='connectors'></ejs-diagram>`
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram?: DiagramComponent;
  public connectors: ConnectorModel[] = [
    {
      id: "connector",
      //Define connector start and end points
      sourcePoint: { x: 100, y: 100 },
      targetPoint: { x: 300, y: 300 },
      type: "Straight",

```

```

    shape: {
      type: "UmlClassifier",
      //Set relationship for connector
      relationship: "Dependency"
    }
  }
];
}

```

MAIN.TS

```

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

```

Inheritance

Inheritance is also called as “generalization”. Inheritance is a binary taxonomic directed relationship between a more general classifier (super class) and a more specific classifier (subclass). Inheritance is shown as a line with hollow triangle.

To create an inheritance, define the relationship as “inheritance”.

The following code example illustrates how to create an inheritance.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { DiagramModule, DiagramContextMenuService } from '@syncfusion/ej2-angular-diagrams';
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import { DiagramComponent } from '@syncfusion/ej2-angular-diagrams';
import { ConnectorModel } from '@syncfusion/ej2-diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [DiagramContextMenuService],
  standalone: true,
  selector: "app-container",
  // specifies the template string for the diagram component
  template: `<ejs-diagram id="diagram" width="100%" height="580px"
[connectors]='connectors'></ejs-diagram>`
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram?: DiagramComponent;
  public connectors: ConnectorModel[] = [
    {
      id: "connector",
      //Define connector start and end points
      sourcePoint: { x: 100, y: 100 },
      targetPoint: { x: 300, y: 300 },
      type: "Straight",
      shape: {

```

```

    type: "UmlClassifier",
    //set an relation of connector
    relationship: "Inheritance"
  }
}
];
}

```

MAIN.TS

```

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

```

Multiplicity

Multiplicity is a definition of an inclusive interval of non-negative integers to specify the allowable number of instances of described element. The type of multiplicity are as follows.

- OneToOne
- ManyToOne
- OneToMany
- ManyToMany 1.By default the multiplicity will be considered as “OneToOne”. 2.The multiplicity property in UML allows you to specify large number of elements or some collection of elements. 3.The shape multiplicity’s source property is used to set the source label to connector and the target property is used to set the target label to connector. 4.To set an optionality or cardinality for the connector source label, use optional property. 5.The [lowerBounds](#) and [upperBounds](#) could be natural constants or constant expressions evaluated to natural (non negative) number. Upper bound could be also specified as asterisk ‘*’ which denotes unlimited number of elements. Upper bound should be greater than or equal to the lower bound.

The following code example illustrates how to customize the multiplicity.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule, DiagramContextMenuService } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, ViewChild } from "@angular/core";
import { DiagramComponent } from '@syncfusion/ej2-angular-diagrams';
import { ConnectorModel } from '@syncfusion/ej2-diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [DiagramContextMenuService],
  standalone: true,
  selector: "app-container",
  // specifies the template string for the diagram component
  template: `<ejs-diagram id="diagram" width="100%" height="580px"
[connectors]='connectors'></ejs-diagram>`
})

```

```

export class AppComponent {
  @ViewChild("diagram")
  public diagram?: DiagramComponent;
  public connectors: ConnectorModel[] = [
    {
      id: "connector",
      //Define connector start and end points
      sourcePoint: { x: 100, y: 100 },
      targetPoint: { x: 300, y: 300 },
      type: "Straight",
      shape: {
        type: "UmlClassifier",
        relationship: "Dependency",
        multiplicity: {
          //Set multiplicity type
          type: "OneToMany",
          //Set source label to connector
          source: {
            optional: true,
            lowerBounds: '89',
            upperBounds: '67'
          },
          //Set target label to connector
          target: {
            optional: true,
            lowerBounds: '78',
            upperBounds: '90'
          }
        }
      }
    }
  ];
}

```

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 UML child at runtime

In UML nodes, child elements such as member, method and attribute can be added either programmatically or interactively.

Adding UML child through code

The [addChildToUmlNode](#) method is employed for dynamically adding a child to the UML node during runtime, providing flexibility in modifying the diagram structure programmatically.

The following code illustrates how to add methods to UML nodes in diagram.

```
`ts
```

```
let node = diagram.selectedItems.nodes[0];
```

```
let methods = { name: 'getHistory', style: { color: "red", }, parameters: [{ name: 'Date', style: {} }], type: 'History' };
```

```
diagram.addChildToUmlNode(node, methods, 'Method');
```

```
,
```

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule, DiagramContextMenuService } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, ViewChild } from "@angular/core";
import { DiagramComponent } from '@syncfusion/ej2-angular-diagrams';
import { NodeModel, UmlClassifierShapeModel } from '@syncfusion/ej2-diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [DiagramContextMenuService],
  standalone: true,
  selector: "app-container",
  // specifies the template string for the diagram component
  template: `
    <button (click)="addMethod()">addMethod</button>
    <ejs-diagram #diagram id="diagram" width="100%" height="600"
[nodes]="nodes" >
    </ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild('diagram')
  public diagram?: DiagramComponent | undefined;
  addMethod() {
    let node: any = (this.diagram as any).nodes[0];
    let methods: any = {
      name: 'getHistory',
      style: { color: 'red' },
      parameters: [{ name: 'Date', style: {} }],
      type: 'History',
    };
    (this.diagram as any).addChildToUmlNode(node, methods, 'Method');
  }
  public nodes: NodeModel[] = [
    {
      id: 'node1',
      offsetX: 150,
      offsetY: 150,
      style: {
        fill: '#26A0DA',
      },
      shape: {
        type: 'UmlClassifier',
        classShape: {
          attributes: [{ name: 'accepted', type: 'Date' }],
          methods: [
```



```

        {
            name: 'getHistory',
            style: {},
            parameters: [{ name: 'Date', style: {} }],
            type: 'History',
        },
    ],
    name: 'Patient',
},
classifier: 'Class',
},
},
];
}

```

MAIN.TS

```

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

```

The following code illustrates how to add attributes to UML nodes in diagram.

```
`ts
```

```
let node = diagram.selectedItems.nodes[0];
```

```
let attributes = { name: 'accepted', type: 'Date', style: { color: "red", } };
```

```
diagram.addChildToUmlNode(node, attributes, "Attribute");
```

```
,
```

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule, DiagramContextMenuService } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, ViewChild } from "@angular/core";
import { DiagramComponent } from '@syncfusion/ej2-angular-diagrams';
import { NodeModel, UmlClassifierShapeModel } from '@syncfusion/ej2-diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [DiagramContextMenuService],
  standalone: true,
  selector: "app-container",
  // specifies the template string for the diagram component
  template: `
    <button (click)="addAttribute()">addAttribute</button>
    <ejs-diagram #diagram id="diagram" width="100%" height="600"
[nodes]="nodes" >
    </ejs-diagram>`,

```

```

        encapsulation: ViewEncapsulation.None
    })
    export class AppComponent {
        @ViewChild('diagram')
        public diagram?: DiagramComponent | undefined;
        addAttribute() {
            let node: any = (this.diagram as any).nodes[0];
            let attributes: any = { name: 'accepted', type: 'Date', style: { color:
"red", } };
            (this.diagram as any).addChildToUmlNode(node, attributes, "Attribute");
        }
        public nodes: NodeModel[] = [
            {
                id: 'node1',
                offsetX: 150,
                offsetY: 150,
                style: {
                    fill: '#26A0DA',
                },
                shape: {
                    type: 'UmlClassifier',
                    classShape: {
                        attributes: [{ name: 'accepted', type: 'Date' }],
                        methods: [
                            {
                                name: 'getHistory',
                                style: {},
                                parameters: [{ name: 'Date', style: {} }],
                                type: 'History',
                            },
                        ],
                        name: 'Patient',
                    },
                    classifier: 'Class',
                },
            },
        ];
    }
}

```

MAIN.TS

```

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

```

The following code illustrates how to add members to UML nodes in diagram.

```
`ts
```

```
let node = diagram.selectedItems.nodes[0];
```

```
let members = { name: 'Checking new', style: { color: 'red', }, isSeparator: true };
```

```
diagram.addChildToUmlNode(node, members, "Member");
```

```
`
```

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule, DiagramContextMenuService } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, ViewChild } from "@angular/core";
import { DiagramComponent } from '@syncfusion/ej2-angular-diagrams';
import { NodeModel, UmlClassifierShapeModel } from '@syncfusion/ej2-diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [DiagramContextMenuService],
  standalone: true,
  selector: "app-container",
  // specifies the template string for the diagram component
  template: `
    <button (click)="addMember()">addMember</button>
    <ejs-diagram #diagram id="diagram" width="100%" height="600"
[nodes]="nodes" >
    </ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild('diagram')
  public diagram?: DiagramComponent | undefined;
  addMember() {
    let node: any = (this.diagram as any).nodes[0];
    let members:any = { name: 'Checking new', style: { color: "red", } };
    (this.diagram as any).addChildToUmlNode(node, members, "Member");
  }
  public nodes: NodeModel[] = [
    {
      id: 'node1',
      offsetX: 150,
      offsetY: 150, style: {
        fill: '#26A0DA',
      }, borderColor: 'white',
      shape: {
        type: 'UmlClassifier',
        enumerationShape: {
          name: 'AccountType',
          members: [
            {
              name: 'Checking Account',
            },
          ],
        },
        classifier: 'Enumeration'
      },
    },
  ];
}

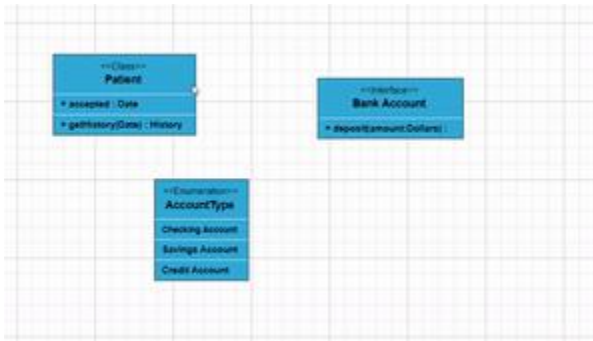
```

MAIN.TS

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

Adding UML child through user interaction

To include a child, select a node, move the mouse outside it, and position the pointer near the right side. A highlighter emerges between the two child elements. Click the highlighter to add a child type to the chosen UML node seamlessly. The following gif illustrates how to add Child through user interaction.

*Adding UML Nodes in Symbol palette*

UML built-in shapes are efficiently rendered in a symbol palette. The `symbols` property is utilized to define UML symbols with the necessary classes and methods. By incorporating this feature, you can seamlessly augment the palette with a curated collection of predefined UML symbols, thereby enhancing the versatility of your UML diagramming application.

The following code example showcases the rendering of UML built-in shapes in a symbol palette

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule, SymbolPaletteModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, OnInit, ViewEncapsulation, ViewChild } from '@angular/core';
import { SymbolPaletteComponent, SymbolPalette, NodeModel, MarginModel, PaletteModel, SymbolInfo, DiagramComponent } from '@syncfusion/ej2-angular-diagrams';
@Component({
  imports: [
    DiagramModule, SymbolPaletteModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  template: `<div style="width: 100%">
    <div id="palette-space" class="sb-mobile-palette">
      <ejs-symbolpalette
        id="symbolpalette"
        width="100%"
        height="700px">
```

```

        [symbolHeight]="80"
        [symbolWidth]="80"
        [palettes]="palettes"
        [getSymbolInfo]="getSymbolInfo"
        [symbolMargin]="symbolMargin"
        [getNodeDefaults]="getSymbolDefaults"
      >
    </ejs-symbolpalette>
  </div>
  <div id="diagram-space" class="sb-mobile-diagram">
    <div class="content-wrapper">
      <ejs-diagram #diagram id="diagram" width="100%" height="700px">
    </ejs-diagram>
    </div>
  </div>
</div>
`
    encapsulation: ViewEncapsulation.None
  })
export class AppComponent {
  public palettes?: PaletteModel[];
  public diagram?: DiagramComponent;
  public symbolMargin?: MarginModel;
  public getUmlShapes(): NodeModel[] {
    let umlShapes: NodeModel[] = [
      {
        id: 'class',
        style: {
          fill: '#26A0DA',
        },
        borderColor: 'white',
        shape: {
          type: 'UmlClassifier',
          classShape: {
            attributes: [
              { name: 'accepted', type: 'Date', style: {
color: "red", fontFamily: "Arial", textDecoration: 'Underline', italic:
true },isSeparator: true },
            ],
            methods: [{ name: 'getHistory', style: {}},
parameters: [{ name: 'Date', style: {} }], type: 'History' }],
              name: 'Patient'
            },
            classifier: 'Class'
          },
        },
      {
        id: 'Interface',
        style: {
          fill: '#26A0DA',
        }, borderColor: 'white',
        shape: {
          type: 'UmlClassifier',
          interfaceShape: {
            name: "Bank Account",
          },
          classifier: 'Interface'
        }
      }
    ];
  }
}

```

```

        },
        {
            id: 'Enumeration',
            style: {
                fill: '#26A0DA',
            }, borderColor: 'white',
            shape: {
                type: 'UmlClassifier',
                enumerationShape: {
                    name: 'AccountType',
                    members: [
                        {
                            name: 'Checking Account', style: {}
                        },
                    ],
                },
                classifier: 'Enumeration'
            },
        },
    ];
    return umlShapes;
};

public getSymbolInfo(symbol: NodeModel): SymbolInfo {
    return { fit: true, description: { text: symbol.id, } };
};

public getSymbolDefaults(symbol: NodeModel): void {
    symbol.width = 100;
    symbol.height = 100;
};

ngOnInit(): void {
    this.palettes = [{
        id: 'uml',
        expanded: true,
        symbols: this.getUmlShapes(),
        title: 'UML Shapes',
    }]
    //Sets the space to be left around a symbol
    this.symbolMargin = {
        left: 12, right: 12, top: 12, bottom: 12
    }
}
}

```

MAIN.TS

```

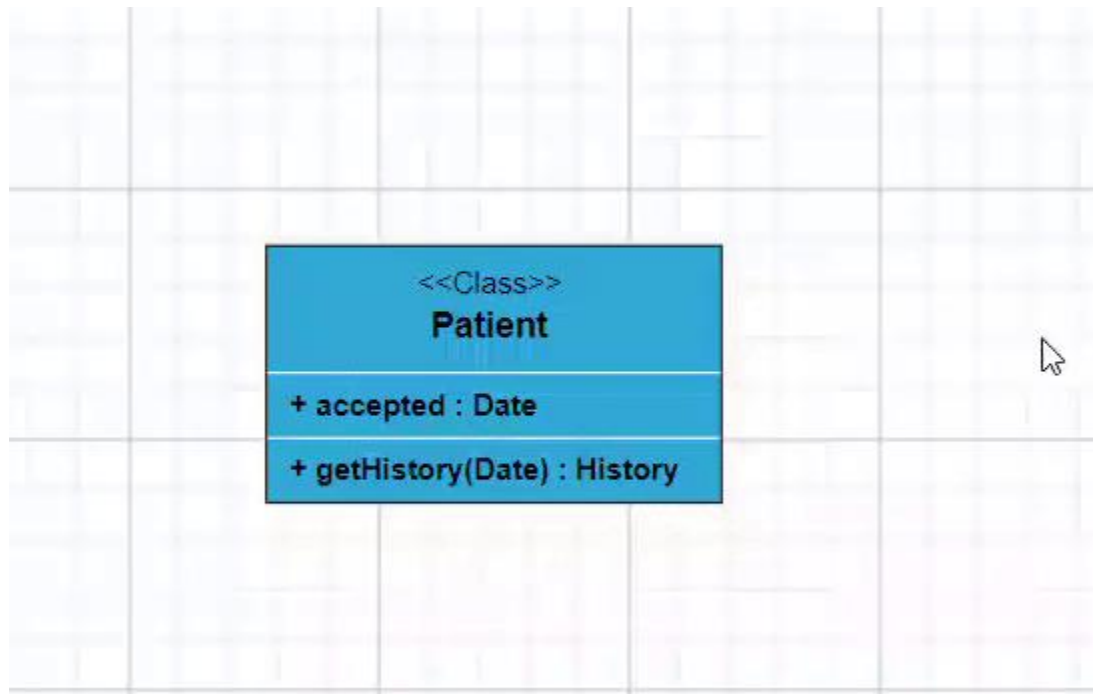
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Editing

You can edit the name, attributes, and methods of the class diagram shapes just double clicking, similar to editing a node annotation.

The following image illustrates how the text editor looks in an edit mode.



UML Activity diagram

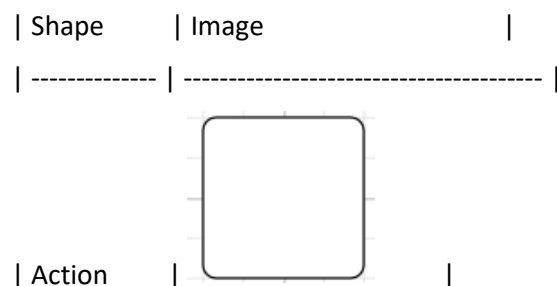
An Activity diagram functions as a visual flowchart, illustrating the progression from one activity to the next within a system. Each activity corresponds to a system operation, providing a clear depiction of the sequential flow in a dynamic process..

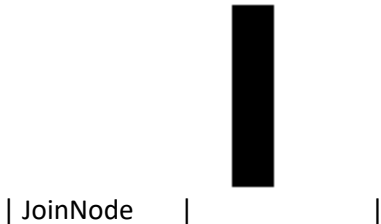
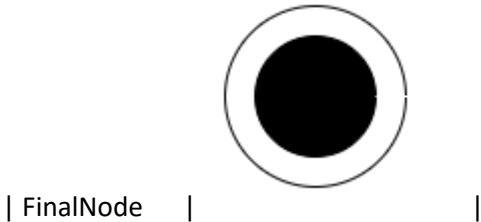
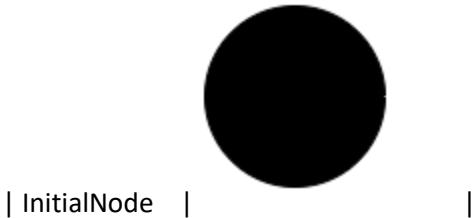
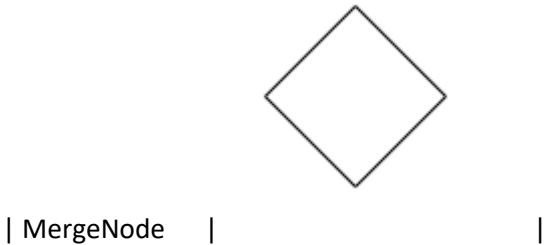
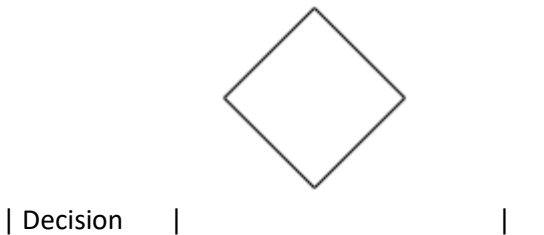
The purpose of an activity diagram can be described as follows.

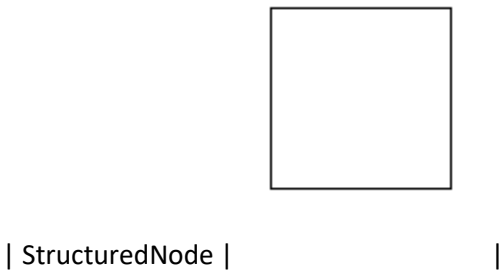
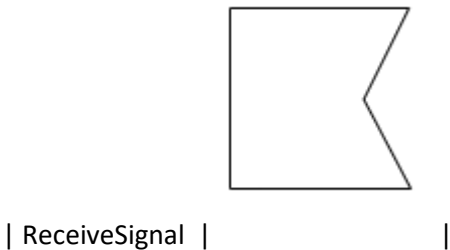
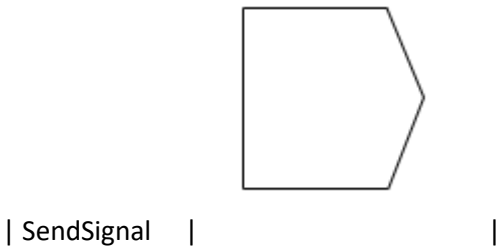
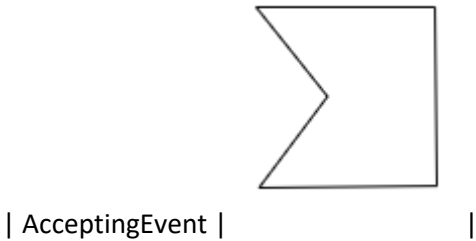
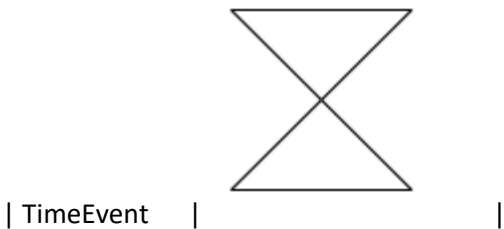
1. Draw the activity flow of a system.
2. Describe the sequence from one activity to another.
3. Describe the parallel, branched, and concurrent flow of the system.

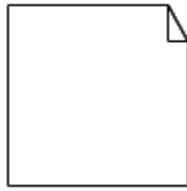
UML Activity diagram Shapes

To create a UmlActivity, define type as "UmlActivity" and the list of built-in shapes as demonstrated as follows and it should be set in the "shape" property.









| Note

The following code illustrates how to create a UmlActivity shapes.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule, DiagramContextMenuService } from '@syncfusion/ej2-
angular-diagrams'
import { Component, ViewEncapsulation, ViewChild } from "@angular/core";
import { DiagramComponent } from '@syncfusion/ej2-angular-diagrams';
import { NodeModel, UmlClassifierShapeModel } from '@syncfusion/ej2-
diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [DiagramContextMenuService],
  standalone: true,
  selector: "app-container",
  // specifies the template string for the diagram component
  template: `<ejs-diagram id="diagram" width="100%" height="580px"
[nodes]='nodes'></ejs-diagram>`
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram?: DiagramComponent;
  public nodes: NodeModel[] = [
    {
      id: "UmlDiagram",
      //Set node size
      width: 100,
      height: 100,
      //position the node
      offsetX: 200,
      offsetY: 200,
      shape: {
        type: "UmlActivity",
        //Define UmlActivity shape
        shape: "Action"
      }
    }
  ];
}
```

MAIN.TS

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

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

Uml Activity connector

To create an Uml Activity connector, define the type as "UmlActivity" and flow as either "Exception" or "Control" or "Object".

The following code illustrates how to create a UmlActivity connector.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule, DiagramContextMenuService } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, ViewChild } from "@angular/core";
import { BpmnFlows, DiagramComponent } from '@syncfusion/ej2-angular-diagrams';
import { ConnectorModel } from '@syncfusion/ej2-diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [DiagramContextMenuService],
  standalone: true,
  selector: "app-container",
  // specifies the template string for the diagram component
  template: `<ejs-diagram id="diagram" width="100%" height="580px"
[connectors]='connectors'></ejs-diagram>`
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram?: DiagramComponent;
  public connectors: ConnectorModel[] = [
    {
      id: 'connector',
      type: 'Straight',
      //Define connector start and end points
      sourcePoint: { x: 100, y: 100 },
      targetPoint: { x: 200, y: 200 },
      shape: { type: 'UmlActivity', flow: 'Exception' as BpmnFlows }
    }
  ];
}
```

MAIN.TS

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

Connectors in Angular Diagram component

Connectors are objects used to create link between two points, nodes or ports to represent the relationships between them.

Create connector

Connector can be created by defining the source and target point of the connector. The path to be drawn can be defined with a collection of segments. To explore the properties of a [connector](#), refer to [Link to the Video](#).

To create and customize the connectors easily in the Angular Diagram component, refer to the below video link.

Add connectors through connectors collection

The [sourcePoint](#) and [targetPoint](#) properties of connector allow you to define the end points of a connector.

The following code example illustrates how to add a connector through connector collection.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, OnInit, ViewChild } from '@angular/core';
import { DiagramComponent, Diagram, NodeModel, ConnectorModel, PointModel } from '@syncfusion/ej2-angular-diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%" height="580px" [getConnectorDefaults] = 'getConnectorDefaults'>
    <e-connectors>
      <e-connector id='connector' type='Straight' [sourcePoint]='sourcePoint' [targetPoint]='targetPoint'>
    </e-connector>
    </e-connectors>
  </ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram?: DiagramComponent;
  public sourcePoint?: PointModel;
  public targetPoint?: PointModel;
  ngOnInit(): void {
    this.sourcePoint = { x: 100, y: 100 };
    this.targetPoint = { x: 200, y: 200 };
  }
  public getConnectorDefaults(obj: ConnectorModel): void {
    obj.style = {
      strokeColor: '#6BA5D7',
    }
  }
}
```

```

        fill: '#6BA5D7',
        strokeWidth: 2
    }
    obj.targetDecorator = {
        style: {
            fill: '#6BA5D7',
            strokeColor: '#6BA5D7'
        }
    }
}
}
}

```

MAIN.TS

```

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

```

Add connector at runtime

Connectors can be added at runtime by using public method, `diagram.add` and can be removed at runtime by using public method, `diagram.remove`.

The following code example illustrates how to add connector at runtime.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, OnInit, ViewChild } from '@angular/core';
import { DiagramComponent, Diagram, NodeModel, ConnectorModel } from '@syncfusion/ej2-angular-diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%"
height="580px" [getConnectorDefaults] ='getConnectorDefaults'
(created)='created($event)'>
</ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram?: DiagramComponent;
  public connectors: ConnectorModel[] = [{
    id: "connector1",
    style: {
      strokeColor: '#6BA5D7',
      fill: '#6BA5D7',

```

```

        strokeWidth: 2
    },
    targetDecorator: {
        style: {
            fill: '#6BA5D7',
            strokeColor: '#6BA5D7'
        }
    },
    sourcePoint: {
        x: 100,
        y: 100
    },
    targetPoint: {
        x: 200,
        y: 200
    }
}
]] as ConnectorModel[];
public getConnectorDefaults(obj: ConnectorModel): void {
    obj.style = {
        strokeColor: '#6BA5D7',
        fill: '#6BA5D7',
        strokeWidth: 2
    }
    obj.targetDecorator = {
        style: {
            fill: '#6BA5D7',
            strokeColor: '#6BA5D7'
        }
    }
}
public created(args: Object): void {
    // Adds to the Diagram
    (this.diagram as DiagramComponent).add(this.connectors as any);
    // Remove from the diagram
    (this.diagram as DiagramComponent).remove(this.connectors as any);
}
}

```

MAIN.TS

```

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

```

Connectors from palette

Connectors can be predefined and added to the symbol palette. You can drop those connectors into the diagram, when required.

For more information about adding connectors from symbol palette, refer to [Symbol Palette](#).

Draw connectors

Connectors can be interactively drawn by clicking and dragging on the diagram surface by using [drawingObject](#).

For more information about drawing connectors, refer to [Draw Connectors](#).

Update connector at runtime

Various connector properties such as `sourcePoint`, `targetPoint`, `style`, `sourcePortID`, `targetPortID`, etc., can be updated at the runtime.

The following code example illustrates how to update a connector's source point, target point, styles properties at runtime.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, OnInit, ViewChild } from
 '@angular/core';
import { DiagramComponent, Diagram, NodeModel, ConnectorModel, PointModel,
 StrokeStyleModel, DecoratorModel, ShapeStyleModel } from '@syncfusion/ej2-
angular-diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%"
height="580px" [getConnectorDefaults] = 'getConnectorDefaults'
(created)='created($event)'>
    <e-connectors>
      <e-connector id='connector' type='Straight'
[sourcePoint]='sourcePoint' [targetPoint]='targetPoint'>
      </e-connector>
    </e-connectors>
  </ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram?: DiagramComponent;
  public sourcePoint?: PointModel;
  public targetPoint?: PointModel;
  ngOnInit(): void {
    this.sourcePoint = { x: 100, y: 100 };
    this.targetPoint = { x: 200, y: 200 };
  }
  public getConnectorDefaults(obj: ConnectorModel): void {
    obj.style = {
      strokeColor: '#6BA5D7',
      fill: '#6BA5D7',
      strokeWidth: 2
    }
    obj.targetDecorator = {
      style: {
        fill: '#6BA5D7',
        strokeColor: '#6BA5D7'
      }
    }
  }
}
```

```

    }
  }
  public created(args: Object): void {
    // Update the connector properties at the run time
    ((this.diagram as DiagramComponent).connectors[0].style as
    StrokeStyleModel).strokeColor = '#6BA5D7';
    ((this.diagram as DiagramComponent).connectors[0].style as
    StrokeStyleModel).fill = '#6BA5D7';
    ((this.diagram as DiagramComponent).connectors[0].style as
    StrokeStyleModel).strokeWidth = 2;
    (((this.diagram as DiagramComponent).connectors[0].targetDecorator
    as DecoratorModel).style as StrokeStyleModel).fill = '#6BA5D7';
    (((this.diagram as DiagramComponent).connectors[0].targetDecorator
    as DecoratorModel).style as ShapeStyleModel).strokeColor = '#6BA5D7';
    ((this.diagram as DiagramComponent).connectors[0].sourcePoint as
    PointModel).x = 150;
    ((this.diagram as DiagramComponent).connectors[0].targetPoint as
    PointModel).x = 150;
    (this.diagram as DiagramComponent).dataBind();
  }
}

```

MAIN.TS

```

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

```

Connect nodes

- The [sourceID](#) and [targetID](#) properties allow to define the nodes to be connected.
- The [connectorSpacing](#) property allows you to define the distance between the source node and the connector. It is the minimum distance the connector will re-route or the new segment will create.

The following code example illustrates how to connect two nodes.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import { DiagramComponent, Diagram, NodeModel, ConnectorModel,
ShapeStyleModel } from '@syncfusion/ej2-angular-diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",

```



```

    template: `<ejs-diagram #diagram id="diagram" width="100%"
height="580px" [getNodeDefaults] ='getNodeDefaults' [getConnectorDefaults]
='getConnectorDefaults'>
    <e-nodes>
        <e-node id='node1' [offsetX]=150 [offsetY]=150>
            <e-node-annotations>
                <e-node-annotation content="Node1">
                </e-node-annotation>
            </e-node-annotations>
        </e-node>
        <e-node id='node2' [offsetX]=350 [offsetY]=150>
            <e-node-annotations>
                <e-node-annotation content="Custom Template">
                </e-node-annotation>
            </e-node-annotations>
        </e-node>
    </e-nodes>
    <e-connectors>
        <e-connector id='connector' type='Orthogonal' sourceID='node1'
targetID='node2' connectorSpacing='connectorSpacing'>
        </e-connector>
    </e-connectors>
</ejs-diagram>`,
    encapsulation: ViewEncapsulation.None
})
export class AppComponent {
    @ViewChild("diagram")
    public diagram?: DiagramComponent;
    public connectorSpacing?: number;
    ngOnInit(): void {
        this.connectorSpacing = 7;
    }
    public getNodeDefaults(node: NodeModel | any): NodeModel {
        node.height = 100;
        node.width = 100;
        ((node as NodeModel).style as ShapeStyleModel).fill = "#6BA5D7";
        ((node as NodeModel).style as ShapeStyleModel).strokeColor =
"White";
        return node;
    }
    public getConnectorDefaults(obj: ConnectorModel): void {
        obj.style = {
            strokeColor: '#6BA5D7',
            fill: '#6BA5D7',
            strokeWidth: 2
        }
        obj.targetDecorator = {
            style: {
                fill: '#6BA5D7',
                strokeColor: '#6BA5D7'
            }
        }
    }
}

```

MAIN.TS

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

- When you remove NodeConstraints [InConnect](#) from Default, the node accepts only an outgoing connection to dock in it. Similarly, when you remove NodeConstraints [OutConnect](#) from Default, the node accepts only an incoming connection to dock in it.
- When you remove both InConnect and OutConnect NodeConstraints from Default, the node restricts connector to establish connection in it.

The following code illustrates how to disable InConnect constraints.

```
`typescript
```

```
import { Component, ViewEncapsulation, ViewChild } from "@angular/core";
import { NodeConstraints } from "@syncfusion/ej2-angular-diagrams";

@Component({
  selector: "app-container",
  template: `<ejs-diagram id="diagram" width="100%" height="580px">
    <e-nodes>
    <e-node id='node1' [height]=60 [width]=100 [offsetX]=300 [offsetY]=80 [constraints]='nodeConstraints'>
    </e-node>
    </e-nodes>
  </ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild("diagram")
  public nodeConstraints: NodeConstraints;
  ngOnInit(): void {
    this.nodeConstraints = NodeConstraints.Default & ~NodeConstraints.InConnect;
  }
}
```

Connections with ports

The [sourcePortID](#) and [targetPortID](#) properties allow to create connections between some specific points of source/target nodes.

The following code example illustrates how to create port to port connections.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import { DiagramComponent, Diagram, NodeModel, ConnectorModel,
PointPortModel, PortVisibility, ShapeStyleModel } from '@syncfusion/ej2-
angular-diagrams';
@Component({
imports: [
    DiagramModule
],
providers: [ ],
standalone: true,
    selector: "app-container",
    template: `<ejs-diagram #diagram id="diagram" width="100%"
height="580px" [getNodeDefaults] ='getNodeDefaults'>
    <e-nodes>
        <e-node id='node1' [offsetX]=150 [offsetY]=150 [ports]='port1'>
            <e-node-annotations>
                <e-node-annotation content="Node1">
                    </e-node-annotation>
                </e-node-annotations>
            </e-node>
            <e-node id='node2' [offsetX]=350 [offsetY]=150 [ports]='port2'>
                <e-node-annotations>
                    <e-node-annotation content="Custom Template">
                        </e-node-annotation>
                    </e-node-annotations>
                </e-node>
            </e-nodes>
            <e-connectors>
                <e-connector id='connector' type='Straight' sourceID='node1'
sourcePortID='port1' targetID='node2' targetPortID='port2'>
                    </e-connector>
                </e-connectors>
            </ejs-diagram>`,
    encapsulation: ViewEncapsulation.None
})
export class AppComponent {
    @ViewChild("diagram")
    public diagram?: DiagramComponent;
    public port1: PointPortModel[] = [{
        id: 'port1',
        offset: {
            x: 0,
            y: 0.5
        },
        visibility: PortVisibility.Visible
    }]
    public port2: PointPortModel[] = [
        {
            id: 'port2',
            offset: {
                x: 1,
                y: 0.5
            }
        }
    ]
}

```

```

        },
        visibility: PortVisibility.Visible
    },
    {
        id: 'port3',
        offset: {
            x: 0.5,
            y: 0
        },
        visibility: PortVisibility.Visible
    }
]
public getNodeDefaults(node: NodeModel | any): NodeModel {
    node.height = 100;
    node.width = 100;
    ((node as NodeModel).style as ShapeStyleModel).fill = "#6BA5D7";
    ((node as NodeModel).style as ShapeStyleModel).strokeColor =
    "White";
    return node;
}
public getConnectorDefaults(obj: ConnectorModel): void {
    obj.style = {
        strokeColor: '#6BA5D7',
        fill: '#6BA5D7',
        strokeWidth: 2
    }
    obj.targetDecorator = {
        style: {
            fill: '#6BA5D7',
            strokeColor: '#6BA5D7'
        }
    }
}
}
}

```

MAIN.TS

```

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

```

Similarly, the `sourcePortID` or `targetPortID` can be changed at the runtime by changing the port [sourcePortID](#) or [targetPortID](#).

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import { DiagramComponent, Diagram, NodeModel, ConnectorModel,
PointPortModel ,PortVisibility, ShapeStyleModel} from '@syncfusion/ej2-
angular-diagrams';
@Component({
imports: [

```

```

    DiagramModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%"
height="580px" [getNodeDefaults] ='getNodeDefaults'
(created)='created($event)'>
    <e-nodes>
      <e-node id='node1' [offsetX]=150 [offsetY]=150 [ports]='port1'>
        <e-node-annotations>
          <e-node-annotation content="Node1">
            </e-node-annotation>
          </e-node-annotations>
        </e-node>
      <e-node id='node2' [offsetX]=350 [offsetY]=150 [ports]='port2'>
        <e-node-annotations>
          <e-node-annotation content="Custom Template">
            </e-node-annotation>
          </e-node-annotations>
        </e-node>
      </e-nodes>
    <e-connectors>
      <e-connector id='connector' type='Straight' sourceID='node1'
sourcePortID='port1' targetID='node2' targetPortID='port2'>
      </e-connector>
    </e-connectors>
  </ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram?: DiagramComponent;
  public port1: PointPortModel[] = [{
    id: 'port1',
    offset: {
      x: 0,
      y: 0.5
    },
    visibility: PortVisibility.Visible
  }]
  public port2: PointPortModel[] = [
    {
      id: 'port2',
      offset: {
        x: 1,
        y: 0.5
      },
      visibility: PortVisibility.Visible
    },
    {
      id: 'port3',
      offset: {
        x: 0.5,
        y: 0
      },
      visibility: PortVisibility.Visible
    }
  ]
}

```

```

    }
  ]
  public getNodeDefaults(node: NodeModel | any): NodeModel {
    node.height = 100;
    node.width = 100;
    ((node as NodeModel).style as ShapeStyleModel).fill = "#6BA5D7";
    ((node as NodeModel).style as ShapeStyleModel).strokeColor =
    "White";
    return node;
  }
  public getConnectorDefaults(obj: ConnectorModel): void {
    obj.style = {
      strokeColor: '#6BA5D7',
      fill: '#6BA5D7',
      strokeWidth: 2
    }
    obj.targetDecorator = {
      style: {
        fill: '#6BA5D7',
        strokeColor: '#6BA5D7'
      }
    }
  }
  public created(args: Object): void {
    // Update the target portID at the run time
    (this.diagram as DiagramComponent).connectors[0].targetPortID =
    'port3';
  }
}

```

MAIN.TS

```

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

```

- When you set PortConstraints to [InConnect](#), the port accepts only an incoming connection to dock in it. Similarly, when you set PortConstraints to [OutConnect](#), the port accepts only an outgoing connection to dock in it.
- When you set PortConstraints to None, the port restricts connector to establish connection in it.

`typescript

```

import { Component, ViewEncapsulation, ViewChild } from "@angular/core";
import { PointPortModel, PortConstraints } from "@syncfusion/ej2-angular-diagrams";
@Component({
  selector: "app-container",
  template: `<ejs-diagram id="diagram" width="100%" height="580px">
<e-nodes>

```

```

<e-node id='node1' [height]=60 [width]=100 [offsetX]=300 [offsetY]=80 [ports]='port1'>
</e-node>
</e-nodes>
</ejs-diagram>`,
encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild("diagram")
  public port1: PointPortModel[];
  ngOnInit(): void {
    this.port1 = [{
      id: 'port1',
      shape: 'Circle',
      offset: { x: 0, y: 0.5 },
      text: 'In - 1',
      constraints: PortConstraints.InConnect
    }];
  }
}
`

```

Segments

The path of the connector is defined with a collection of segments. There are three types of segments.

Straight

To create a straight line, specify the [type](#) of the segment as **straight** and add a straight segment to [segments](#) collection and need to specify [type](#) for the connector. The following code example illustrates how to create a default straight segment.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, OnInit, ViewChild } from '@angular/core';
import { DiagramComponent, Diagram, NodeModel, ConnectorModel,
  StraightSegmentModel, PointModel } from '@syncfusion/ej2-angular-diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [ ],

```

```

standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%"
height="580px" [getConnectorDefaults] ='getConnectorDefaults'>
  <e-connectors>
    <e-connector id='connector' type='Straight'
[sourcePoint]='sourcePoint' [targetPoint]='targetPoint'
[segments]='segments'>
    </e-connector>
  </e-connectors>
</ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram?: DiagramComponent;
  public sourcePoint?: PointModel;
  public targetPoint?: PointModel;
  public segments?: StraightSegmentModel;
  ngOnInit(): void {
    this.sourcePoint = { x: 100, y: 100 };
    this.targetPoint = { x: 200, y: 200 };
    this.segments = [{
      // Defines the segment type of the connector
      type: 'Straight'
    }] as StraightSegmentModel;
  }
  public getConnectorDefaults(obj: ConnectorModel): void {
    obj.style = {
      strokeColor: '#6BA5D7',
      fill: '#6BA5D7',
      strokeWidth: 2
    }
    obj.targetDecorator = {
      style: {
        fill: '#6BA5D7',
        strokeColor: '#6BA5D7'
      }
    }
  }
}

```

MAIN.TS

```

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

```

The [point](#) property of straight segment allows you to define the end point of it. The following code example illustrates how to define the end point of a straight segment.

APP.COMPONENT.TS

```

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

```



```

import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, OnInit, ViewChild } from
 '@angular/core';
import { DiagramComponent, Diagram, NodeModel, ConnectorModel,
 StraightSegmentModel, PointModel } from '@syncfusion/ej2-angular-diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%"
height="580px" [getConnectorDefaults] ='getConnectorDefaults'>
    <e-connectors>
      <e-connector id='connector' type='Straight'
[sourcePoint]='sourcePoint' [targetPoint]='targetPoint'
[segments]='segments'>
        </e-connector>
      </e-connectors>
    </ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram?: DiagramComponent;
  public sourcePoint?: PointModel;
  public targetPoint?: PointModel;
  public segments?: StraightSegmentModel;
  ngOnInit(): void {
    this.sourcePoint = { x: 100, y: 100 };
    this.targetPoint = { x: 200, y: 200 };
    this.segments = [{
      // Defines the segment type of the connector
      type: 'Straight',
      point: { x: 100, y: 150 }
    }] as StraightSegmentModel;
  }
  public getConnectorDefaults(obj: ConnectorModel): void {
    obj.style = {
      strokeColor: '#6BA5D7',
      fill: '#6BA5D7',
      strokeWidth: 2
    }
    obj.targetDecorator = {
      style: {
        fill: '#6BA5D7',
        strokeColor: '#6BA5D7'
      }
    }
  }
}

```

MAIN.TS

```

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

```

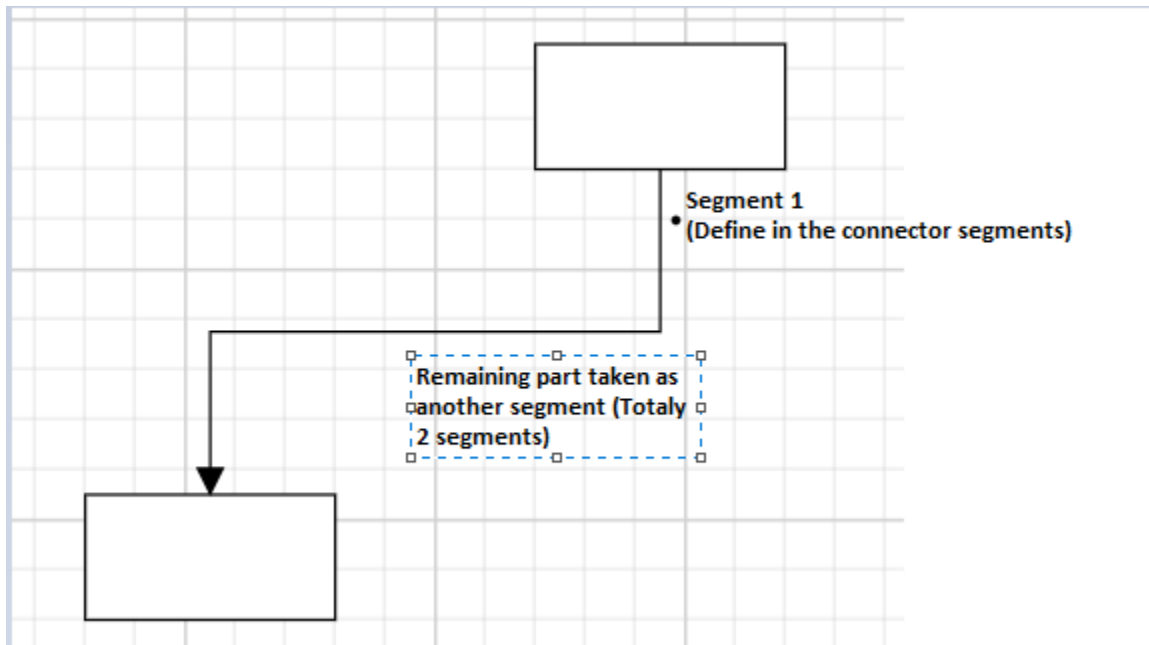
```
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Orthogonal

Orthogonal segments is used to create segments that are perpendicular to each other.

Set the segment [type](#) as orthogonal to create a default orthogonal segment and need to specify [type](#). The following code example illustrates how to create a default orthogonal segment.

If we set the segments for the connector in the diagram, the connector's first segment is rendered based on the given segment by default. Then the remaining segments are considered as single segment. Please refer the below screenshot.



If no segments are defined for the connector means, then the whole connector is considered as a single segment. We adopted this method for rendering the connector and its segments.

Multiple segments can be defined one after another. To create a connector with multiple segments, define and add the segments to [connector.segments](#) collection. The following code example illustrates how to create a connector with multiple segments.

The property [maxSegmentThumb](#) is used to limit the segment thumb in the connector.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, OnInit, ViewChild } from
 '@angular/core';
import { DiagramComponent, Diagram, NodeModel,
ConnectorModel, ConnectorEditing, OrthogonalSegmentModel, PointModel } from
 '@syncfusion/ej2-angular-diagrams';
import { ConnectorConstraints } from '@syncfusion/ej2-diagrams';
Diagram.Inject(ConnectorEditing);
```

```

@Component({
  imports: [
    DiagramModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%"
height="580px" [getConnectorDefaults] ='getConnectorDefaults'>
  <e-connectors>
    <e-connector id='connector' type='Orthogonal'
[sourcePoint]='sourcePoint' [targetPoint]='targetPoint'
maxSegmentThumb='maxSegmentThumb' [constraints] ='constraints'
[segments]='segments'>
    </e-connector>
  </e-connectors>
</ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram?: DiagramComponent;
  public sourcePoint?: PointModel;
  public targetPoint?: PointModel;
  public maxSegmentThumb?: number;
  public constraints?: ConnectorConstraints;
  public segments?: OrthogonalSegmentModel;
  ngOnInit(): void {
    this.sourcePoint = { x: 100, y: 100 };
    this.targetPoint = { x: 200, y: 200 };
    this.maxSegmentThumb = 3;
    this.constraints = ConnectorConstraints.Default &
~ConnectorConstraints.DragSegmentThumb;
    this.segments = [{
      // Defines the segment type of the connector
      type: 'Orthogonal'
    }]
  }
  public getConnectorDefaults(obj: ConnectorModel): void {
    obj.style = {
      strokeColor: '#6BA5D7',
      fill: '#6BA5D7',
      strokeWidth: 2
    }
    obj.targetDecorator = {
      style: {
        fill: '#6BA5D7',
        strokeColor: '#6BA5D7'
      }
    }
  }
}

```

MAIN.TS

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

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

The [length](#) and [direction](#) properties allow to define the flow and length of segment. The following code example illustrates how to create customized orthogonal segments.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, OnInit, ViewChild } from '@angular/core';
import { DiagramComponent, Diagram, NodeModel, ConnectorModel,
OrthogonalSegmentModel, PointModel } from '@syncfusion/ej2-angular-
diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%"
height="580px" [getConnectorDefaults] ='getConnectorDefaults'>
  <e-connectors>
    <e-connector id='connector' type='Orthogonal'
[sourcePoint]='sourcePoint' [targetPoint]='targetPoint'
[segments]='segments'>
    </e-connector>
  </e-connectors>
</ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram?: DiagramComponent;
  public sourcePoint?: PointModel;
  public targetPoint?: PointModel;
  public segments?: OrthogonalSegmentModel;
  ngOnInit(): void {
    this.sourcePoint = { x: 100, y: 100 };
    this.targetPoint = { x: 200, y: 200 };
    this.segments = [
      {
        type: 'Orthogonal',
        direction: 'Bottom',
        length: 150
      },
      {
        type: 'Orthogonal',
        direction: 'Right',
        length: 150
      }
    ]
  }
}
```

```

    }
    public getConnectorDefaults(obj: ConnectorModel): void {
        obj.style = {
            strokeColor: '#6BA5D7',
            fill: '#6BA5D7',
            strokeWidth: 2
        }
        obj.targetDecorator = {
            style: {
                fill: '#6BA5D7',
                strokeColor: '#6BA5D7'
            }
        }
    }
}

```

MAIN.TS

```

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

```

Note: You need to mention the segment type as same as what you mentioned in connector type. There should be no contradiction between connector type and segment type.

Avoid overlapping

Orthogonal segments are automatically re-routed, in order to avoid overlapping with the source and target nodes. The following preview illustrates how orthogonal segments are re-routed.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import { DiagramComponent, Diagram, NodeModel, ConnectorModel,
OrthogonalSegmentModel, PointModel, PortVisibility, PointPortModel,
ShapeStyleModel } from '@syncfusion/ej2-angular-diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%"
height="580px" [getNodeDefaults] ='getNodeDefaults' [getConnectorDefaults]
='getConnectorDefaults'>
    <e-nodes>
      <e-node id='node1' [offsetX]=150 [offsetY]=150>
        <e-node-annotations>
          <e-node-annotation content="Node1">
            </e-node-annotation>
          </e-node-annotations>
        </e-node>
      </e-nodes>
    `
})
export class AppComponent {
}

```

```

        </e-node>
        <e-node id='node2' [offsetX]=350 [offsetY]=150>
            <e-node-annotations>
                <e-node-annotation content="Custom Template">
                </e-node-annotation>
            </e-node-annotations>
        </e-node>
    </e-nodes>
    <e-connectors>
        <e-connector id='connector' type='Straight' sourceID='node1'
sourcCrPortID='port1' targetID='node2' targetPortID='port2'>
        </e-connector>
    </e-connectors>
</ejs-diagram>`,
    encapsulation: ViewEncapsulation.None
})
export class AppComponent {
    @ViewChild("diagram")
    public diagram?: DiagramComponent;
    public port1: PointPortModel[] = [{
        id: 'port1',
        offset: {
            x: 0,
            y: 0.5
        },
        visibility: PortVisibility.Visible
    }]
    public port2: PointPortModel[] = [
        {
            id: 'port2',
            offset: {
                x: 0.5,
                y: 0
            },
            visibility: PortVisibility.Visible
        }
    ]
    public getNodeDefaults(node: NodeModel | any): NodeModel {
        node.height = 100;
        node.width = 100;
        ((node as NodeModel).style as ShapeStyleModel).fill = "#6BA5D7";
        ((node as NodeModel).style as ShapeStyleModel).strokeColor =
"White";
        return node;
    }
    public getConnectorDefaults(obj: ConnectorModel): void {
        obj.style = {
            strokeColor: '#6BA5D7',
            fill: '#6BA5D7',
            strokeWidth: 2
        }
        obj.targetDecorator = {
            style: {
                fill: '#6BA5D7',
                strokeColor: '#6BA5D7'
            }
        }
    }
}

```

```
}
}
```

MAIN.TS

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

How to customize Orthogonal Segment Thumb Shape

The orthogonal connector has a number of segments in between the source and the target point. The segments are rendered with the default shape rhombus. Now, the option has been provided to change the segment thumb shape using the [segmentThumbShape](#) property. The predefined shapes provided are as follows:

- Rhombus
- Square
- Rectangle
- Ellipse
- Arrow
- Diamond
- OpenArrow
- Circle
- Fletch
- OpenFetch
- IndentedArrow
- OutdentedArrow
- DoubleArrow

You can customize the style of the thumb shape by overriding the class e-orthogonal-thumb.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, OnInit, ViewChild } from '@angular/core';
import { DiagramComponent, Diagram, NodeModel, ConnectorEditing, ConnectorModel, ConnectorConstraints, OrthogonalSegmentModel, PointModel } from '@syncfusion/ej2-angular-diagrams';
Diagram.Inject(ConnectorEditing);
@Component({
  imports: [
    DiagramModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="900px" height="500px" [getConnectorDefaults] ='getConnectorDefaults'>`
})
```

```

        <e-connectors>
            <e-connector id='connector2' type='Orthogonal'
[sourcePoint]='sourcePoint' [targetPoint]='targetPoint'
[segments]='segments'>
                </e-connector>
            </e-connectors>
        </ejs-diagram>`,
        encapsulation: ViewEncapsulation.None
    })
    export class AppComponent {
        @ViewChild("diagram")
        public diagram?: DiagramComponent;
        public sourcePoint?: PointModel;
        public targetPoint?: PointModel;
        public segments?: OrthogonalSegmentModel;
        ngOnInit(): void {
            this.sourcePoint = { x: 250, y: 250 };
            this.targetPoint = { x: 350, y: 350 };
            this.segments = [
                {
                    type: 'Orthogonal',
                    // Defines the direction for the segment lines
                    direction: "Right",
                    // Defines the length for the segment lines
                    length: 70
                },
                {
                    type: 'Orthogonal',
                    direction: "Bottom",
                    length: 20
                }
            ]
        }
        public getConnectorDefaults(connector: ConnectorModel): void {
            connector.style = {
                strokeColor: '#6BA5D7',
                fill: '#6BA5D7',
                strokeWidth: 2
            }
            connector.constraints = ConnectorConstraints.Default |
ConnectorConstraints.DragSegmentThumb;
        }
    }
}

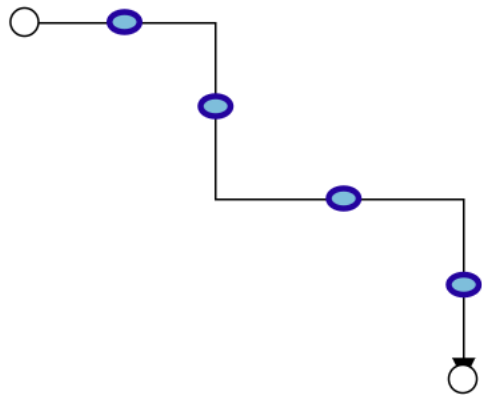
```

MAIN.TS

```

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

```

Use the following CSS to customize the segment thumb shape.

```
`scss
.e-orthogonal-thumb {
fill: rgb(126, 190, 219);
stroke: #24039e;
stroke-width: 3px;
}
`
```

Bezier

Bezier segments are used to create curve segments and the curves are configurable either with the control points or with vectors.

To create a bezier segment, the [segment.type](#) is set as `bezier` and need to specify [type](#) for the connector. The following code example illustrates how to create a default bezier segment.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, OnInit, ViewChild } from
 '@angular/core';
import { DiagramComponent, Diagram, NodeModel, ConnectorModel,
BezierSegmentModel, PointModel, OrthogonalSegmentModel } from
 '@syncfusion/ej2-angular-diagrams';
@Component({
imports: [
```

```

    DiagramModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%"
height="580px" [getConnectorDefaults] ='getConnectorDefaults'>
    <e-connectors>
      <e-connector id='connector' type='Bezier'
[sourcePoint]='sourcePoint' [targetPoint]='targetPoint'
[segments]='segments'>
        </e-connector>
      </e-connectors>
    </ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram?: DiagramComponent;
  public sourcePoint?: PointModel;
  public targetPoint?: PointModel;
  public segments?: OrthogonalSegmentModel;
  ngOnInit(): void {
    this.sourcePoint = { x: 100, y: 100 };
    this.targetPoint = { x: 200, y: 200 };
    this.segments = [
      {
        type: 'Bezier'
      }
    ]
  }
  public getConnectorDefaults(obj: ConnectorModel): void {
    obj.style = {
      strokeColor: '#6BA5D7',
      fill: '#6BA5D7',
      strokeWidth: 2
    }
    obj.targetDecorator = {
      style: {
        fill: '#6BA5D7',
        strokeColor: '#6BA5D7'
      }
    }
  }
}

```

MAIN.TS

```

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

```

The [point1](#) and [point2](#) properties of bezier segment enable you to set the control points. The following code example illustrates how to configure the bezier segments with control points.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, OnInit, ViewChild } from
 '@angular/core';
import { DiagramComponent, Diagram, NodeModel, ConnectorModel,
 BezierSegmentModel, PointModel, OrthogonalSegmentModel } from
 '@syncfusion/ej2-angular-diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%"
height="580px" [getConnectorDefaults] ='getConnectorDefaults'>
    <e-connectors>
      <e-connector id='connector' type='Bezier'
[sourcePoint]='sourcePoint' [targetPoint]='targetPoint'
[segments]='segments'>
        </e-connector>
      </e-connectors>
    </ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram?: DiagramComponent;
  public sourcePoint?: PointModel;
  public targetPoint?: PointModel;
  public segments?: OrthogonalSegmentModel;
  ngOnInit(): void {
    this.sourcePoint = { x: 100, y: 200 };
    this.targetPoint = { x: 200, y: 100 };
    this.segments = [{
      type: 'Bezier',
      // First control point: an absolute position from the page
      origin
      point1: {
        x: 100,
        y: 100
      },
      // Second control point: an absolute position from the page
      origin
      point2: {
        x: 200,
        y: 200
      }
    }
  ]
}
  public getConnectorDefaults(obj: ConnectorModel): void {
    obj.style = {
      strokeColor: '#6BA5D7',
      fill: '#6BA5D7',

```

```

        strokeWidth: 2
      }
      obj.targetDecorator = {
        style: {
          fill: '#6BA5D7',
          strokeColor: '#6BA5D7'
        }
      }
    }
  }
}

```

MAIN.TS

```

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

```

The [vector1](#) and [vector2](#) properties of bezier segment enable you to define the vectors. The following code illustrates how to configure a bezier curve with vectors.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, OnInit, ViewChild } from '@angular/core';
import { DiagramComponent, Diagram, NodeModel, ConnectorModel, BezierSegmentModel, PointModel, OrthogonalSegmentModel } from '@syncfusion/ej2-angular-diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%" height="580px" [getConnectorDefaults] ='getConnectorDefaults'>
    <e-connectors>
      <e-connector id='connector' type='Bezier'
[sourcePoint]='sourcePoint' [targetPoint]='targetPoint'
[segments]='segments'>
        </e-connector>
      </e-connectors>
    </ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram?: DiagramComponent;
  public sourcePoint?: PointModel;
  public targetPoint?: PointModel;
  public segments?: OrthogonalSegmentModel;
  ngOnInit(): void {

```

```

    this.sourcePoint = { x: 100, y: 100 };
    this.targetPoint = { x: 200, y: 200 };
    this.segments = [{
      type: 'Bezier',
      // Length and angle between the source point and the first
control point
      vector1: {
        distance: 100,
        angle: 90
      },
      // Length and angle between the target point and the second
control point
      vector2: {
        distance: 45,
        angle: 270
      }
    }]
  }
  public getConnectorDefaults(obj: ConnectorModel): void {
    obj.style = {
      strokeColor: '#6BA5D7',
      fill: '#6BA5D7',
      strokeWidth: 2
    }
    obj.targetDecorator = {
      style: {
        fill: '#6BA5D7',
        strokeColor: '#6BA5D7'
      }
    }
  }
}

```

MAIN.TS

```

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

```

Avoid overlapping with bezier

By default, when there are no segments defined for a bezier connector, the bezier segments will be created automatically and routed in such a way that avoids overlapping with the source and target nodes.

`typescript

```

import { Component, ViewEncapsulation, OnInit, ViewChild } from '@angular/core';

import {
  DiagramComponent, Diagram, NodeModel, ConnectorModel, ConnectorEditing, ConnectorConstraints }
from '@syncfusion/ej2-angular-diagrams';

Diagram.Inject(ConnectorEditing);

@Component({

```

```
selector: "app-container",
template: `<ejs-diagram #diagram id="diagram" width="100%" height="600px"
[getNodeDefaults]='getNodeDefaults' [getConnectorDefaults]='getConnectorDefaults'>
<e-nodes>
<e-node id="Start" [offsetX]=250 [offsetY]=150>
<e-node-annotations>
<e-node-annotation content="Start">
</e-node-annotation>
</e-node-annotations>
</e-node>
<e-node id='End' [offsetX]=450 [offsetY]=200>
<e-node-annotations>
<e-node-annotation content="End">
</e-node-annotation>
</e-node-annotations>
</e-node>
</e-nodes>
<e-connectors>
<e-connector id='connector1' type='Bezier' sourceID='Start' targetID='End'>
</e-connector>
</e-connectors>
</ejs-diagram>`,
encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram: DiagramComponent;
  public getNodeDefaults(node: NodeModel): NodeModel {
    node.height = 100;
    node.width = 100;
    node.shape = { type: 'Basic', shape: 'Rectangle' };
    node.style.fill = "#6BA5D7";
    node.style.strokeColor = "White";
```

```

return node;
}

public getConnectorDefaults(connector: ConnectorModel): ConnectorModel {
  connector.style: {
    strokeColor: '#6BA5D7',
    fill: '#6BA5D7',
    strokeWidth: 2
  },
  connector.targetDecorator: { shape: 'None' },
  connector.constraints = ConnectorConstraints.Default | ConnectorConstraints.DragSegmentThumb;
}
}
`

```

Also, the intermediate point of two adjacent bezier segments can be edited interactively based on the `bezierSettings.segmentEditOrientation` property of the connector class.

How to interact with the bezier segments efficiently

While interacting with multiple bezier segments, maintain their control points at the same distance and angle by using the `bezierSettings.smoothness` property of the connector class.

BezierSmoothness value	Description
----- -----	
SymmetricDistance	Both control points of adjacent segments will be at the same distance when any one of them is editing.
SymmetricAngle	Both control points of adjacent segments will be at the same angle when any one of them is editing.
Default	Both control points of adjacent segments will be at the same angle and same distance when any one of them is editing.
None	Segment's control points are interacted independently from each other.

```

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

import {
  DiagramComponent, Diagram, NodeModel, ConnectorModel, ConnectorEditing, ConnectorConstraints }
  from '@syncfusion/ej2-angular-diagrams';

Diagram.Inject(ConnectorEditing);

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

```

```

template: `<ejs-diagram #diagram id="diagram" width="100%" height="600px"
[getNodeDefaults]='getNodeDefaults' [getConnectorDefaults]='getConnectorDefaults'>
<e-nodes>
<e-node id="Start" [offsetX]=250 [offsetY]=150 [ports]='StartPort'>
<e-node-annotations>
<e-node-annotation content="Start">
</e-node-annotation>
</e-node-annotations>
</e-node>
<e-node id='End' [offsetX]=450 [offsetY]=200 [ports]='EndPort'>
<e-node-annotations>
<e-node-annotation content="End">
</e-node-annotation>
</e-node-annotations>
</e-node>
</e-nodes>
<e-connectors>
<e-connector id='connector1' type='Bezier' sourceID='Start' targetID='End'
sourcePortID='StartPort' targetPortID='EndPort' [bezierSettings]='bezierSettings'>
</e-connector>
</e-connectors>
</ejs-diagram>`,
encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram: DiagramComponent;
  public StartPort: PointPortModel[];
  public EndPort: PointPortModel[];
  public bezierSettings;
  ngOnInit(): void {
    this.StartPort = [{
      id: 'StartPort',
      visibility: PortVisibility.Visible,

```



```
shape: 'Circle',
offset: { x: 1, y: 0.5 },
style: { strokeColor: '#366F8C', fill: '#366F8C' }
});
this.EndPort = [{
id: 'EndPort',
visibility: PortVisibility.Visible,
shape: 'Circle',
offset: { x: 0, y: 0.5 },
style: { strokeColor: '#366F8C', fill: '#366F8C' }
}];
this.bezierSettings = { smoothness: BezierSmoothness.SymmetricAngle };
}
public getNodeDefaults(node: NodeModel): NodeModel {
node.height = 100;
node.width = 100;
node.shape = { type: 'Basic', shape: 'Rectangle' };
node.style.fill = "#6BA5D7";
node.style.strokeColor = "White";
return node;
}
public getConnectorDefaults(connector: ConnectorModel): ConnectorModel {
connector.style: {
strokeColor: '#6BA5D7',
fill: '#6BA5D7',
strokeWidth: 2
},
connector.targetDecorator: { shape: 'None' },
connector.constraints = ConnectorConstraints.Default | ConnectorConstraints.DragSegmentThumb;
}
}
```

Also, the visibility of control points can be controlled using the `bezierSettings.controlPointsVisibility` property of the connector class.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, OnInit, ViewChild } from
 '@angular/core';
import { DiagramComponent, Diagram, NodeModel,
ConnectorModel, ConnectorEditing, ConnectorConstraints, BezierSegmentModel,
PointPortModel, ControlPointsVisibility, PortVisibility, ShapeStyleModel }
from '@syncfusion/ej2-angular-diagrams';
Diagram.Inject(ConnectorEditing);
@Component({
  imports: [
    DiagramModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%"
height="600px" [getNodeDefaults]='getNodeDefaults' [getConnectorDefaults]
='getConnectorDefaults'>
    <e-nodes>
      <e-node id='Start' [offsetX]=250 [offsetY]=150
[ports]='StartPort'>
        <e-node-annotations>
          <e-node-annotation content="Start">
            </e-node-annotation>
          </e-node-annotations>
        </e-node>
      <e-node id='End' [offsetX]=450 [offsetY]=200 [ports]='EndPort'>
        <e-node-annotations>
          <e-node-annotation content="End">
            </e-node-annotation>
          </e-node-annotations>
        </e-node>
      </e-nodes>
      <e-connectors>
        <e-connector id='connector1' type='Bezier' sourceID='Start'
targetID='End' sourcePortID='StartPort' targetPortID='EndPort'
bezierSettings='bezierSettings'>
          </e-connector>
        </e-connectors>
      </ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram?: DiagramComponent;
  public StartPort?: PointPortModel[];
  public EndPort?: PointPortModel[];
  public bezierSettings?: any;
  ngOnInit(): void {
    this.StartPort = [{
```

```

        id: 'StartPort',
        visibility: PortVisibility.Visible,
        shape: 'Circle',
        offset: { x: 1, y: 0.5 },
        style: { strokeColor: '#366F8C', fill: '#366F8C' }
    }];
    this.EndPort = [{
        id: 'EndPort',
        visibility: PortVisibility.Visible,
        shape: 'Circle',
        offset: { x: 0, y: 0.5 },
        style: { strokeColor: '#366F8C', fill: '#366F8C' }
    }];
    this.bezierSettings = { controlPointsVisibility:
ControlPointsVisibility.Source | ControlPointsVisibility.Target };
    }

    public getNodeDefaults(node: NodeModel | any): NodeModel {
        node.height = 100;
        node.width = 100;
        node.shape = { type: 'Basic', shape: 'Rectangle' };
        ((node as NodeModel).style as ShapeStyleModel).fill = "#6BA5D7";
        ((node as NodeModel).style as ShapeStyleModel).strokeColor =
"White";
        return node;
    }

    public getConnectorDefaults(connector: ConnectorModel): void {
        connector.style = {
            strokeColor: '#6BA5D7',
            fill: '#6BA5D7',
            strokeWidth: 2
        },
        connector.targetDecorator = { shape: 'None' },
        connector.constraints = ConnectorConstraints.Default |
ConnectorConstraints.DragSegmentThumb;
    }
}

```

MAIN.TS

```

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

```

Decorator

- Starting and ending points of a connector can be decorated with some customizable shapes like arrows, circles, diamond, or path. The connection end points can be decorated with the [sourceDecorator](#) and [targetDecorator](#) properties of the connector.
- The [shape](#) property of [sourceDecorator](#) allows to define the shape of the decorators. Similarly, the [shape](#) property of [targetDecorator](#) allows to define the shape of the decorators.
- To create custom shape for source decorator, use [pathData](#) property. Similarly, to create custom shape for target decorator, use [pathData](#) property.

The following code example illustrates how to create decorators of various shapes.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, OnInit, ViewChild } from
 '@angular/core';
import { DiagramComponent, Diagram, ConnectorModel, DecoratorModel,
PointModel, OrthogonalSegmentModel } from '@syncfusion/ej2-angular-
diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%"
height="580px" [getConnectorDefaults] ='getConnectorDefaults'>
    <e-connectors>
      <e-connector id='connector' type='Bezier'
[sourcePoint]='sourcePoint' [targetPoint]='targetPoint'
[sourceDecorator]='sourceDecorator' [targetDecorator]='targetDecorator'>
        </e-connector>
      </e-connectors>
    </ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram?: DiagramComponent;
  public sourcePoint?: PointModel;
  public targetPoint?: PointModel;
  public sourceDecorator?: DecoratorModel;
  public targetDecorator?: DecoratorModel;
  public segments?: OrthogonalSegmentModel;
  ngOnInit(): void {
    this.sourcePoint = { x: 100, y: 100 };
    this.targetPoint = { x: 200, y: 200 };
    this.sourceDecorator = {
      shape: 'Circle',
      // Defines the style for the sourceDecorator
      style: {
        // Defines the strokeWidth for the sourceDecorator
        strokeWidth: 3,
        // Defines the strokeColor for the sourceDecorator
        strokeColor: 'red'
      },
    },
    // Decorator shape - Diamond
    this.targetDecorator = {
      // Defines the custom shape for the connector's target decorator
      shape: 'Custom',
      // Defines the path for the connector's target decorator

```

```

        pathData: 'M80.5,12.5 C80.5,19.127417 62.59139,24.5 40.5,24.5
C18.40861,24.5 0.5,19.127417 0.5,12.5' +
        'C0.5,5.872583 18.40861,0.5 40.5,0.5 C62.59139,0.5
80.5,5.872583 80.5,12.5 z',
        //defines the style for the target decorator
        style: {
            // Defines the strokeWidth for the targetDecorator
            strokeWidth: 3,
            // Defines the strokeColor for the sourceDecorator
            strokeColor: 'green',
            // Defines the opacity for the sourceDecorator
            opacity: .8
        }
    } as DecoratorModel;
}
public getConnectorDefaults(obj: ConnectorModel): void {
    obj.style = {
        strokeColor: '#6BA5D7',
        fill: '#6BA5D7',
        strokeWidth: 2
    }
}
}
}

```

MAIN.TS

```

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

```

Padding

Padding is used to leave the space between the Connector's end point and the object to where it is connected.

- The [sourcePadding](#) property of connector defines space between the source point and the source node of the connector.
- The [targetPadding](#) property of connector defines space between the end point and the target node of the connector.

The following code example illustrates how to leave space between the connection end points and source and target nodes.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import { DiagramComponent, Diagram, NodeModel, ConnectorModel,
ShapeStyleModel } from '@syncfusion/ej2-angular-diagrams';
@Component({
    imports: [
        DiagramModule
    ]
})

```

```

    ],
    providers: [ ],
    standalone: true,
    selector: "app-container",
    template: `<ejs-diagram #diagram id="diagram" width="100%"
height="580px" [getNodeDefaults] ='getNodeDefaults' [getConnectorDefaults]
='getConnectorDefaults'>
      <e-nodes>
        <e-node id='node1' [offsetX]=150 [offsetY]=150>
          <e-node-annotations>
            <e-node-annotation content="Node1">
            </e-node-annotation>
          </e-node-annotations>
        </e-node>
        <e-node id='node2' [offsetX]=350 [offsetY]=150>
          <e-node-annotations>
            <e-node-annotation content="Custom Template">
            </e-node-annotation>
          </e-node-annotations>
        </e-node>
      </e-nodes>
      <e-connectors>
        <e-connector id='connector' type='Straight' sourceID='node1'
targetID='node2'>
        </e-connector>
      </e-connectors>
    </ejs-diagram>`,
    encapsulation: ViewEncapsulation.None
  })
}
export class AppComponent {
  @ViewChild("diagram")
  public diagram?: DiagramComponent;
  public getNodeDefaults(node: NodeModel | any): NodeModel {
    node.height = 100;
    node.width = 100;
    ((node as NodeModel).style as ShapeStyleModel).fill = "#6BA5D7";
    ((node as NodeModel).style as ShapeStyleModel).strokeColor =
    "White";
    return node;
  }
  public getConnectorDefaults(obj: ConnectorModel): void {
    obj.style = {
      strokeColor: '#6BA5D7',
      fill: '#6BA5D7',
      strokeWidth: 2
    }
    // Set Source Padding value
    obj.sourcePadding = 20
    // Set Target Padding value
    obj.targetPadding = 20
    obj.targetDecorator = {
      style: {
        fill: '#6BA5D7',
        strokeColor: '#6BA5D7'
      }
    }
  }
}

```

```
}

```

MAIN.TS

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

Hit padding

- The [hitPadding](#) property enables you to define the clickable area around the connector path. The default value for hitPadding is 10.
- The following code example illustrates how to specify hit padding for connector.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import { DiagramComponent, Diagram, NodeModel, ConnectorModel,
ShapeStyleModel, PointModel, ConnectorConstraints, ConnectorEditing } from
 '@syncfusion/ej2-angular-diagrams';
Diagram.Inject(ConnectorEditing);
@Component({
  imports: [
    DiagramModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%"
height="580px" [getConnectorDefaults] ='getConnectorDefaults'>
  <e-connectors>
    <e-connector id='connector' type='Orthogonal'
[sourcePoint]='sourcePoint' [targetPoint]='targetPoint'
[constraints]="constraints" [hitPadding]="hitPadding">
    </e-connector>
  </e-connectors>
</ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram?: DiagramComponent;
  public constraints?: ConnectorConstraints;
  public sourcePoint?: PointModel;
  public targetPoint?: PointModel;
  public hitPadding?: number;
  ngOnInit(): void {
    this.sourcePoint = { x: 100, y: 100 };
    this.targetPoint = { x: 200, y: 200 };
    this.hitPadding = 50;
  }
}
```

```

        this.constraints = ConnectorConstraints.Default |
ConnectorConstraints.DragSegmentThumb
    }
    public getConnectorDefaults(obj: ConnectorModel): void {
        obj.style = {
            strokeColor: '#6BA5D7',
            fill: '#6BA5D7',
            strokeWidth: 2
        }
        obj.targetDecorator = {
            style: {
                fill: '#6BA5D7',
                strokeColor: '#6BA5D7'
            }
        }
    }
}

```

MAIN.TS

```

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

```

Flip

The diagram Provides support to flip the connector. The [flip](#) is performed to give the mirrored image of the original element.

The flip types are as follows:

- HorizontalFlip - [Horizontal](#) is used to interchange the connector source and target x points.
- VerticalFlip - [Vertical](#) is used to interchange the connector source and target y points.
- Both - [Both](#) is used to interchange the source point as target point and target point as source point

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, OnInit, ViewChild } from
 '@angular/core';
import { DiagramComponent, Diagram, ConnectorModel, DecoratorModel,
PointModel, OrthogonalSegmentModel } from '@syncfusion/ej2-angular-
diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",

```



```

    template: `<ejs-diagram #diagram id="diagram" width="100%"
height="580px" [getConnectorDefaults] ='getConnectorDefaults'>
    <e-connectors>
        <e-connector id='connector' type='Orthogonal'
[sourcePoint]='sourcePoint' [targetPoint]='targetPoint'>
        </e-connector>
    </e-connectors>
</ejs-diagram>`,
    encapsulation: ViewEncapsulation.None
})
export class AppComponent {
    @ViewChild("diagram")
    public diagram?: DiagramComponent;
    public sourcePoint?: PointModel;
    public targetPoint?: PointModel;
    public segments?: OrthogonalSegmentModel;
    ngOnInit(): void {
        this.sourcePoint = { x: 100, y: 100 };
        this.targetPoint = { x: 200, y: 200 };
    }
    public getConnectorDefaults(obj: ConnectorModel): void {
        obj.style = {
            strokeColor: '#6BA5D7',
            fill: '#6BA5D7',
            strokeWidth: 2
        }
        // Flip the connector in horizontal direction
        obj.flip = "Horizontal"
    }
}

```

MAIN.TS

```

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

```

Note: The flip is not applicable when the connectors connect in nodes.

Bridging

Line bridging creates a bridge for lines to smartly cross over the other lines, at points of intersection. By default, [bridgeDirection](#) is set to top. Depending upon the direction given bridging direction appears.

Bridging can be enabled/disabled either with the `connector.constraints` or `diagram.constraints`. The following code example illustrates how to enable line bridging.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule, ConnectorBridgingService } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, OnInit, ViewChild } from '@angular/core';

```

```

import { DiagramComponent, Diagram, ConnectorModel, DecoratorModel,
PointModel, DiagramConstraints } from '@syncfusion/ej2-angular-diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [ConnectorBridgingService],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%"
height="580px" [getConnectorDefaults] = 'getConnectorDefaults'
[constraints]='constraints'>
    <e-connectors>
      <e-connector id='connector1' type='Straight'
[sourcePoint]='sourcePoint1' [targetPoint]='targetPoint1'></e-connector>
      <e-connector id='connector2' type='Straight'
[sourcePoint]='sourcePoint2' [targetPoint]='targetPoint2'></e-connector>
    </e-connectors>
  </ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram?: DiagramComponent;
  public sourcePoint1?: PointModel;
  public targetPoint1?: PointModel;
  public sourcePoint2?: PointModel;
  public targetPoint2?: PointModel;
  public sourceDecorator?: DecoratorModel;
  public targetDecorator?: DecoratorModel;
  public constraints?: DiagramConstraints;
  ngOnInit(): void {
    this.sourcePoint1 = { x: 100, y: 100 };
    this.targetPoint1 = { x: 200, y: 200 };
    this.sourcePoint2 = { x: 200, y: 100 };
    this.targetPoint2 = { x: 100, y: 200 };
    this.constraints = DiagramConstraints.Default |
DiagramConstraints.Bridging;
  }
  public getConnectorDefaults(obj: ConnectorModel): void {
    obj.style = {
      strokeColor: '#6BA5D7',
      fill: '#6BA5D7',
      strokeWidth: 2
    }
    obj.targetDecorator = {
      style: {
        fill: '#6BA5D7',
        strokeColor: '#6BA5D7'
      }
    }
  }
}

```

MAIN.TS

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

Note: You need to inject connector bridging module into the diagram.

The [bridgeSpace](#) property of connectors can be used to define the width for line bridging.

Limitation: Bezier segments do not support bridging.

Corner radius

Corner radius allows to create connectors with rounded corners. The radius of the rounded corner is set with the [cornerRadius](#) property.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, OnInit, ViewChild } from
 '@angular/core';
import { DiagramComponent, Diagram, NodeModel, ConnectorModel,
OrthogonalSegmentModel, PointModel, ShapeStyleModel } from '@syncfusion/ej2-
angular-diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%"
height="580px" [getNodeDefaults] ='getNodeDefaults' [getConnectorDefaults]
='getConnectorDefaults'>
    <e-nodes>
      <e-node id='node1' [offsetX]=150 [offsetY]=150>
        <e-node-annotations>
          <e-node-annotation content="Node1">
            </e-node-annotation>
          </e-node-annotations>
        </e-node>
      <e-node id='node2' [offsetX]=350 [offsetY]=350>
        <e-node-annotations>
          <e-node-annotation content="Custom Template">
            </e-node-annotation>
          </e-node-annotations>
        </e-node>
      </e-nodes>
      <e-connectors>
        <e-connector id='connector' type='Straight' sourceID='node1'
sourcrPortID='port1' targetID='node2' targetPortID='port2'
[cornerRadius]='cornerRadius'>
          </e-connector>
        </e-connectors>
      </ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
```

```

}))
export class AppComponent {
  @ViewChild("diagram")
  public diagram?: DiagramComponent;
  public cornerRadius?: number;
  ngOnInit(): void {
    this.cornerRadius = 10;
  }
  public getNodeDefaults(node: NodeModel | any): NodeModel {
    node.height = 100;
    node.width = 100;
    ((node as NodeModel).style as ShapeStyleModel).fill = "#6BA5D7";
    ((node as NodeModel).style as ShapeStyleModel).strokeColor =
    "White";
    return node;
  }
  public getConnectorDefaults(obj: ConnectorModel): void {
    obj.style = {
      strokeColor: '#6BA5D7',
      fill: '#6BA5D7',
      strokeWidth: 2
    }
    obj.targetDecorator = {
      style: {
        fill: '#6BA5D7',
        strokeColor: '#6BA5D7'
      }
    }
  }
}
}

```

MAIN.TS

```

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

```

Appearance

- The connector's [strokeWidth](#), [strokeColor](#), [strokeDashArray](#), and [opacity](#) properties are used to customize the appearance of the connector segments.
- The [visible](#) property of the connector enables or disables the visibility of connector.
- Default values for all the connectors can be set using the [getConnectorDefaults](#) properties. For example, if all connectors have the same type or having the same property then such properties can be moved into [getConnectorDefaults](#).

Segment appearance

The following code example illustrates how to customize the segment appearance.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'

```

```

import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, OnInit, ViewChild } from
 '@angular/core';
import { DiagramComponent, Diagram, ConnectorModel, OrthogonalSegmentModel,
 PointModel, StrokeStyleModel } from '@syncfusion/ej2-angular-diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%"
height="580px" [getConnectorDefaults] ='getConnectorDefaults'>
    <e-connectors>
      <e-connector id='connector' type='Straight'
[sourcePoint]='sourcePoint' [targetPoint]='targetPoint'>
    </e-connector>
    </e-connectors>
  </ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram?: DiagramComponent;
  public sourcePoint?: PointModel;
  public targetPoint?: PointModel;
  public style?: StrokeStyleModel;
  ngOnInit(): void {
    this.sourcePoint = { x: 100, y: 100 };
    this.targetPoint = { x: 200, y: 200 };
    this.style = {
      // Stroke color
      strokeColor: '#6BA5D7',
      fill: '#6BA5D7',
      // Stroke width of the line
      strokeWidth: 2,
      // Line style
      strokeDashArray: '2,2'
    }
  }
  public getConnectorDefaults(obj: ConnectorModel): void {
    obj.targetDecorator = {
      style: {
        fill: '#6BA5D7',
        strokeColor: '#6BA5D7'
      }
    }
  }
}

```

MAIN.TS

```

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

```

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

Decorator appearance

- The source decorator's [strokeColor](#), [strokeWidth](#), and [strokeDashArray](#) properties are used to customize the color, width, and appearance of the decorator.
- To set the border stroke color, stroke width, and stroke dash array for the target decorator, use [strokeColor](#), [strokeWidth](#), and [strokeDashArray](#).
- To set the size for source and target decorator, use width and height property.

The following code example illustrates how to customize the appearance of the decorator.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, OnInit, ViewChild } from '@angular/core';
import { DiagramComponent, Diagram, ConnectorModel, DecoratorModel, PointModel } from '@syncfusion/ej2-angular-diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%" height="580px" [getConnectorDefaults] ='getConnectorDefaults'>
    <e-connectors>
      <e-connector id='connector' type='Bezier'
[sourcePoint]='sourcePoint' [targetPoint]='targetPoint'
[targetDecorator]='targetDecorator'>
    </e-connector>
    </e-connectors>
  </ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram?: DiagramComponent;
  public sourcePoint?: PointModel;
  public targetPoint?: PointModel;
  public targetDecorator?: DecoratorModel;
  ngOnInit(): void {
    this.sourcePoint = { x: 100, y: 100 };
    this.targetPoint = { x: 200, y: 200 };
    this.targetDecorator = {
      style: {
        // Fill color of the decorator
        fill: '#6BA5D7',
        // Stroke color of the decorator
        strokeColor: '#6BA5D7'
      }
    }
  }
}
```

```

    }
  }
}
public getConnectorDefaults(obj: ConnectorModel): void {
  obj.style = {
    strokeColor: '#6BA5D7',
    fill: '#6BA5D7',
    strokeWidth: 2
  }
}
}
}

```

MAIN.TS

```

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

```

Interaction

Diagram allows to edit the connectors at runtime. To edit the connector segments at runtime, refer to [Connection Editing](#).

Automatic line routing

Diagram provides additional flexibility to re-route the diagram connectors. A connector will frequently re-route itself when a shape moves next to it. The following screenshot illustrates how the connector automatically re-routes the segments.

1. Dependency LineRouting module should be injected to the application as the following code snippet.

`typescript

```

import { LineRouting, Diagram } from '@syncfusion/ej2-diagrams';
/

```

- Injecting the automatic line routing module.

*/

```

Diagram.Inject(LineRouting);
`

```

2. Now, the line routing constraints must be included to the default diagram constraints to enable automatic line routing support like below.

`html

/

- Initialize the Diagram

*/

```
<ejs-diagram #diagram [constraints]='constraints'>
```

```
,
```

```
`typescript
```

```
// Enable line routing constraints.
```

```
public constraints: DiagramConstraints = DiagramConstraints.Default | DiagramConstraints.LineRouting;
```

```
,
```

3.The following code block shows how to create the diagram with specifying nodes, connectors, constraints, and necessary modules for line routing.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import { Diagram, DiagramComponent, NodeModel, ConnectorModel } from
 '@syncfusion/ej2-angular-diagrams';
import { NodeConstraints, LineRouting, DiagramConstraints,
ConnectorConstraints, SnapConstraints, SnapSettingsModel } from
 '@syncfusion/ej2-diagrams';
Diagram.Inject(LineRouting);
@Component({
  imports: [
    DiagramModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%"
height="645px" [nodes]='nodes' [connectors]='connectors'
[getNodeDefaults]='getNodeDefaults' [constraints]='constraints'>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild('diagram')
  public diagram?: DiagramComponent;
  public constraints: DiagramConstraints = DiagramConstraints.Default |
DiagramConstraints.LineRouting;
  public nodes: NodeModel[] = [
    { id: 'shape1', offsetX: 100, offsetY: 100, width: 120, height: 50 },
    { id: 'shape2', offsetX: 300, offsetY: 300, width: 120, height: 50 },
    { id: 'shape3', offsetX: 150, offsetY: 200, width: 120, height: 50 }
  ];
  public connectors: ConnectorModel[] = [
    { id: 'connector', sourceID: 'shape1', targetID: 'shape2', type:
'Orthogonal' }
  ];
  public getNodeDefaults(obj: NodeModel): NodeModel {
    obj = { style: { strokeColor: '#6BA5D7', fill: '#6BA5D7' } };
    return obj;
  }
}
```


MAIN.TS

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

4. In some situations, automatic line routing enabled diagram needs to ignore a specific connector from automatic line routing. So, in this case, auto routing feature can be disabled to the specific connector using the [constraints](#) property of the connector like the following code snippet.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import { Diagram, DiagramComponent, NodeModel, ConnectorModel } from
 '@syncfusion/ej2-angular-diagrams';
import { NodeConstraints, LineRouting, DiagramConstraints,
ConnectorConstraints, SnapConstraints, SnapSettingsModel } from
 '@syncfusion/ej2-diagrams';
Diagram.Inject(LineRouting);
@Component({
  imports: [
    DiagramModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%"
height="645px" [nodes]='nodes' [connectors]='connectors'
[getNodeDefaults]='getNodeDefaults' [constraints]='constraints'>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild('diagram')
  public diagram?: DiagramComponent;
  public constraints: DiagramConstraints = DiagramConstraints.Default |
DiagramConstraints.LineRouting;
  public nodes: NodeModel[] = [
    { id: 'shapel', offsetX: 100, offsetY: 100, width: 120, height: 50 },
    { id: 'shape2', offsetX: 350, offsetY: 300, width: 120, height: 50 },
    { id: 'shape3', offsetX: 150, offsetY: 200, width: 120, height: 50 },
    { id: 'shape4', offsetX: 300, offsetY: 200, width: 120, height: 50 }
  ];
  public connectors: ConnectorModel[] = [
    { id: 'connector', sourceID: 'shapel', targetID: 'shape2', type:
'Orthogonal', annotations: [{ offset: .7, content: ' Routing \n enabled',
style: { fill: "white" } } ] },
    { id: 'connector2', sourceID: 'shapel', targetID: 'shape2', annotations:
[ { offset: .6, content: ' Routing \n disabled', style: { fill: "white" } } ],
type: 'Orthogonal', constraints: ConnectorConstraints.Default &
~ConnectorConstraints.InheritLineRouting }
  ];
  public getNodeDefaults(obj: NodeModel): NodeModel {
```

```

    obj = { style: { strokeColor: '#6BA5D7', fill: '#6BA5D7' } };
    return obj;
  }
}

```

MAIN.TS

```

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

```

Constraints

The [constraints](#) property of connector allows to enable/disable certain features of connectors. To enable or disable the constraints, refer [constraints](#).

The following code illustrates how to disable selection.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, OnInit, ViewChild } from '@angular/core';
import { DiagramComponent, Diagram, ConnectorModel, DecoratorModel,
PointModel, ConnectorConstraints } from '@syncfusion/ej2-angular-diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%"
height="580px" [getConnectorDefaults] ='getConnectorDefaults'>
  <e-connectors>
    <e-connector id='connector1' type='Straight'
[sourcePoint]='sourcePoint1' [targetPoint]='targetPoint1'
[constraints]='constraints'></e-connector>
  </e-connectors>
</ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram?: DiagramComponent;
  public sourcePoint1?: PointModel;
  public targetPoint1?: PointModel;
  public constraints?: ConnectorConstraints;
  ngOnInit(): void {
    this.sourcePoint1 = { x: 100, y: 100 };
    this.targetPoint1 = { x: 200, y: 200 };
    this.constraints = ConnectorConstraints.Default &
~ConnectorConstraints.Select;
  }
}

```

```

public getConnectorDefaults(obj: ConnectorModel): void {
  obj.style = {
    strokeColor: '#6BA5D7',
    fill: '#6BA5D7',
    strokeWidth: 2
  }
  obj.targetDecorator = {
    style: {
      fill: '#6BA5D7',
      strokeColor: '#6BA5D7'
    }
  }
}
}

```

MAIN.TS

```

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

```

Custom properties

The [addInfo](#) property of connectors allow you to maintain additional information to the connectors.

```
`html
```

```
<e-connectors>
```

```
<e-connector id='connector1' type='Straight' addInfo='centralconnector' [sourcePoint]='sourcePoint1'
[targetPoint]='targetPoint1' [constraints]='constraints'></e-connector>
```

```
</e-connectors>
```

```
`
```

Stack order

The connectors [zIndex](#) property specifies the stack order of the connector. A connector with greater stack order is always in front of a connector with a lower stack order.

The following code illustrates how to render connector based on the stack order.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, OnInit, ViewChild } from '@angular/core';
import { DiagramComponent, Diagram, ConnectorModel, DecoratorModel,
PointModel, ConnectorConstraints } from '@syncfusion/ej2-angular-diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [ ],
  standalone: true,

```

```

    selector: "app-container",
    template: `<ejs-diagram #diagram id="diagram" width="100%"
height="580px" [getConnectorDefaults] ='getConnectorDefaults'>
    <e-connectors>
        <e-connector id='connector1' type='Straight' zIndex=2
[sourcePoint]='sourcePoint1' [targetPoint]='targetPoint1'></e-connector>
        <e-connector id='connector2' type='Straight' zIndex=1
[sourcePoint]='sourcePoint2' [targetPoint]='targetPoint2'></e-connector>
    </e-connectors>
</ejs-diagram>`,
    encapsulation: ViewEncapsulation.None
})
export class AppComponent {
    @ViewChild("diagram")
    public diagram?: DiagramComponent;
    public sourcePoint1?: PointModel;
    public targetPoint1?: PointModel;
    public sourcePoint2?: PointModel;
    public targetPoint2?: PointModel;
    ngOnInit(): void {
        this.sourcePoint1 = { x: 100, y: 100 };
        this.targetPoint1 = { x: 200, y: 200 };
        this.sourcePoint2 = { x: 200, y: 100 };
        this.targetPoint2 = { x: 100, y: 200 };
    }
    public getConnectorDefaults(obj: ConnectorModel): void {
        obj.style = {
            strokeColor: '#6BA5D7',
            fill: '#6BA5D7',
            strokeWidth: 2
        }
        obj.targetDecorator = {
            style: {
                fill: '#6BA5D7',
                strokeColor: '#6BA5D7'
            }
        }
    }
}

```

MAIN.TS

```

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

```

Enable Connector Splitting

The connectors are used to create a link between two points, ports, or nodes to represent the relationship between them. Split the connector between two nodes when dropping a new node onto an existing connector and create a connection between the new node and existing nodes by setting the [enableConnectorSplit](#) as true. The default value of the [enableConnectorSplit](#) is false.

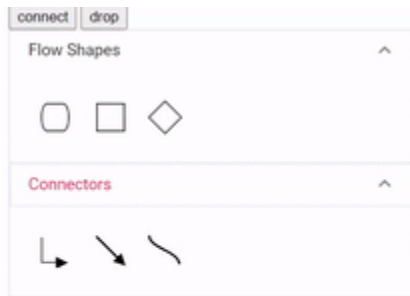
The following code illustrates how to split the connector and create a connection with new node.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, OnInit, ViewChild } from '@angular/core';
import { Diagram, DiagramComponent, NodeModel, ConnectorModel, ConnectorConstraints } from '@syncfusion/ej2-angular-diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%" height="580px" [enableConnectorSplit]="enableConnectorSplit" [nodes]='nodes' [connectors]='connectors'>
    </ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram?: DiagramComponent;
  public enableConnectorSplit: boolean = true;
  public constraints?: ConnectorConstraints;
  public nodes: NodeModel[] = [
    { id: 'node1', offsetX: 150, offsetY: 150, width: 100, height: 100, annotations: [{ content: 'node1' }] },
    { id: 'node2', offsetX: 650, offsetY: 150, width: 100, height: 100, annotations: [{ content: 'node2' }] },
    { id: 'node3', offsetX: 490, offsetY: 290, width: 100, height: 100, annotations: [{ content: 'node3' }] }
  ];
  public connectors: ConnectorModel[] = [{
    id: 'connector1', sourceID: "node1", targetID: "node2",
    constraints: ConnectorConstraints.Default | ConnectorConstraints.AllowDrop
  }
  ];
}
```

MAIN.TS

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



See Also

- [How to add annotations to the connector](#)
- [How to enable/disable the behavior of the node](#)
- [How to add connectors to the symbol palette](#)
- [How to perform the interaction on the connector](#)
- [How to create diagram connectors using drawing tools](#)

Group in Angular Diagram component

Group is used to cluster multiple nodes and connectors into a single element. It acts like a container for its children (nodes, groups, and connectors). Every change made to the group also affects the children. Child elements can be edited individually.

Create group

Add group when initializing diagram

A group can be added to the diagram model through [nodes](#) collection. To define an object as group, add the child objects to the [children](#) collection of the group.

- The [padding](#) property of a group node defines the spacing between the group node's edges and its children.
- While creating group, its child node need to be declared before the group declaration.
- Add a node to the existing group child by using the `diagram.group` method.
- The group's [Link to the Video](#) method is used to define whether the group can be ungrouped or not.
- A group can be added into a child of another group.

To create a group using Nodes and Connectors in the Angular Diagram, refer to the below video link,

The following code illustrates how to create a group node.

APP.COMPONENT.TS

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

```

import { Component, ViewEncapsulation, OnInit, ViewChild } from
'@angular/core';
import { DiagramComponent, Diagram, NodeModel, MarginModel, ShapeStyleModel
} from '@syncfusion/ej2-angular-diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%"
height="580px" [getNodeDefaults] = 'getNodeDefaults'
(created)='created($event)'>
    <e-nodes>
      <e-node id='node1' [offsetX]=100 [offsetY]=100>
      </e-node>
      <e-node id='node2' [offsetX]=200 [offsetY]=200>
      </e-node>
      <e-node id='node3' [offsetX]=400 [offsetY]=300 >
      </e-node>
      <e-node id='group' [children]='children' [padding]="padding">
      </e-node>
    </e-nodes>
  </ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram?: DiagramComponent;
  public getNodeDefaults(node: NodeModel): NodeModel {
    node.height = 100;
    node.width = 100;
    ((node as NodeModel).style as ShapeStyleModel).fill = "#6BA5D7";
    ((node as NodeModel).style as ShapeStyleModel).strokeColor =
    "White";
    return node;
  }
  public children?: string[];
  public padding: MarginModel = {left:10,right:10,top:10,bottom:10}
  ngOnInit(): void {
    this.children = ['node1', 'node2']
  }
  public created(args: Object): void {
    (this.diagram as DiagramComponent).selectAll();
    // Adding the third node into the existing group
    (this.diagram as DiagramComponent).group();
  }
}

```

MAIN.TS

```

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

```

The following code illustrates how a `ungroup` at runtime.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, OnInit, ViewChild } from
 '@angular/core';
import { DiagramComponent, Diagram, NodeModel, ShapeStyleModel } from
 '@syncfusion/ej2-angular-diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%"
height="580px" [getNodeDefaults] ='getNodeDefaults'
(created)='created($event)'>
    <e-nodes>
      <e-node id='node1' [offsetX]=100 [offsetY]=100>
      </e-node>
      <e-node id='node2' [offsetX]=200 [offsetY]=200>
      </e-node>
      <e-node id='node3' [offsetX]=400 [offsetY]=300 >
      </e-node>
      <e-node id='group' [children]='children'>
      </e-node>
    </e-nodes>
  </ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram?: DiagramComponent;
  public getNodeDefaults(node: NodeModel): NodeModel {
    node.height = 100;
    node.width = 100;
    ((node as NodeModel).style as ShapeStyleModel).fill = "#6BA5D7";
    ((node as NodeModel).style as ShapeStyleModel).strokeColor =
    "White";
    return node;
  }
  public children?: string[];
  ngOnInit(): void {
    this.children = ['node1', 'node2']
  }
  public created(args: Object): void {
    (this.diagram as DiagramComponent).selectAll();
    // Ungroup the selected group into nodes
    (this.diagram as DiagramComponent).unGroup();
  }
}
```


MAIN.TS

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

Connectors can be added to a group. The following code illustrates how to add connectors into a group.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, ViewChild, } from '@angular/core';
import {
    DiagramComponent, NodeModel, ConnectorModel,
} from '@syncfusion/ej2-angular-diagrams';
@Component({
    imports: [
        DiagramModule
    ],
    providers: [ ],
    standalone: true,
    selector: 'app-container',
    template: `<ejs-diagram #diagram id="diagram" width="1000px"
height="700px" [nodes]='nodes' [connectors]="connectors">
</ejs-diagram>`,
    encapsulation: ViewEncapsulation.None
})
export class AppComponent {
    @ViewChild('diagram')
    public diagram?: DiagramComponent;
    public connectors: ConnectorModel[] = [{
        id: 'connector1', type: 'Orthogonal', sourceID: 'node1', targetID:
'node2'
    },
    ];
    public nodes: NodeModel[] = [{
        id: 'node1', height: 100, width: 100, offsetX: 100, offsetY: 100,
        annotations: [{ content: 'Node1' }]
    },
    {
        id: 'node2', height: 100, width: 100, offsetX: 300, offsetY: 100,
        annotations: [{ content: 'Node2' }]
    },
    {
        id: 'group', children: ['node1', 'node2', 'connector1'], style: {
strokeWidth: 0}
    }
    ];
```

MAIN.TS

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

Add group at runtime

A group node can be added at runtime by using the client-side method `diagram.add`.

The following code illustrates how a group node is added at runtime.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, OnInit, ViewChild } from
 '@angular/core';
import { DiagramComponent, Diagram, NodeModel, ShapeStyleModel } from
 '@syncfusion/ej2-angular-diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%"
height="580px" [getNodeDefaults] ='getNodeDefaults'
(created)='created($event)'>
    <e-nodes>
      <e-node id='node1' [offsetX]=100 [offsetY]=100>
      </e-node>
      <e-node id='node2' [offsetX]=200 [offsetY]=200>
      </e-node>
    </e-nodes>
  </ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram?: DiagramComponent;
  public getNodeDefaults(node: NodeModel): NodeModel {
    node.height = 100;
    node.width = 100;
    ((node as NodeModel).style as ShapeStyleModel).fill = "#6BA5D7";
    ((node as NodeModel).style as ShapeStyleModel).strokeColor =
    "White";
    return node;
  }
  public group: NodeModel[] = [{
    id: 'group2',
    children: ['node1', 'node2']
  }]
  public created(args: Object): void {
    // Add the group into the diagram
    (this.diagram as DiagramComponent).add(this.group as any);
```

```
}
}
```

MAIN.TS

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

Add children To group at runtime

A childNode can be added to the specified Group at runtime by utilizing the client-side method `diagram.addChildToGroup`.

This functionality is achieved by passing the group and existing children as arguments to the method.

The following code illustrates how a child node and a group node can be passed as arguments to the method and executed at runtime.

```
`html
```

```
this.diagram.addChildToGroup(groupNode, childNode);
```

```
,
```

Remove children from group at runtime

A specific child from a group node can be removed at runtime by utilizing the client-side method `diagram.removeChildFromGroup`.

This functionality is achieved by passing the group and its children as arguments to the method.

The following code illustrates how a child node is removed from a group at runtime.

```
`html
```

```
this.diagram.removeChildFromGroup (groupNode, childNode);
```

```
,
```

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, OnInit, ViewChild } from
 '@angular/core';
import { DiagramComponent, Diagram, NodeModel } from '@syncfusion/ej2-
angular-diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  template: `
    <button (click)="addChild()">addChild</button>
    <button (click)="removeChild()">removeChild</button>
```

```

    <ejs-diagram #diagram id="diagram" width="100%" height="600"
[nodes]="nodes" >
    </ejs-diagram>`,
    encapsulation: ViewEncapsulation.None
  })
  export class AppComponent {
    @ViewChild("diagram")
    public diagram?: DiagramComponent;
    public nodes?: NodeModel[];
    addChild() {
      (this.diagram as any).addChildToGroup(
        (this.diagram as any).nodes[2],
        'node3'
      );
    }
    removeChild() {
      (this.diagram as any).removeChildFromGroup(
        (this.diagram as any).nodes[2],
        'node3'
      );
    }

    ngOnInit(): void {
      this.nodes = [
        {
          id: 'node1', width: 150, height: 100, offsetX: 100, offsetY:
100, annotations: [{ content: 'Node1' }]
        }, {
          id: 'node2', width: 80, height: 130, offsetX: 200, offsetY:
200, annotations: [{ content: 'Node2' }]
        }, {
          id: 'group1', children: ['node1', 'node2']
        }, {
          id: 'node3', width: 100, height: 100, offsetX: 300, offsetY:
300, annotations: [{ content: 'Node3' }]
        }
      ];
    }
  }
}

```

MAIN.TS

```

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

```

Container

Containers are used to automatically measure and arrange the size and position of the child elements in a predefined manner.

There are two types of containers available.

- Canvas

- Stack

Canvas

- The canvas panel supports absolute positioning and provides the least layout functionality to its contained diagram elements.
- Canvas allows you to position its contained elements by using the margin and alignment properties.
- Rendering alone possible in canvas container.
- It allows elements to be either vertically or horizontally aligned.
- Child can be defined with the collection [canvas.children](#) property.
- Basic element can be defined with the collection of [basicElements](#).

Stack

- Stack panel is used to arrange its children in a single line or stack order, either vertically or horizontally.
- It controls spacing by setting margin properties of child and padding properties of group. By default, a stack panel's [orientation](#) is vertical.

The following code illustrates how to add a stack panel.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, OnInit, ViewChild } from '@angular/core';
import { DiagramComponent, Diagram, NodeModel, TextElement, StackPanel,
PointModel, VerticalAlignment, ShapeStyleModel } from '@syncfusion/ej2-
angular-diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%"
height="580px" [getNodeDefaults] ='getNodeDefaults'>
  <e-nodes>
    <e-node id='node1' [offsetX]=100 [offsetY]=100>
      <e-node-annotations>
        <e-node-annotation content="Custom Template"
[offset]='offset' [verticalAlignment]='verticalAlignment'>
      </e-node-annotation>
    </e-node-annotations>
    </e-node>
  </e-nodes>
</ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
```

```

export class AppComponent {
  @ViewChild("diagram")
  public diagram?: DiagramComponent;
  public offset?: PointModel
  public verticalAlignment?: VerticalAlignment
  public getTextElement(text: string): TextElement {
    let textElement: TextElement = new TextElement();
    textElement.width = 50;
    textElement.height = 20;
    textElement.content = text;
    return textElement;
  };
  public addRows(column: StackPanel) {
    column.children.push(this.getTextElement('Row1'));
    column.children.push(this.getTextElement('Row2'));
    column.children.push(this.getTextElement('Row3'));
    column.children.push(this.getTextElement('Row4'));
  };
  public getNodeDefaults(node: NodeModel): NodeModel {
    node.height = 100;
    node.width = 100;
    ((node as NodeModel).style as ShapeStyleModel).fill = '#6BA5D7';
    ((node as NodeModel).style as ShapeStyleModel).strokeColor =
'white';
    return node;
  }
  ngOnInit(): void {
    this.verticalAlignment = 'Top';
    this.offset = {y: 1};
    (this.diagram as DiagramComponent).setNodeTemplate = (obj: NodeModel
| any, diagram: Diagram): StackPanel | any => {
      if (obj.id.indexOf('node1') !== -1) {
        // It will be replaced with grid panel
        let table: StackPanel = new StackPanel();
        table.orientation = 'Horizontal';
        let column1: StackPanel = new StackPanel();
        column1.children = [];
        column1.children.push(this.getTextElement('Column1'));
        this.addRows(column1);
        let column2: StackPanel = new StackPanel();
        column2.children = [];
        column2.children.push(this.getTextElement('Column2'));
        this.addRows(column2);
        table.children = [column1, column2];
        return table;
      }
      return undefined;
    }
  }
}

```

MAIN.TS

```

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

```

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

Difference between a basic group and containers

| Group | Container |

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

| It arranges the child elements based on the child elements position and size properties. | Each container has a predefined behavior to measure and arrange its child elements. Canvas and stack containers are supported in the diagram. |

| The Padding, Min, and Max Size properties are not applicable for basic group. | It is applicable for container. |

| The Children's margin and alignment properties are not applicable for basic group. | It is applicable for container. |

Interaction

You can edit the group and its children at runtime. For more information about how to interact with a group, refer to [Edit Groups](#).

See Also

- [How to add annotations to the node](#)
- [How to add ports to the node](#)
- [How to enable/disable the behavior of the node](#)
- [How to add nodes to the symbol palette](#)
- [How to create diagram nodes using drawing tools](#)
- [How to perform the interaction on the group](#)

Swim lane in Angular Diagram component

Swimlane is a type of diagram nodes, which is typically used to visualize the relationship between a business process and the department responsible for it by focusing on the logical relationships between activities.

Create a swimlane

To create a swimlane, the type of shape should be set as [swimlane](#). By Default swimlane's are arranged vertically.

The following code example illustrates how to define a swimlane object.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import { DiagramComponent, SwimLaneModel, Diagram, NodeModel, Node,
LaneModel, HeaderModel } from '@syncfusion/ej2-angular-diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [ ],
```

```

standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%"
height="580px" [nodes]="nodes">
  </ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  public nodes: NodeModel[] = [
    {
      shape: {
        type: 'SwimLane',
        lanes: [
          {
            id: 'stackCanvas1',
            height: 100,
          },
        ],
        phases: [{
          id: 'phase1', offset: 170,
          header: { annotation: { content: 'Phase' } }
        }
        ],
        phaseSize: 20,
      },
      offsetX: 300, offsetY: 200,
      height: 200,
      width: 350
    },
  ],
  @ViewChild("diagram")
  public diagram?: DiagramComponent;
}

```

MAIN.TS

```

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

```

Headers

Header was the primary element for swimlanes. The [header](#) property of swimlane allows you to define its textual description and to customize its appearance.

Note: By using this header, the swimlane interaction will be performed, like selection, dragging, etc.

The following code example illustrates how to define a swimlane header.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';

```



```

import { DiagramComponent, SwimLaneModel, Diagram, NodeModel, Node,
LaneModel, HeaderModel } from '@syncfusion/ej2-angular-diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%"
height="580px" [nodes]="nodes">
</ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  public nodes: NodeModel[] = [
    {
      shape: {
        type: 'SwimLane',
        orientation: 'Horizontal',
        // Intialize header to swimlane
        header: {
          annotation: { content: 'ONLINE PURCHASE STATUS', style:
{ fill: '#111111' } },
          height: 50, style: { fontSize: 11 },
        },
        lanes: [
          {
            id: 'stackCanvas1',
            height: 100,
          },
        ],
        phases: [{
          id: 'phase1', offset: 170,
          header: { annotation: { content: 'Phase' } }
        }
        ],
        phaseSize: 20,
      },
      offsetX: 300, offsetY: 200,
      height: 200,
      width: 350
    },
  ],
  @ViewChild("diagram")
  public diagram?: DiagramComponent;
}

```

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 headers

The height and width of swimlane header can be customized with [weight](#) and [height](#) properties of swimlane header. set fill color of header by using the [style](#) property. The orientation of swimlane can be customized with the [orientation](#) property of the header.

Note: By default the swimlane orientation has Horizontal.

The following code example illustrates how to customize the swimlane header.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import { DiagramComponent, SwimLaneModel, Diagram, NodeModel, Node,
LaneModel, HeaderModel } from '@syncfusion/ej2-angular-diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%"
height="580px" [nodes]="nodes">
  </ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  public nodes: NodeModel[] = [
    {
      shape: {
        type: 'SwimLane',
        orientation: 'Horizontal',
        // customize the swimlane header
        header: {
          annotation: { content: 'SALES PROCESS FLOW CHART', },
          height: 70, style: { fontSize: 11, fill: 'pink' }
        },
        lanes: [
          {
            id: 'stackCanvas1',
            height: 100,
          },
        ],
        phases: [{
          id: 'phase1', offset: 170,
          header: { annotation: { content: 'Phase' } }
        }
      ],
      phaseSize: 20,
    },
    {
      offsetX: 420, offsetY: 270,
      height: 100,
      width: 650
    },
  ],
```

```

    ]
    @ViewChild("diagram")
    public diagram?: DiagramComponent;
  }

```

MAIN.TS

```

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

```

Dynamic customization of swimlane header

You can customize the swimlane header style and text properties dynamically. The following code illustrates how to dynamically customize the lane header.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import { DiagramComponent, SwimLaneModel, Diagram, NodeModel, Node,
LaneModel, HeaderModel, ShapeModel } from '@syncfusion/ej2-angular-diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%"
height="580px" [nodes]="nodes" (created)="created($event)">
</ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  public nodes: NodeModel[] = [
    {
      shape: {
        type: 'SwimLane',
        orientation: 'Horizontal',
        // customize the swimlane header
        header: {
          annotation: { content: 'SALES PROCESS FLOW CHART', },
          height: 70, style: { fontSize: 11, fill: 'pink' }
        },
        lanes: [
          {
            id: 'stackCanvas1',
            height: 100,
          },
        ],
        phases: [{
          id: 'phase1', offset: 170,

```

```

        header: { annotation: { content: 'Phase' } }
    },
    phaseSize: 20,
},
offsetX: 420, offsetY: 270,
height: 100,
width: 650
},
]
@ViewChild("diagram")
public diagram?: DiagramComponent;
public created(args: Object): void {
    ((this.diagram as Diagram).nodes[0].shape as ShapeModel |
any).header.style.fill = 'red';
    (this.diagram as Diagram).dataBind();
}
}

```

MAIN.TS

```

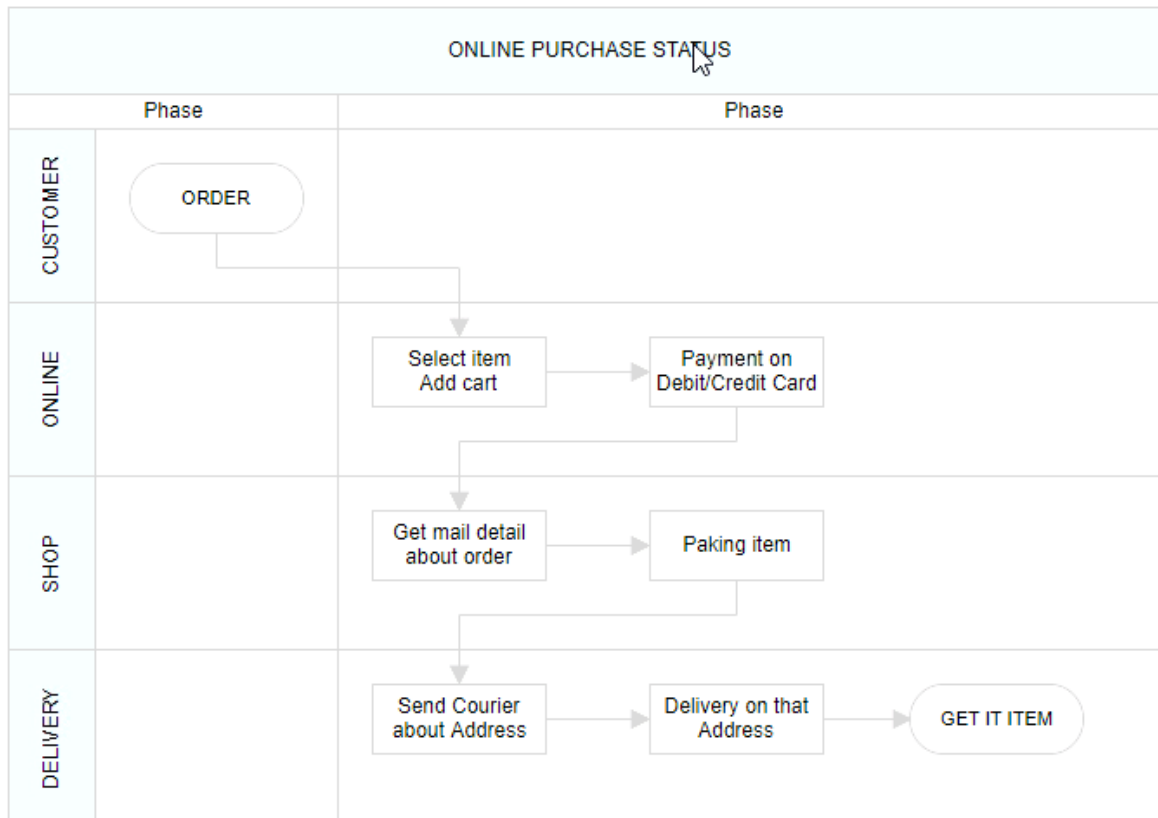
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Header editing

Diagram provides the support to edit swimlane headers at runtime. We achieve the header editing by double click event. Double clicking the header label will enables the editing of that.

The following image illustrates how to edit the swimlane header.



Lanes

Lane is a functional unit or a responsible department of a business process that helps to map a process within the functional unit or in between other functional units.

The number of [lanes](#) can be added to swimlane. The lanes are automatically stacked inside swimlane based on the order they are added.

Create an empty lane

- The lane `id` is used to define the name of the lane and its further used to find the lane at runtime and do any customization.
- We can provide additional information to the lane by using the [addInfo](#) property of the lane.

The following code example illustrates how to define a swimlane with lane.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import { DiagramComponent, SwimLaneModel, Diagram, NodeModel, Node,
LaneModel, HeaderModel } from '@syncfusion/ej2-angular-diagrams';
@Component({
  imports: [
    DiagramModule
  ],

```

```

providers: [ ],
standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%"
height="580px" [nodes]="nodes">
  </ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  public nodes: NodeModel[] = [
    {
      shape: {
        type: 'SwimLane',
        orientation: 'Horizontal',
        header: {
          annotation: { content: 'ONLINE PURCHASE STATUS', style:
{ fill: 'pink' } },
          height: 50, style: { fontSize: 11 },
        },
        // initialize the lane of swimlane
        lanes: [
          {
            id: 'stackCanvas1',
            // set the lane height
            height: 100,
            // set the lane info
            addInfo:{name:'lane1'}
          },
        ],
        phases: [
          {
            id: 'phase1', offset: 170,
            header: { annotation: { content: 'Phase' } }
          }
        ],
        phaseSize: 20,
      },
      offsetX: 300, offsetY: 200,
      height: 200,
      width: 350
    },
  ]
  @ViewChild("diagram")
  public diagram?: DiagramComponent;
}

```

MAIN.TS

```

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

```

Create lane header

- The [header](#) property of lane allows you to textually describe the lane and to customize the appearance of the description.

The following code example illustrates how to define a lane header.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import { DiagramComponent, SwimLaneModel, Diagram, NodeModel, Node,
LaneModel, HeaderModel } from '@syncfusion/ej2-angular-diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%"
height="580px" [nodes]="nodes">
  </ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  public nodes: NodeModel[] = [
    {
      id: 'swimlane',
      shape: {
        type: 'SwimLane',
        orientation: 'Horizontal',
        // Intialize header to swimlane
        header: {
          annotation: { content: 'ONLINE PURCHASE STATUS', style:
{ fill: 'pink' } },
          height: 50, style: { fontSize: 11 },
        },
        // Intialize lane to swimlane
        lanes: [
          {
            id: 'stackCanvas1',
            height: 100,
            // Intialize header to lane
            header: {
              annotation: { content: 'CUSTOMER' }, width: 50,
              style: { fontSize: 11 }
            },
          },
        ],
      },
      phases: [{
        id: 'phase1', offset: 170,
        header: { annotation: { content: 'Phase' } }
      }
    ]
  ]
}
```

```

        ],
        phaseSize: 20,
    },
    offsetX: 420, offsetY: 270,
    height: 100,
    width: 650
  },
]
@ViewChild("diagram")
public diagram?: DiagramComponent;
}

```

MAIN.TS

```

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

```

Customizing lane header

- The size of lane can be controlled by using [width](#) and [height](#) properties of lane.
- The appearance of lane can be set by using the [style](#) properties.

The following code example illustrates how to customize the lane header.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import { DiagramComponent, SwimLaneModel, Diagram, NodeModel, Node,
LaneModel, HeaderModel } from '@syncfusion/ej2-angular-diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%"
height="580px" [nodes]="nodes">
</ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  public nodes: NodeModel[] = [
    {
      shape: {
        type: 'SwimLane',
        orientation: 'Horizontal',
        //Intialize header to swimlane
        header: {

```



```

        annotation: { content: 'ONLINE PURCHASE STATUS', style:
{ fill: 'pink' } },
        height: 50, style: { fontSize: 11 },
    },
    lanes: [
        {
            id: 'stackCanvas1',
            height: 100,
            // customization of lane header
            header: {
                annotation: { content: 'Online Consumer' },
width: 30,
                style: { fontSize: 11, fill: 'red' }
            }
        },
    ],
    phases: [{
        id: 'phase1', offset: 170,
        header: { annotation: { content: 'Phase' } }
    }
    ],
    phaseSize: 20,
    },
    offsetX: 300, offsetY: 200,
    height: 200,
    width: 350
    },
    ],
    @ViewChild("diagram")
    public diagram?: DiagramComponent;
}

```

MAIN.TS

```

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

```

Dynamic customization of lane header

We can customize the lane header style and text properties dynamically. The following code illustrates how to dynamically customize the lane header..

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import { DiagramComponent, SwimLaneModel, Diagram, NodeModel, Node,
LaneModel, HeaderModel } from '@syncfusion/ej2-angular-diagrams';
@Component({
  imports: [
    DiagramModule
  ],

```

```

providers: [ ],
standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%"
height="580px" [nodes]="nodes" (created)='created($event)'>
  </ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  public nodes: NodeModel[] = [
    {
      shape: {
        type: 'SwimLane',
        orientation: 'Horizontal',
        //Intialize header to swimlane
        header: {
          annotation: { content: 'ONLINE PURCHASE STATUS', style:
{ fill: 'pink' } },
          height: 50, style: { fontSize: 11 },
        },
        lanes: [
          {
            id: 'stackCanvas1',
            height: 100,
            // customization of lane header
            header: {
              annotation: { content: 'Online Consumer' }, width:
30,
              style: { fontSize: 11, fill: 'red' }
            },
          },
        ],
        phases: [{
          id: 'phasel', offset: 170,
          header: { annotation: { content: 'Phase' } }
        }
        ],
        phaseSize: 20,
      },
      offsetX: 300, offsetY: 200,
      height: 200,
      width: 350
    },
  ]
  @ViewChild("diagram")
  public diagram?: DiagramComponent;
  public created(args: Object): void {
    // Update the connector properties at the run time
    ((this.diagram as Diagram).nodes[0].shape as
any).lanes[0].header.style.fill = 'blue';
    (this.diagram as Diagram).dataBind();
  }
}

```

MAIN.TS

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

Add lane at runtime

You can add the a lane at runtime by using the client side API method called `addLanes`. The following code illustrates how to dynamically add lane to swimlane.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import { DiagramComponent, SwimLaneModel, Diagram, NodeModel, Node,
LaneModel, HeaderModel } from '@syncfusion/ej2-angular-diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%"
height="580px" [nodes]="nodes" (created)="created($event)">
</ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  public nodes: NodeModel[] = [
    {
      shape: {
        type: 'SwimLane',
        orientation: 'Horizontal',
        //Intialize header to swimlane
        header: {
          annotation: { content: 'ONLINE PURCHASE STATUS', style:
{ fill: 'pink' } },
          height: 50, style: { fontSize: 11 },
        },
        lanes: [
          {
            id: 'stackCanvas1',
            height: 100,
            // customization of lane header
            header: {
              annotation: { content: 'Online Consumer' },
              style: { fontSize: 11, fill: 'red' },
            },
          },
        ],
        phases: [{
          id: 'phase1', offset: 170,
          header: { annotation: { content: 'Phase' } }
        }
      ],
      width: 30,
    }
  ]
}
```

```

        },
        phaseSize: 20,
    },
    offsetX: 300, offsetY: 200,
    height: 200,
    width: 350
  },
]
@ViewChild("diagram")
public diagram?: DiagramComponent;
public created(args: Object): void {
  let lane = [{id:"lane1",height:100,}];
  (this.diagram as Diagram).addLanes((this.diagram as
Diagram).nodes[0],lane,1);
  (this.diagram as Diagram).dataBind();
}
}

```

MAIN.TS

```

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

```

Add children to lane

To add nodes to lane,you should add [children](#) collection of the lane.

The following code example illustrates how to add nodes to lane.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import { DiagramComponent, SwimLaneModel,Diagram, NodeModel,Node,
LaneModel,HeaderModel } from '@syncfusion/ej2-angular-diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%"
height="580px" [nodes]="nodes">
</ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  public nodes: NodeModel[] = [
    {
      id: 'swimlane',
      orientation: 'Horizontal' as any,

```

```

        shape: {
            type: 'SwimLane',
            header: {
                annotation: { content: 'ONLINE PURCHASE STATUS', style: {
fill: 'pink' } } },
                height: 50, style: { fontSize: 11 },
            },
            lanes: [
                {
                    id: 'stackCanvas1',
                    height: 100,
                    header: {
                        annotation: { content: 'CUSTOMER' }, width: 50,
                        style: { fontSize: 11 }
                    },
                    // Set the children of lane
                    children: [
                        {
                            id: 'node1',
                            annotations: [
                                {
                                    content: 'Consumer learns \n of product',
                                    style: { fontSize: 11 }
                                }
                            ],
                            margin: { left: 60, top: 30 },
                            height: 40, width: 100,
                        }, {
                            id: 'node2',
                            shape: { type: 'Flow', shape: 'Decision' },
                            annotations: [
                                {
                                    content: 'Does \n Consumer want \nthe product',
                                    style: { fontSize: 11 }
                                }
                            ],
                            margin: { left: 200, top: 20 },
                            height: 60, width: 120,
                        },
                    ],
                },
            ],
            phases: [{
                id: 'phase1', offset: 170,
                header: { annotation: { content: 'Phase' } }
            }
        ],
        phaseSize: 20,
    },
    offsetX: 420, offsetY: 270,
    height: 100,
    width: 650
}
] as unknown as NodeModel[]
@ViewChild("diagram")
public diagram?: DiagramComponent;
}

```

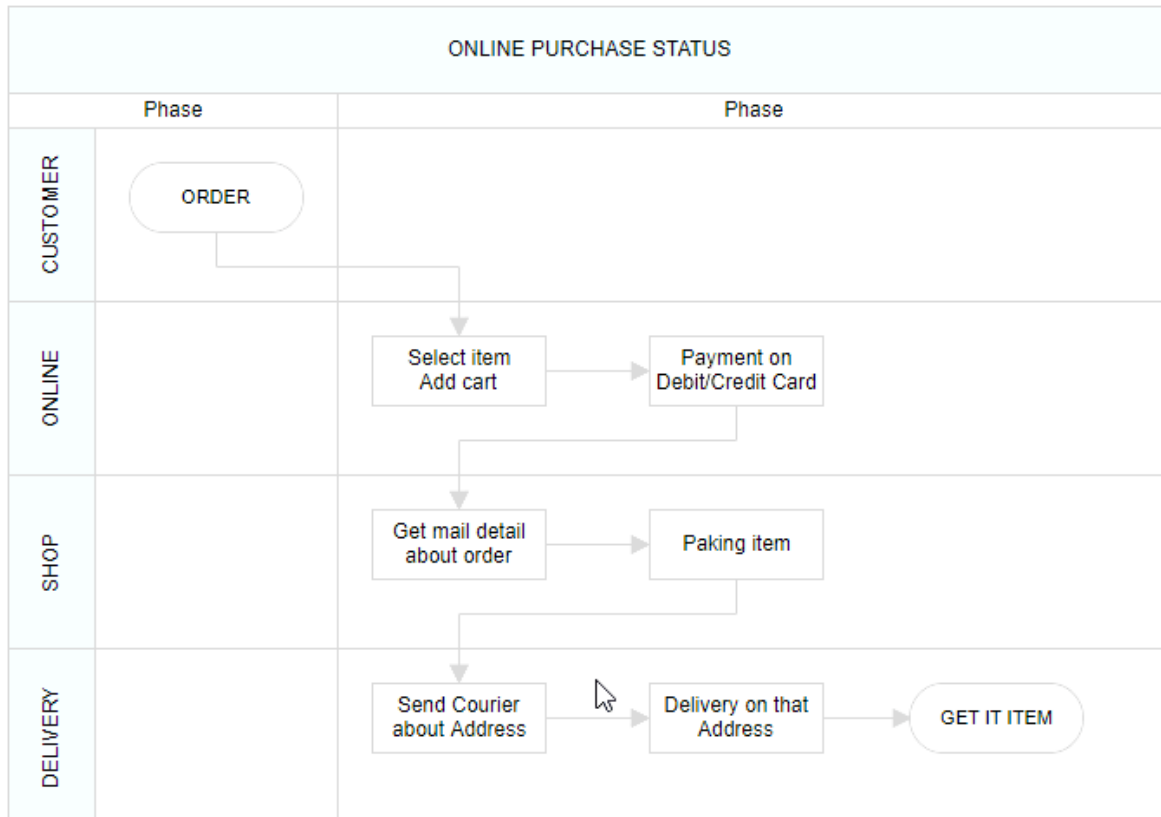
MAIN.TS

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

*Lane interaction**Resizing lane*

- Lane can be resized in the bottom and left direction.
- Lane can be resized by using resize selector of the lane.
- Once you can resize the lane, the swimlane will be resized automatically.
- The lane can be resized either resizing the selector or the tight bounds of the child object. If the child node moves to the edge of the lane, it can be automatically resized.

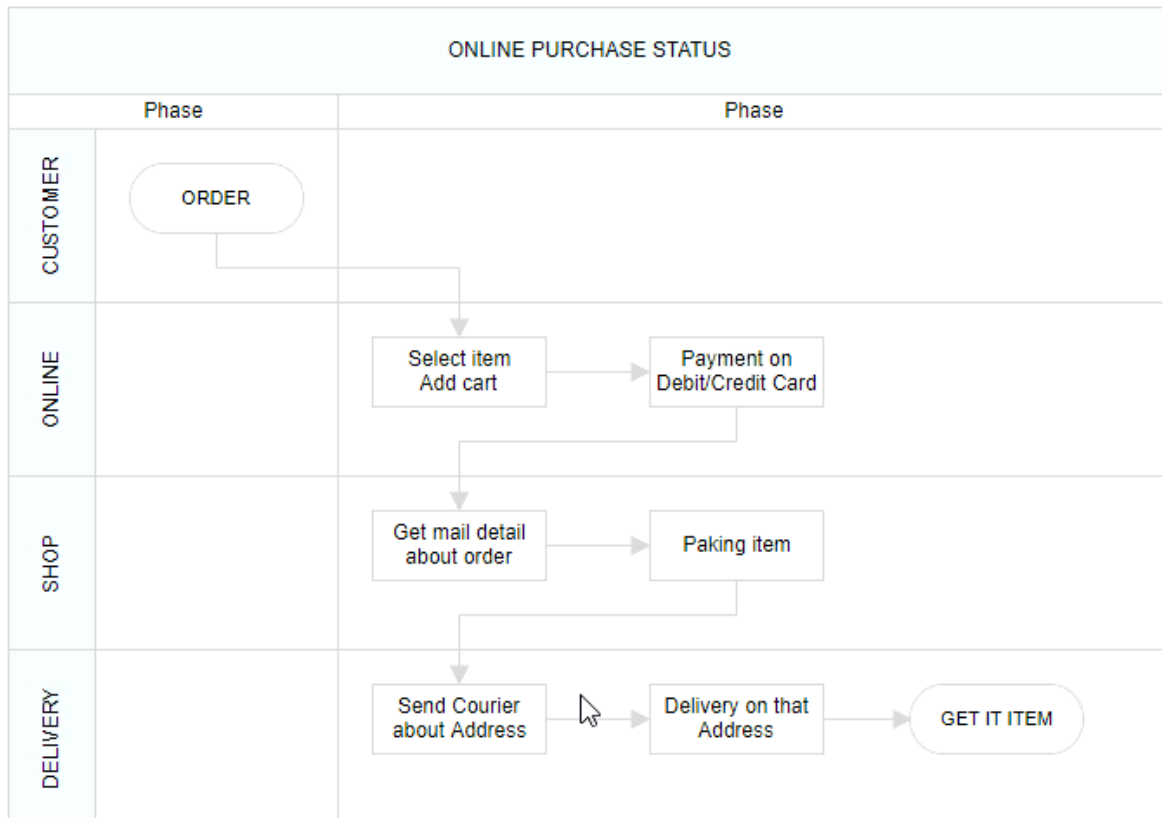
The following image illustrates how to resize the lane.



Lane swapping

- Lanes can be swapped using drag the lanes over another lane.
- Helper should intimate the insertion point while lane swapping.

The following image illustrates how swapping the lane.



Disable Swimlane Lane swapping

You can disable swimlane lane swapping by using the property called `canMove`.

The following code illustrates how to disable swimlane lane swapping.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import { DiagramComponent, SwimLaneModel, Diagram, NodeModel, Node,
LaneModel, HeaderModel } from '@syncfusion/ej2-angular-diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%"
height="580px" [nodes]="nodes" (created)="created($event)">
</ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  public nodes: NodeModel[] = [
    {

```



```

        shape: {
            type: 'SwimLane',
            orientation: 'Horizontal',
            //Intialize header to swimlane
            header: {
                annotation: { content: 'ONLINE PURCHASE STATUS', style:
{ fill: 'pink' } },
                height: 50, style: { fontSize: 11 },
            },
            lanes: [
                {
                    id: 'stackCanvas1',
                    height: 100,
                    // customization of lane header
                    header: {
                        annotation: { content: 'Online Consumer' }, width:
30,
                        style: { fontSize: 11, fill: 'red' }
                    },
                    canMove: false ,
                },
            ],
            phases: [{
                id: 'phase1', offset: 170,
                header: { annotation: { content: 'Phase' } }
            }
            ],
            phaseSize: 20,
        },
        offsetX: 300, offsetY: 200,
        height: 200,
        width: 350
    },
]
@ViewChild("diagram")
public diagram?: DiagramComponent;
public created(args: Object): void {
    let lane = [{id:"lane1",height:100,canMove: false}];
    (this.diagram as Diagram).addLanes((this.diagram as
Diagram).nodes[0],lane,1);
    (this.diagram as Diagram).dataBind();
}
}

```

MAIN.TS

```

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

```

Resize helper

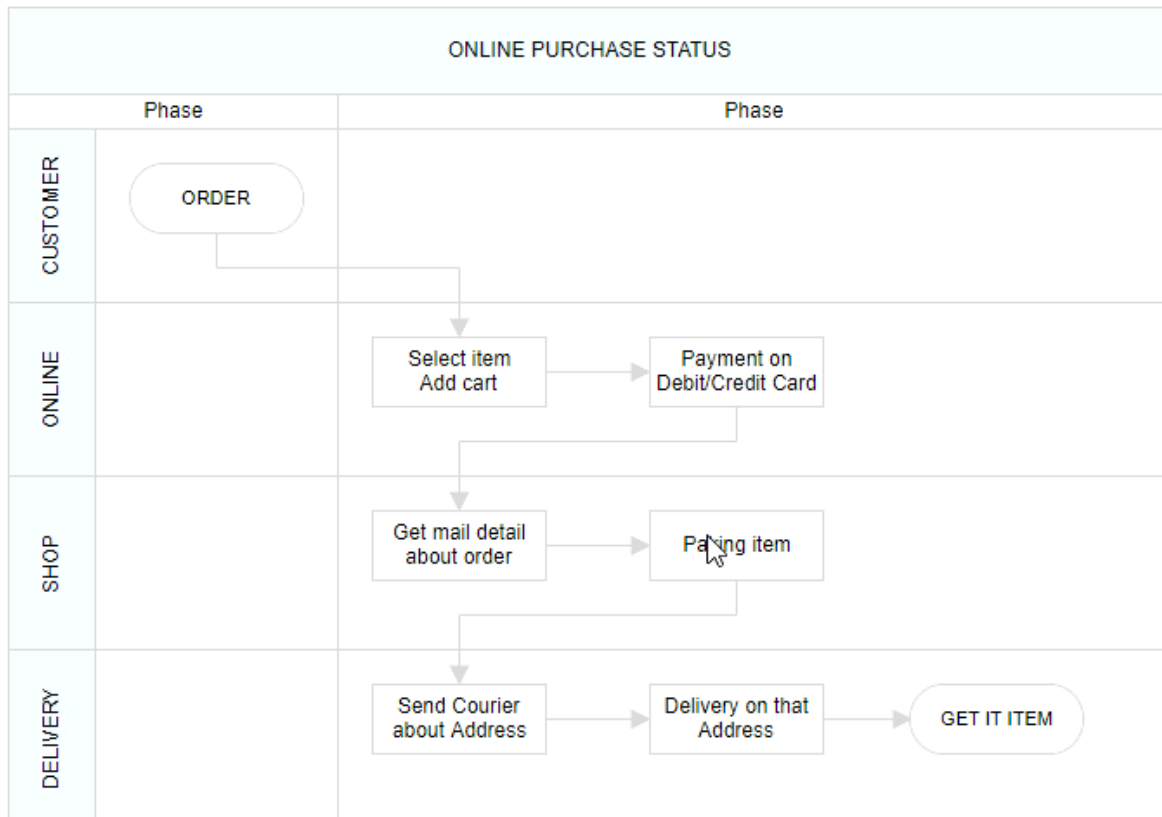
- The special resize helper will be used to resize the lanes.
- The resize cursor will be available on the left and bottom direction alone.

- Once resize the lane the swimlane will be resized automatically.

Children interaction in lanes

- You can resize the child node within swimlanes.
- You can drag the child nodes within lane.
- Interchange the child nodes from one lane to another lane.
- Drag and drop the child nodes from lane to diagram.
- Drag and drop the child nodes from diagram to lane.
- Based on the child node interactions,the lane size should be updated.

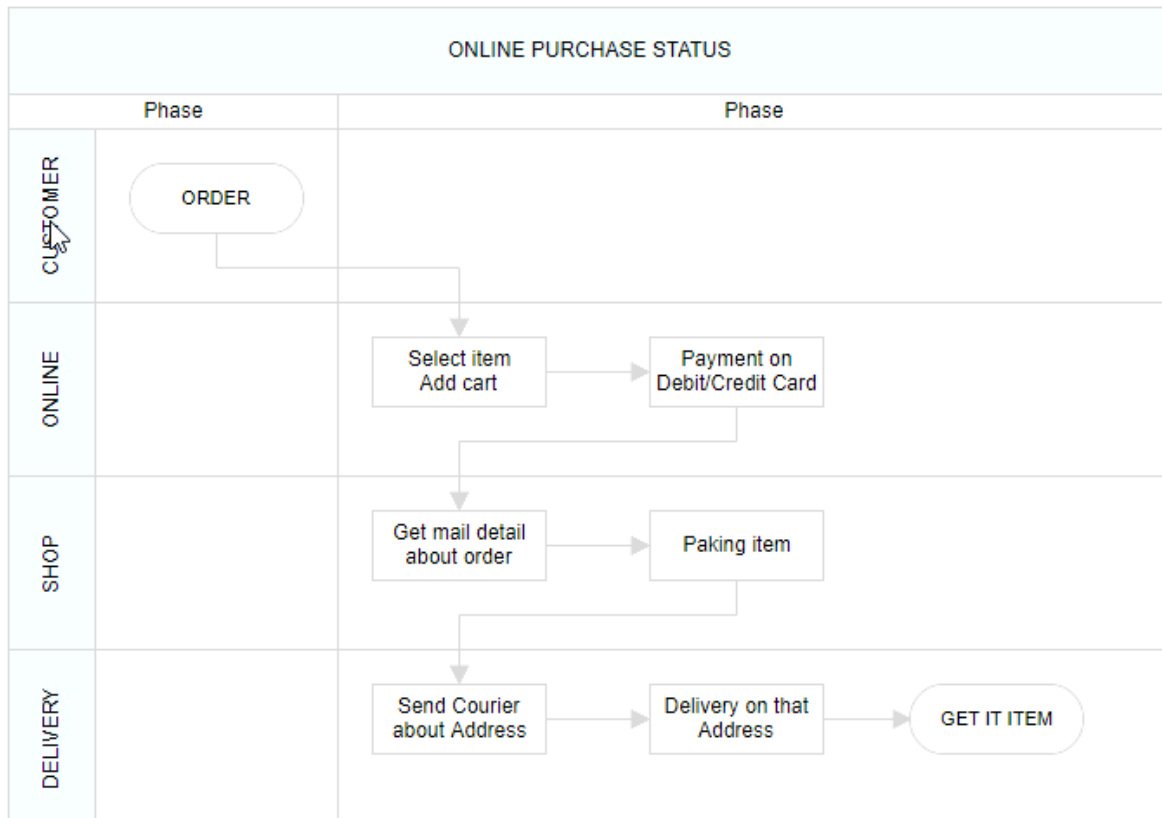
The following image illustrates children interaction in lane.



Lane header editing

Diagram provides the support to edit Lane headers at runtime. We achieve the header editing by double click event. Double clicking the header label will enables the editing of that.

The following image illustrates how to edit the lane header.



Phase

Phase are the subprocess which will split each lanes as horizontally or vertically based on the swimlane orientation. The multiple number of [Phase](#) can be added to swimlane.

The following code example illustrates how to add phase at swimlane.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import { DiagramComponent, SwimLaneModel, Diagram, NodeModel, Node,
LaneModel, HeaderModel } from '@syncfusion/ej2-angular-diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%"
height="580px" [nodes]="nodes">
</ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  public nodes: NodeModel[] = [

```

```

    {
      id: 'swimlane',
      shape: {
        type: 'SwimLane',
        orientation: 'Horizontal',
        header: {
          annotation: { content: 'ONLINE PURCHASE STATUS', style:
{ fill: 'pink' } },
          height: 50, style: { fontSize: 11 },
        },
        lanes: [
          {
            id: 'stackCanvas1',
            height: 100,
            header: {
              annotation: { content: 'CUSTOMER' }, width: 50,
              style: { fontSize: 11 }
            },
            children: [
              {
                id: 'Order',
                margin: { left: 60, top: 20 },
                height: 40, width: 100
              }
            ]
          },
        ],
      },
      // Set phase to swimlane
      phases: [
        {
          id: 'phase1', offset: 120,
          header: { annotation: { content: 'Phase' } }
        }, {
          id: 'phase2', offset: 200,
          header: { annotation: { content: 'Phase' } }
        },
      ],
      phaseSize: 20,
    },
    offsetX: 420, offsetY: 270,
    height: 100,
    width: 650
  ],
  @ViewChild("diagram")
  public diagram?: DiagramComponent;
}

```

MAIN.TS

```

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

```

Dynamically add phase to lane

You can add the a phase at runtime by using client side API method called `addPhases`. The following code example illustrates how to add phase at run time.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import { DiagramComponent, SwimLaneModel, Diagram, NodeModel, Node,
LaneModel, HeaderModel } from '@syncfusion/ej2-angular-diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%"
height="580px" [nodes]="nodes" (created)='created($event)'>
</ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  public nodes: NodeModel[] = [
    {
      id: 'swimlane',
      shape: {
        type: 'SwimLane',
        orientation: 'Horizontal',
        header: {
          annotation: { content: 'ONLINE PURCHASE STATUS', style:
{ fill: 'pink' } },
          height: 50, style: { fontSize: 11 },
        },
        lanes: [
          {
            id: 'stackCanvas1',
            height: 100,
            header: {
              annotation: { content: 'CUSTOMER' }, width: 50,
              style: { fontSize: 11 }
            },
            children: [
              {
                id: 'Order',
                margin: { left: 60, top: 20 },
                height: 40, width: 100
              }
            ]
          },
        ],
      },
      // Set phase to swimlane
      phases: [
        {
          id: 'phase1', offset: 120,
```

```

        header: { annotation: { content: 'Phase' } }
      }, {
        id: 'phase2', offset: 200,
        header: { annotation: { content: 'Phase' } }
      },
    ],
    phaseSize: 20,
  },
  offsetX: 420, offsetY: 270,
  height: 100,
  width: 650
},
]
@ViewChild("diagram")
public diagram?: DiagramComponent;
public created(args: Object): void {
  let phase = [{
    id: 'phase3', offset: 220,
    header: { annotation: { content: 'Phase' } }
  }] as any
  (this.diagram as Diagram).addPhases((this.diagram as
Diagram).nodes[0], phase);
  (this.diagram as Diagram).dataBind();
}
}

```

MAIN.TS

```

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

```

Customizing phase

- The length of region can be set by using the [offset](#) property of the phase.
- Every phase region can be textually described with the [header](#) property of the phase.
- You can increase the width of phase by using [phaseSize](#) property of swimlane.
- We can provide additional information to the phase by using the [addInfo](#) property of the phase.

The following code example illustrates how to customize the phase in swimlane.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import { DiagramComponent, SwimLaneModel, Diagram, NodeModel, Node,
LaneModel, HeaderModel } from '@syncfusion/ej2-angular-diagrams';
@Component({
  imports: [
    DiagramModule
  ],

```

```

providers: [ ],
standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%"
height="580px" [nodes]="nodes">
  </ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  public nodes: NodeModel[] = [
    {
      id: 'swimlane',
      shape: {
        type: 'SwimLane',
        orientation: 'Horizontal',
        header: {
          annotation: { content: 'ONLINE PURCHASE STATUS', style:
{ fill: 'pink' } },
          height: 50, style: { fontSize: 11 },
        },
        lanes: [
          {
            id: 'stackCanvas1',
            height: 100,
            header: {
              annotation: { content: 'CUSTOMER' }, width: 50,
              style: { fontSize: 11 }
            },
            children: [
              {
                id: 'Order',
                margin: { left: 60, top: 20 },
                height: 40, width: 100
              }
            ]
          },
        ],
      },
      phases: [
        {
          id: 'phase1', offset: 120,
          // set the phase info
          addInfo:{name:'phase1'},
          header: { annotation: { content: 'Phase' } }
        },
        {
          id: 'phase2', offset: 200,
          header: { annotation: { content: 'Phase' } }
        },
      ],
      phaseSize: 20,
    },
    {
      offsetX: 420, offsetY: 270,
      height: 100,
      width: 650
    },
  ],
  @ViewChild("diagram")

```

```
public diagram?: DiagramComponent;  
}
```

MAIN.TS

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

Phase interaction

Resizing

- The phase can be resized by using its helper.
- You must select the phase header to enable the phase selection.
- Once the phase can be resized, the lane size will be updated automatically.

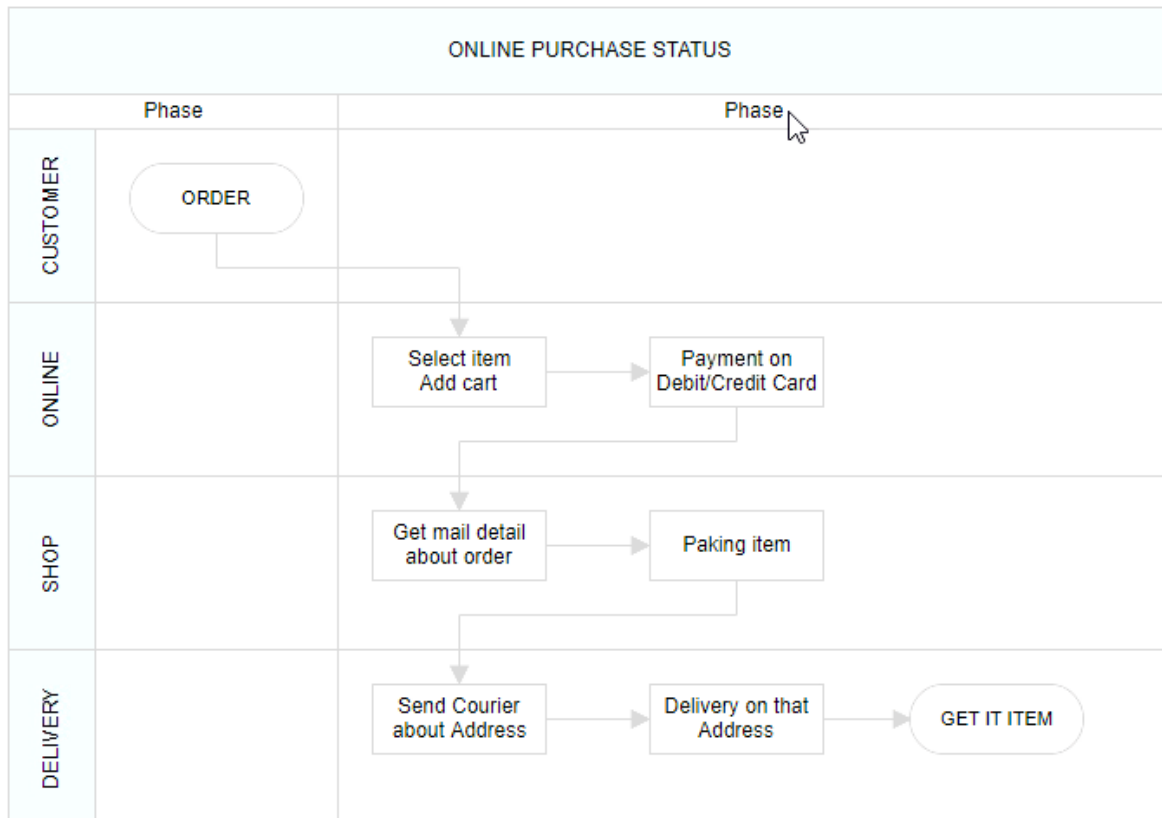
Resizing helper

- The special resize helper will be used to resize the phase.
- The resize cursor will be available on the left and bottom direction for horizontal, and the top and bottom direction for vertical swimlane.

Phase header editing

Diagram provides the support to edit phase headers at runtime. We achieve the header editing by double click event. Double clicking the header label will enables the editing of that.

The following image illustrates how to edit the swimlane header.



Add swimlane to palette

Diagram provides the support to add swimlane and phases to symbol palette. The following code sample illustrate how to add swimlane and phases to palette.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule, SymbolPaletteModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, OnInit, ViewEncapsulation, ViewChild } from '@angular/core';
import { SymbolPaletteComponent, SymbolPalette, SymbolPreviewModel,
NodeModel, ConnectorModel, PaletteModel, SwimLaneModel, LaneModel,
HeaderModel, MarginModel, ExpandTool, ShapeStyleModel } from '@syncfusion/ej2-angular-diagrams';
@Component({
imports: [
    DiagramModule, SymbolPaletteModule
],
providers: [],
standalone: true,
selector: "app-container",
template: `<ejs-symbolpalette id="symbolpalette"width="100%"
height="700px" [symbolHeight]=80 [symbolWidth]=80 [expandMode]="expandMode"
[palettes]="palettes" [getSymbolInfo]="getSymbolInfo"
[symbolMargin]="symbolMargin" [symbolPreview]="symbolPreview"
[getNodeDefaults]="">

```

```

</ejs-symbolpalette>`,
encapsulation: ViewEncapsulation.None
}))
export class AppComponent {
  public expandMode?: ExpandTool;
  public palettes?: PaletteModel[];
  public symbolMargin?: MarginModel;
  public symbolPreview?: SymbolPreviewModel;
  public getswimlaneShapes(): NodeModel[] {
    let swimlaneShapes : NodeModel[] = [
      {
        id: 'stackCanvas1',
        shape: {
          type: 'SwimLane', lanes: [
            {
              id: 'lane1',
              style: { strokeColor: 'black' }, height: 60,
width: 150,
              header: { width: 50, height: 50, style: {
strokeColor: 'black', fontSize: 11 } },
            }
          ],
          orientation: 'Horizontal', isLane: true
        },
        height: 60,
        width: 140,
        offsetX: 70,
        offsetY: 30,
      }, {
        id: 'stackCanvas2',
        shape: {
          type: 'SwimLane',
          lanes: [
            {
              id: 'lane1',
              style: { strokeColor: 'black' }, height: 150,
width: 60,
              header: { width: 50, height: 50, style: {
strokeColor: 'black', fontSize: 11 } },
            }
          ],
          orientation: 'Vertical', isLane: true
        },
        height: 140,
        width: 60,
        // style: { fill: '#f5f5f5' },
        offsetX: 70,
        offsetY: 30,
      }, {
        id: 'verticalPhase',
        shape: {
          type: 'SwimLane',
          phases: [{ style: { strokeWidth: 1, strokeDashArray:
'3,3', strokeColor: '#A9A9A9' }, }],
          annotations: [{ text: '' }],
          orientation: 'Vertical', isPhase: true
        },
      },
    ],
  }
}

```

```

        height: 60,
        width: 140
      }, {
        id: 'horizontalPhase',
        shape: {
          type: 'SwimLane',
          phases: [{ style: { strokeWidth: 1, strokeDashArray:
'3,3', strokeColor: '#A9A9A9' }, }],
          annotations: [{ text: '' }],
          orientation: 'Horizontal', isPhase: true
        },
        height: 60,
        width: 140
      }
    ];
    return swimlaneShapes;
  };
  public getPaletteNodeDefaults(node: NodeModel): void {
    node.width = 100;
    node.height = 100;
    ((node as NodeModel).style as ShapeStyleModel).strokeColor =
    '#3A3A3A';
  }
  public getSymbolInfo() {
    // Enables to fit the content into the specified palette item size
    return {
      fit: true
    };
    // When it is set as false, the element is rendered with actual node
size
  }
  ngOnInit(): void {
    this.expandMode = 'Multiple' as any
    this.palettes = [{
      id: 'swimlane',
      expanded: true,
      symbols: this.getswimlaneShapes(),
      title: 'Swimlane Shapes',
      iconCss: 'e-ddb-icons e-basic'
    } as any] ,
    this.symbolMargin = {
      left: 15,
      right: 15,
      top: 15,
      bottom: 15
    },
    //Specifies the preview size and position to symbol palette items.
    this.symbolPreview = {
      height: 100,
      width: 100,
      offset: {
        x: 0.5,
        y: 0.5
      }
    }
  }
}

```

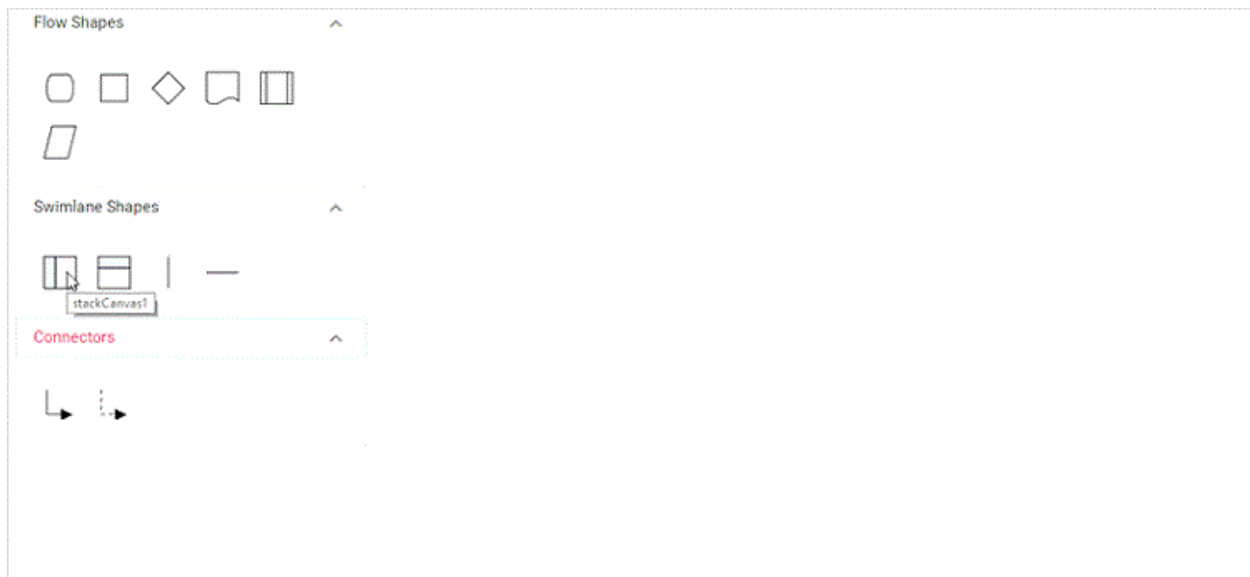
MAIN.TS

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

Drag and drop swimlane to palette

- The drag and drop support for swimlane shapes has been provided.
- When you drag and drop the lane shape, if the diagram already contains swimlane with the same orientation, the lane will be added and stacked inside a swimlane based on the order. Otherwise, it will be added a new swimlane.
- The phase will only drop on swimlane shape with same orientation.

The following image illustrates how to drag symbol from palette.

**Limitations**

- Connectors cannot be canceled when added directly to swimlane. we must initialize the connector through connector collection.
- We cannot edit the phase line style.

Labels in Angular Diagram component

Annotation is a block of text that can be displayed over a node or connector. Annotation is used to textually represent an object with a string that can be edited at runtime. Multiple annotations can be added to a node/connector.

<!-- markdownlint-disable MD033 -->

Create annotation

An annotation can be added to a node/connector by defining the annotation object and adding that to the annotation collection of the node/connector. The [Link to the Video](#) property of annotation defines the text to be displayed. The following code illustrates how to create an annotation.

To create and add annotation to Nodes and Connectors using the Angular Diagram, refer to the below video link.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, OnInit, ViewChild } from
 '@angular/core';
import { DiagramComponent, Diagram, NodeModel, ConnectorModel, PointModel,
 ShapeStyleModel } from '@syncfusion/ej2-angular-diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%"
height="580px" [getNodeDefaults] ='getNodeDefaults' [getConnectorDefaults]
='getConnectorDefaults'>
    <e-nodes>
      <e-node id='node1' [offsetX]=150 [offsetY]=150>
        <e-node-annotations>
          <e-node-annotation content="Annotation">
          </e-node-annotation>
        </e-node-annotations>
      </e-node>
    </e-nodes>
    <e-connectors>
      <e-connector id='connector' type='Orthogonal'
[sourcePoint]='sourcePoint1' [targetPoint]='targetPoint1'>
        <e-connector-annotations>
          <e-connector-annotation content='Annotation'>
          </e-connector-annotation>
        </e-connector-annotations>
      </e-connector>
    </e-connectors>
  </ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram?: DiagramComponent;
  public sourcePoint1?: PointModel;
  public targetPoint1?: PointModel;
  ngOnInit(): void {
    this.sourcePoint1 = { x: 300, y: 100 };
    this.targetPoint1 = { x: 400, y: 300 };
  }
  public getNodeDefaults(node: NodeModel): NodeModel {
```

```

        node.height = 100;
        node.width = 100;
        ((node as NodeModel).style as ShapeStyleModel).fill = "#6BA5D7";
        ((node as NodeModel).style as ShapeStyleModel).strokeColor =
"White";
        return node;
    }
    public getConnectorDefaults(obj: ConnectorModel): void {
        obj.style = {
            strokeColor: '#6BA5D7',
            fill: '#6BA5D7',
            strokeWidth: 2
        }
        obj.targetDecorator = {
            style: {
                fill: '#6BA5D7',
                strokeColor: '#6BA5D7'
            }
        }
    }
}

```

MAIN.TS

```

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

```

Add annotations at runtime

- Annotations can be added at runtime by using the client-side method [addLabels](#). The following code illustrates how to add a annotation to a node.
- The annotation's [ID](#) property is used to define the name of the annotation and its further used to find the annotation at runtime and do any customization.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import { DiagramComponent, Diagram, NodeModel, ShapeAnnotationModel,
ConnectorModel, PathAnnotationModel, PointModel, ShapeStyleModel } from
 '@syncfusion/ej2-angular-diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",

```

```

    template: `<ejs-diagram #diagram id="diagram" width="100%"
height="580px" [getNodeDefaults] ='getNodeDefaults'
(created)='created($event)'>
    <e-nodes>
        <e-node id='node1' [offsetX]=150 [offsetY]=150>
        </e-node>
    </e-nodes>
    <e-connectors>
        <e-connector id='connector1' type='Orthogonal'
[sourcePoint]='sourcePoint1' [targetPoint]='targetPoint1'>
        </e-connector>
    </e-connectors>
</ejs-diagram>`,
    encapsulation: ViewEncapsulation.None
})
export class AppComponent {
    @ViewChild("diagram")
    public diagram?: DiagramComponent;
    public annotation?: ShapeAnnotationModel[];
    public pathannotations?: PathAnnotationModel[];
    public connectors?: ConnectorModel[];
    public sourcePoint1?: PointModel;
    public targetPoint1?: PointModel;
    ngOnInit(): void {
        this.sourcePoint1 = { x: 300, y: 100 };
        this.targetPoint1 = { x: 400, y: 300 };
    }
    public getNodeDefaults(node: NodeModel): NodeModel {
        node.height = 100;
        node.width = 100;
        ((node as NodeModel).style as ShapeStyleModel).fill = "#6BA5D7";
        ((node as NodeModel).style as ShapeStyleModel).strokeColor =
"White";
        return node;
    }
    public created(args: Object): void {
        this.annotation = [{
            id: 'label1',
            content: 'Annotation'
        }];
        this.pathannotations = [{
            content: 'New Connector'
        }];
        //Method to add labels at run time
        (this.diagram as Diagram).addLabels((this.diagram as
Diagram).nodes[0], this.annotation);
        (this.diagram as Diagram).addLabels((this.diagram as
Diagram).getObject('connector1'), this.pathannotations);
        (this.diagram as Diagram).dataBind();
    }
}

```

MAIN.TS

```

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

```

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

Remove annotation

A collection of annotations can be removed from the node by using client-side method [removeLabels](#). The following code illustrates how to remove a annotation to a node.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import { DiagramComponent, Diagram, NodeModel, ShapeAnnotationModel,
ShapeStyleModel, ConnectionShapes } from '@syncfusion/ej2-angular-diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%"
height="580px" [getNodeDefaults] ='getNodeDefaults'
(created)='created($event)'>
    <e-nodes>
      <e-node id='node1' [offsetX]=150 [offsetY]=150>
        <e-node-annotations>
          <e-node-annotation id="label1" content="Annotation">
            </e-node-annotation>
          </e-node-annotations>
        </e-node>
      </e-nodes>
    </ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram?: DiagramComponent;
  public getNodeDefaults(node: NodeModel): NodeModel {
    node.height = 100;
    node.width = 100;
    ((node as NodeModel).style as ShapeStyleModel).fill = "#6BA5D7";
    ((node as NodeModel).style as ShapeStyleModel).strokeColor =
    "White";
    return node;
  }
  public annotation: ShapeAnnotationModel[] = [
    {
      id: 'label1', content: 'Annotation'
    }
  ];
  public created(args: Object): void {
    ((this.diagram as Diagram).removeLabels(((this.diagram as
Diagram).nodes as NodeModel[])[0] as any, this.annotation);
  }
}
```



```
}

```

MAIN.TS

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

Update annotation at runtime

You can change any annotation properties at runtime and update it through the client-side method [dataBind](#).

The following code example illustrates how to change the annotation properties.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import { DiagramComponent, Diagram, NodeModel, ShapeStyleModel,
ShapeAnnotationModel } from '@syncfusion/ej2-angular-diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%"
height="580px" [getNodeDefaults] = 'getNodeDefaults'
(created)='created($event)'>
    <e-nodes>
      <e-node id='node1' [offsetX]=150 [offsetY]=150>
        <e-node-annotations>
          <e-node-annotation id="label1" content="Annotation">
          </e-node-annotation>
        </e-node-annotations>
      </e-node>
    </e-nodes>
  </ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram?: DiagramComponent;
  public getNodeDefaults(node: NodeModel): NodeModel {
    node.height = 100;
    node.width = 100;
    ((node as NodeModel).style as ShapeStyleModel).fill = "#6BA5D7";
    ((node as NodeModel).style as ShapeStyleModel).strokeColor =
    "White";
    return node;
  }
  public created(args: Object): void {
```

```

    ((this.diagram as Diagram).nodes[0].annotations as
ShapeAnnotationModel[])[0].content = 'Updated Annotation';
    (this.diagram as Diagram).dataBind();
  }
}

```

MAIN.TS

```

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

```

Alignment

Annotation can be aligned relative to the node boundaries. It has [margin](#), [offset](#), horizontal, and vertical alignment settings. It is quite tricky when all four alignments are used together but gives more control over alignment.

Offset

The offset property of annotation is used to align the annotations based on fractions. 0 represents top/left corner, 1 represents bottom/right corner, and 0.5 represents half of width/height.

Set the size for a nodes annotation by using [width](#) and [height](#) properties.

The following code shows the relationship between the annotation position (black color circle) and offset (fraction values).

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, OnInit, ViewChild } from
'@angular/core';
import { DiagramComponent, Diagram, NodeModel, PointModel, ShapeStyleModel }
from '@syncfusion/ej2-angular-diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%"
height="580px" [getNodeDefaults] ='getNodeDefaults'>
    <e-nodes>
      <e-node id='node1' [offsetX]=150 [offsetY]=150>
        <e-node-annotations>
          <e-node-annotation id="label1" content="Annotation"
[offset]="offset">
        </e-node-annotation>
      </e-node-annotations>
    </e-node>
  </e-nodes>
</ejs-diagram>`,

```

```

    encapsulation: ViewEncapsulation.None
  })
  export class AppComponent {
    @ViewChild("diagram")
    public diagram?: DiagramComponent;
    public offset?: PointModel;
    ngOnInit(): void {
      //Sets the offset for the content
      this.offset = { x: 0, y: 1}
    }
    public getNodeDefaults(node: NodeModel): NodeModel {
      node.height = 100;
      node.width = 100;
      ((node as NodeModel).style as ShapeStyleModel).fill = "#6BA5D7";
      ((node as NodeModel).style as ShapeStyleModel).strokeColor =
"White";
      return node;
    }
  }
}

```

MAIN.TS

```

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

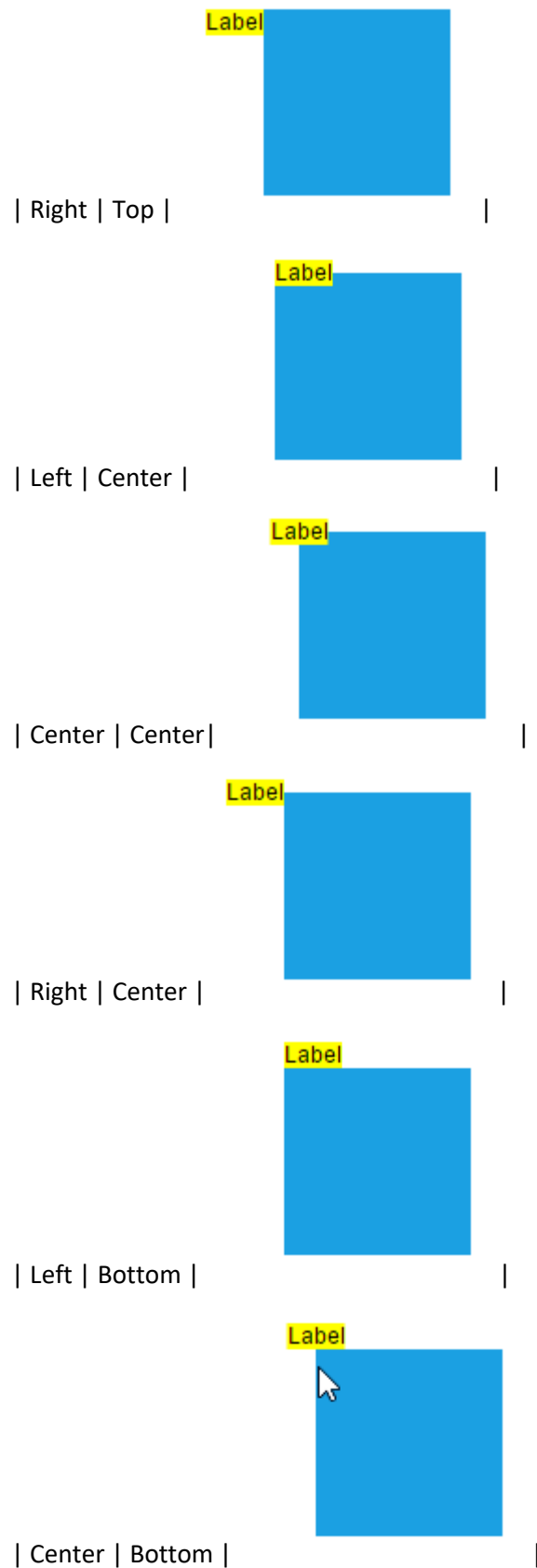
```

Horizontal and vertical alignment

The [horizontalAlignment](#) property of annotation is used to set how the annotation is horizontally aligned at the annotation position determined from the fraction values. The [verticalAlignment](#) property is used to set how annotation is vertically aligned at the annotation position.

The following tables illustrates all the possible alignments visually with 'offset (0, 0)'.

Horizontal Alignment	Vertical Alignment	Output with Offset(0,0)
Left	Top	
Center	Top	





| Right | Bottom |

The following codes illustrates how to align annotations.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, OnInit, ViewChild } from
 '@angular/core';
import { DiagramComponent, Diagram, NodeModel, VerticalAlignment,
 HorizontalAlignment, ShapeStyleModel } from '@syncfusion/ej2-angular-
 diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%"
 height="580px" [getNodeDefaults] ='getNodeDefaults'>
    <e-nodes>
      <e-node id='node1' [offsetX]=150 [offsetY]=150>
        <e-node-annotations>
          <e-node-annotation id="label1" content="Annotation"
 [horizontalAlignment]="horizontalAlignment"
 [verticalAlignment]="verticalAlignment">
            </e-node-annotation>
          </e-node-annotations>
        </e-node>
      </e-nodes>
    </ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram?: DiagramComponent;
  public verticalAlignment?: VerticalAlignment;
  public horizontalAlignment?: HorizontalAlignment;
  ngOnInit(): void {
    // Sets the horizontal alignment as left
    this.horizontalAlignment = 'Left';
    // Sets the vertical alignment as Center
    this.verticalAlignment = 'Center';
  }
  public getNodeDefaults(node: NodeModel): NodeModel {
    node.height = 100;
    node.width = 100;
```

```

        ((node as NodeModel).style as ShapeStyleModel).fill = "#6BA5D7";
        ((node as NodeModel).style as ShapeStyleModel).strokeColor =
"White";
        return node;
    }
}

```

MAIN.TS

```

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

```

Annotation alignment with respect to segments

The offset and alignment properties of annotation allows you to align the connector annotations with respect to the segments.

The following code example illustrates how to align connector annotations.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import { DiagramComponent, Diagram, NodeModel, ConnectorModel,
ShapeStyleModel } from '@syncfusion/ej2-angular-diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%"
height="580px" [getNodeDefaults] ='getNodeDefaults'
[getConnectorDefaults]='getConnectorDefaults'>
    <e-nodes>
      <e-node id='node1' [offsetX]=150 [offsetY]=150>
        <e-node-annotations>
          <e-node-annotation id="label1" content="Task1"
[horizontalAlignment]="horizontalAlignment">
            </e-node-annotation>
          </e-node-annotations>
        </e-node>
      <e-node id='node2' [offsetX]=350 [offsetY]=150>
        <e-node-annotations>
          <e-node-annotation id="label1" content="Task2"
[horizontalAlignment]="horizontalAlignment">
            </e-node-annotation>
          </e-node-annotations>
        </e-node>
      </e-nodes>
    <e-connectors>

```

```

        <e-connector id='connector' type='Orthogonal' [sourceID]='node1'
        [targetID]='node2'>
            <e-connector-annotations>
                <e-connector-annotation content='0' [offset]=0>
                </e-connector-annotation>
                <e-connector-annotation content='1' [offset]=1>
                </e-connector-annotation>
            </e-connector-annotations>
        </e-connector>
    </e-connectors>
</ejs-diagram>`,
    encapsulation: ViewEncapsulation.None
})
export class AppComponent {
    @ViewChild("diagram")
    public diagram?: DiagramComponent;
    node1: any;
    node2: any;
    horizontalAlignment: any;
    public getNodeDefaults(node: NodeModel): NodeModel {
        node.height = 100;
        node.width = 100;
        ((node as NodeModel).style as ShapeStyleModel).fill = "#6BA5D7";
        ((node as NodeModel).style as ShapeStyleModel).strokeColor =
        "White";
        return node;
    }
    public getConnectorDefaults(obj: ConnectorModel): void {
        obj.style = {
            strokeColor: '#6BA5D7',
            fill: '#6BA5D7',
            strokeWidth: 2
        }
        obj.targetDecorator = {
            style: {
                fill: '#6BA5D7',
                strokeColor: '#6BA5D7'
            }
        }
    }
}
}

```

MAIN.TS

```

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

```

Margin

[Margin](#) is an absolute value used to add some blank space in any one of its four sides. The annotations can be displaced with the margin property.

The following code example illustrates how to align a annotation based on its `offset`, `horizontalAlignment`, `verticalAlignment`, and `margin` values.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, OnInit, ViewChild } from
 '@angular/core';
import { DiagramComponent, Diagram, NodeModel, VerticalAlignment,
HorizontalAlignment, MarginModel, PointModel, ShapeStyleModel } from
 '@syncfusion/ej2-angular-diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%"
height="580px" [getNodeDefaults] = 'getNodeDefaults'>
    <e-nodes>
      <e-node id='node1' [offsetX]=150 [offsetY]=150>
        <e-node-annotations>
          <e-node-annotation id="label1" content="Annotation"
[offset]="offset" [horizontalAlignment]="horizontalAlignment"
[verticalAlignment]="verticalAlignment" [margin]="margin">
            </e-node-annotation>
          </e-node-annotations>
        </e-node>
      </e-nodes>
    </ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram?: DiagramComponent;
  public margin?: MarginModel;
  public offset?: PointModel;
  public verticalAlignment?: VerticalAlignment;
  public horizontalAlignment?: HorizontalAlignment;
  ngOnInit(): void {
    // Sets the margin for the content
    this.margin = { top: 10 }
    this.horizontalAlignment = 'Center'
    this.verticalAlignment = 'Top'
    this.offset = { x: 0.5, y: 1 }
  }
  public getNodeDefaults(node: NodeModel): NodeModel {
    node.height = 100;
    node.width = 100;
    ((node as NodeModel).style as ShapeStyleModel).fill = "#6BA5D7";
    ((node as NodeModel).style as ShapeStyleModel).strokeColor =
    "White";
    return node;
  }
}

```


MAIN.TS

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

Text align

The [textAlign](#) property of annotation allows you to set how the text should be aligned (left, right, center, or justify) inside the text block. The following codes illustrate how to set textAlign for an annotation.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, OnInit, ViewChild } from '@angular/core';
import { DiagramComponent, Diagram, NodeModel, TextStyleModel, ShapeStyleModel } from '@syncfusion/ej2-angular-diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%" height="580px" [getNodeDefaults] ='getNodeDefaults'>
    <e-nodes>
      <e-node id='node1' [offsetX]=150 [offsetY]=150>
        <e-node-annotations>
          <e-node-annotation id="label1" content="Text align is set as Left" [style]="style">
            </e-node-annotation>
          </e-node-annotations>
        </e-node>
      </e-nodes>
    </ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram?: DiagramComponent;
  public style?: TextStyleModel;
  ngOnInit(): void {
    // Sets the textAlign as left for the content
    this.style = {
      textAlign: 'Left'
    }
  }
  public getNodeDefaults(node: NodeModel): NodeModel {
    node.height = 100;
    node.width = 100;
    ((node as NodeModel).style as ShapeStyleModel).fill = "#6BA5D7";
```

```

        ((node as NodeModel).style as ShapeStyleModel).strokeColor =
"White";
        return node;
    }
}

```

MAIN.TS

```

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

```

Hyperlink

Diagram provides a support to add a [hyperlink](#) for the nodes/connectors annotation. It can also be customized.

A User can open the hyperlink in the new window, the same tab and the new tab by using the [hyperlinkOpenState](#) property

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, OnInit, ViewChild } from
'@angular/core';
import { DiagramComponent, Diagram, NodeModel, HyperlinkModel,
ShapeStyleModel } from '@syncfusion/ej2-angular-diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%"
height="580px" [getNodeDefaults] ='getNodeDefaults'>
    <e-nodes>
      <e-node id='node1' [height]=80 [width]=180 [offsetX]=300
[offsetY]=100 >
        <e-node-annotations>
          <e-node-annotation [hyperlink]="hyperlink">
            </e-node-annotation>
          </e-node-annotations>
        </e-node>
      </e-nodes>
    </ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram?: DiagramComponent;
  public hyperlink: HyperlinkModel = {
    content: 'Syncfusion', link: 'https://hr.syncfusion.com/home',

```

```

        //Set the link to open in the current tab
        hyperlinkOpenState: 'CurrentTab'
    }
    public getNodeDefaults(node: NodeModel): NodeModel {
        node.height = 100;
        node.width = 100;
        ((node as NodeModel).style as ShapeStyleModel).fill = "#6BA5D7";
        ((node as NodeModel).style as ShapeStyleModel).strokeColor =
        "White";
        return node;
    }
}

```

MAIN.TS

```

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

```

Template Support for Annotation

Diagram provides template support for annotation. you should define a SVG/HTML content as string in the annotation's [template](#) property.

The following code illustrates how to define a template in node's annotation. similarly, you can define it in connectors.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, OnInit, ViewChild } from
 '@angular/core';
import { DiagramComponent, NodeModel, ConnectorModel } from
 '@syncfusion/ej2-angular-diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%"
height="580px" [nodes]="nodes" [connectors]="connectors">
</ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram?: DiagramComponent;
  public nodes: NodeModel[] = [
    {
      id: 'node1', offsetX: 150, offsetY: 150, width:100, height:100,

```

```

        annotations: [{ id:"label1", template:'<div><input type="button"
value="Submit"></div>' }],
    }
  ]
  public connectors: ConnectorModel[] = [
    {
      id: 'connector', sourcePoint: { x: 300, y: 100 }, targetPoint: {
x: 400, y: 300 },
      annotations: [{ id:"label1", height: 60, width: 100, offset:
0.5, template:'<div><input type="button" value="Submit"></div>' }],
    }
  ]
  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));

```

Wrapping

When text overflows node boundaries, you can control it by using [text wrapping](#). So, it is wrapped into multiple lines. The wrapping property of annotation defines how the text should be wrapped. The following code illustrates how to wrap a text in a node.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, OnInit, ViewChild } from
'@angular/core';
import { DiagramComponent, Diagram, NodeModel, TextStyleModel,
ShapeStyleModel } from '@syncfusion/ej2-angular-diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%"
height="580px" [getNodeDefaults] ='getNodeDefaults'>
    <e-nodes>
      <e-node id='node1' [offsetX]=150 [offsetY]=150>
        <e-node-annotations>
          <e-node-annotation id="label1" content="Annotation Text
Wrapping" [style]="style">
            </e-node-annotation>
          </e-node-annotations>
        </e-node>
      </e-nodes>
    `
})

```

```

    </e-nodes>
  </ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram?: DiagramComponent;
  public style?: TextStyleModel;
  ngOnInit(): void {
    this.style = {
      textWrapping: 'Wrap'
    }
  }
  public getNodeDefaults(node: NodeModel): NodeModel {
    node.height = 100;
    node.width = 100;
    ((node as NodeModel).style as ShapeStyleModel).fill = "#6BA5D7";
    ((node as NodeModel).style as ShapeStyleModel).strokeColor =
    "White";
    return node;
  }
}

```

MAIN.TS

```

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

```

| Value | Description | Image |

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



| No Wrap | Text will not be wrapped. |

| Wrap | Text-wrapping occurs, when the text overflows beyond the available node width. |



| WrapWithOverflow (Default) | Text-wrapping occurs, when the text overflows beyond the available node width. However, the text may overflow beyond the node width in the case of a very long word. |



Text overflow

The label's [TextOverflow](#) property is used control whether to display the overflowed content in node or not.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, OnInit, ViewChild } from '@angular/core';
import { DiagramComponent, Diagram, NodeModel, TextStyleModel, ShapeStyleModel } from '@syncfusion/ej2-angular-diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%" height="580px" [getNodeDefaults] = 'getNodeDefaults'>
    <e-nodes>
      <e-node id='node1' [offsetX]=150 [offsetY]=150>
        <e-node-annotations>
          <e-node-annotation id="label1" content="Annotation Text" [style]="style">
            </e-node-annotation>
          </e-node-annotations>
        </e-node>
      </e-nodes>
    </ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram?: DiagramComponent;
  public style?: TextStyleModel;
  ngOnInit(): void {
    this.style = {
      textOverflow: 'Ellipsis'
    }
  }
  public getNodeDefaults(node: NodeModel): NodeModel {
    node.height = 100;
  }
}
```

```

        node.width = 100;
        ((node as NodeModel).style as ShapeStyleModel).fill = "#6BA5D7";
        ((node as NodeModel).style as ShapeStyleModel).strokeColor =
"White";
        return node;
    }
}

```

MAIN.TS

```

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

```

Appearance

- You can change the font style of the annotations with the font specific properties (fontSize, fontFamily, color). The following code illustrates how to customize the appearance of the annotation.
- The label's [bold](#), [italic](#), and [textDecoration](#) properties are used to style the label's text.
- The label's [fill](#), [strokeColor](#), and [strokeWidth](#) properties are used to define the background color and border color of the annotation and the [opacity](#) property is used to define the transparency of the annotations.
- The [visible](#) property of the annotation enables or disables the visibility of annotation.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, OnInit, ViewChild } from
'@angular/core';
import { DiagramComponent, Diagram, NodeModel, TextStyleModel,
ShapeStyleModel } from '@syncfusion/ej2-angular-diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%"
height="580px" [getNodeDefaults] ='getNodeDefaults'>
    <e-nodes>
      <e-node id='node1' [offsetX]=150 [offsetY]=150>
        <e-node-annotations>
          <e-node-annotation id="label1" content="Annotation Text"
[style]="style">
            </e-node-annotation>
          </e-node-annotations>
        </e-node>
      </e-nodes>
    `
})

```

```

</ejs-diagram>`,
    encapsulation: ViewEncapsulation.None
  })
  export class AppComponent {
    @ViewChild("diagram")
    public diagram?: DiagramComponent;
    public style?: TextStyleModel;
    ngOnInit(): void {
      //this.style: {
      //  color: 'black',
      //  bold: true,
      //  italic: true,
      //  fontSize: '12',
      //  fontFamily: 'TimesNewRoman'
      //}
      public getNodeDefaults(node: NodeModel): NodeModel {
        node.height = 100;
        node.width = 100;
        ((node as NodeModel).style as ShapeStyleModel).fill = "#6BA5D7";
        ((node as NodeModel).style as ShapeStyleModel).strokeColor =
"White";
        return node;
      }
    }
  }
}

```

MAIN.TS

```

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

```

The fill, border, and opacity appearances of the text can also be customized with appearance specific properties of annotation. The following code illustrates how to customize background, opacity, and border of the annotation.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, OnInit, ViewChild } from
'@angular/core';
import { DiagramComponent, Diagram, NodeModel, TextStyleModel,
ShapeStyleModel } from '@syncfusion/ej2-angular-diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%"
height="580px" [getNodeDefaults] = 'getNodeDefaults'>

```



```

        <e-nodes>
          <e-node id='node1' [offsetX]=150 [offsetY]=150>
            <e-node-annotations>
              <e-node-annotation id="label1" content="Annotation Text"
[style]="style">
                </e-node-annotation>
              </e-node-annotations>
            </e-node>
          </e-nodes>
        </ejs-diagram>`,
        encapsulation: ViewEncapsulation.None
      })
    export class AppComponent {
      @ViewChild("diagram")
      public diagram?: DiagramComponent;
      public style?: TextStyleModel;
      ngOnInit(): void {
        //this.style: {
        //  color: 'black',
        //  fill: 'white',
        //  opacity: 0.7,
        //  strokeColor: 'black',
        //  strokeWidth: 2
        //}
        public getNodeDefaults(node: NodeModel): NodeModel {
          node.height = 100;
          node.width = 100;
          ((node as NodeModel).style as ShapeStyleModel).fill = "#6BA5D7";
          ((node as NodeModel).style as ShapeStyleModel).strokeColor =
"White";
          return node;
        }
      }
    }
  }

```

MAIN.TS

```

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

```

Interaction

Diagram allows annotation to be interacted by selecting, dragging, rotating, and resizing. Annotation interaction is disabled, by default. You can enable annotation interaction with the [constraints](#) property of annotation. You can also curtail the services of interaction by enabling either selecting, dragging, rotating, or resizing individually with the respective constraints property of annotation. The following code illustrates how to enable annotation interaction.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'

```

```

import { Component, ViewEncapsulation, OnInit, ViewChild } from
'@angular/core';
import { DiagramComponent, Diagram, NodeModel, AnnotationConstraints,
ShapeStyleModel } from '@syncfusion/ej2-angular-diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%"
height="580px" [getNodeDefaults] ='getNodeDefaults'>
    <e-nodes>
      <e-node id='node1' [offsetX]=150 [offsetY]=150>
        <e-node-annotations>
          <e-node-annotation id="label1" content="Annotation Text"
[constraints]="constraints">
            </e-node-annotation>
          </e-node-annotations>
        </e-node>
      </e-nodes>
    </ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram?: DiagramComponent;
  public constraints?: AnnotationConstraints;
  ngOnInit(): void {
    this.constraints = AnnotationConstraints.Interaction
  }
  public getNodeDefaults(node: NodeModel): NodeModel {
    node.height = 100;
    node.width = 100;
    ((node as NodeModel).style as ShapeStyleModel).fill = "#6BA5D7";
    ((node as NodeModel).style as ShapeStyleModel).strokeColor =
"White";
    return node;
  }
}

```

MAIN.TS

```

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

```

Edit

Diagram provides support to edit an annotation at runtime, either programmatically or interactively. By default, annotation is in view mode. But it can be brought to edit mode in two ways;

- Programmatically, by using [startTextEdit](#) method, edit the text through programmatically.

- Interactively
 1. By double-clicking the annotation.
 2. By selecting the item and pressing the F2 key.

Double-clicking any annotation will enable editing and the node enables first annotation editing. When the focus of editor is lost, the annotation for the node is updated.

When you double-click on the node/connector/diagram model, the [doubleClick](#) event gets triggered.

Read-only annotations

Diagram allows to create read-only annotations. You have to set the read-only property of annotation to enable/disable the read-only [constraints](#). The following code illustrates how to enable read-only mode.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, OnInit, ViewChild } from
 '@angular/core';
import { DiagramComponent, Diagram, NodeModel, AnnotationConstraints,
 ShapeStyleModel } from '@syncfusion/ej2-angular-diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%"
height="580px" [getNodeDefaults] ='getNodeDefaults'>
    <e-nodes>
      <e-node id='node1' [offsetX]=150 [offsetY]=150>
        <e-node-annotations>
          <e-node-annotation id="label1" content="Annotation Text"
[constraints]="constraints">
            </e-node-annotation>
          </e-node-annotations>
        </e-node>
      </e-nodes>
    </ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram?: DiagramComponent;
  public constraints?: AnnotationConstraints;
  ngOnInit(): void {
    this.constraints = AnnotationConstraints.ReadOnly
  }
  public getNodeDefaults(node: NodeModel): NodeModel {
    node.height = 100;
    node.width = 100;
    ((node as NodeModel).style as ShapeStyleModel).fill = "#6BA5D7";
    ((node as NodeModel).style as ShapeStyleModel).strokeColor =
"White";
```

```

        return node;
    }
}

```

MAIN.TS

```

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

```

Drag Limit

- The diagram control now supports defining the [dragLimit](#) to the label while dragging from the connector and also update the position to the nearest segment offset.
- You can set the value to dragLimit [left](#), [right](#), [top](#), and [bottom](#) properties which allow the dragging of connector labels to a certain limit based on the user defined values.
- By default, drag limit will be disabled for the connector. It can be enabled by setting connector constraints as drag.
- The following code illustrates how to set a dragLimit for connector annotations.

`typescript

```

import { Component, ViewEncapsulation, OnInit, ViewChild } from '@angular/core';
import { DiagramComponent, Diagram, NodeModel, AnnotationConstraints } from '@syncfusion/ej2-angular-diagrams';

@Component({
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%" height="580px">
    <e-connectors>
      <e-connector id='connector' type='Orthogonal' [sourcePoint]='sourcePoint1'
        [targetPoint]='targetPoint1'>
        <e-connector-annotations>
          <e-connector-annotation content='connector' [constraints]="constraints" [dragLimit]="dragLimit">
          </e-connector-annotation>
        </e-connector-annotations>
      </e-connector>
    </e-connectors>
  </ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {

```

```

@ViewChild("diagram")
public diagram: DiagramComponent;
public constraints: AnnotationConstraints;
public dragLimit: MarginModel;
ngOnInit(): void {
  this.sourcePoint1 = { x: 300, y: 100 };
  this.targetPoint1 = { x: 400, y: 300 };
  //Enables drag constraints for a connector.
  this.constraints = AnnotationConstraints.Interaction | AnnotationConstraints.Drag;
  //Set drag limit for a connector annotation.
  this.dragLimit = {left:20,right:20,top:20,bottom:20};
}
}
,

```

Multiple annotations

You can add any number of annotations to a node or connector. The following code illustrates how to add multiple annotations to a node.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, OnInit, ViewChild } from
 '@angular/core';
import { DiagramComponent, Diagram, NodeModel, PointModel, ShapeStyleModel }
from '@syncfusion/ej2-angular-diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%"
height="580px" [getNodeDefaults] ='getNodeDefaults'>
    <e-nodes>
      <e-node id='node1' [offsetX]=150 [offsetY]=150>
        <e-node-annotations>
          <e-node-annotation id="label1" content="Left"
[offset]="offset1">
        </e-node-annotation>
          <e-node-annotation id="label2" content="Center"
[offset]="offset2">
        </e-node-annotation>
          <e-node-annotation id="label3" content="Right"
[offset]="offset3">

```

```

        </e-node-annotation>
      </e-node-annotations>
    </e-node>
  </e-nodes>
</ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram?: DiagramComponent;
  public offset1?: PointModel;
  public offset2?: PointModel;
  public offset3?: PointModel;
  ngOnInit(): void {
    this.offset1 = { x: 0.5, y: 1}
    this.offset2 = { x: 0.5, y: 1}
    this.offset3 = { x: 0.5, y: 1}
  }
  public getNodeDefaults(node: NodeModel): NodeModel {
    node.height = 100;
    node.width = 100;
    ((node as NodeModel).style as ShapeStyleModel).fill = "#6BA5D7";
    ((node as NodeModel).style as ShapeStyleModel).strokeColor =
    "White";
    return node;
  }
}

```

MAIN.TS

```

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

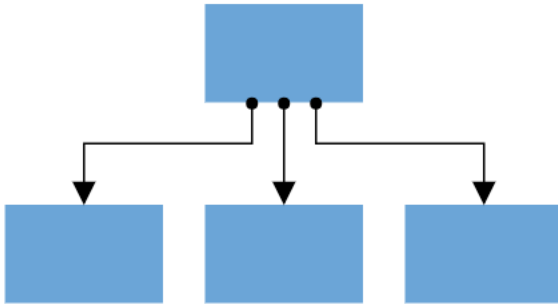
```

Constraints

The constraints property of annotation allows you to enable/disable certain annotation behaviors. For instance, you can disable annotation editing.

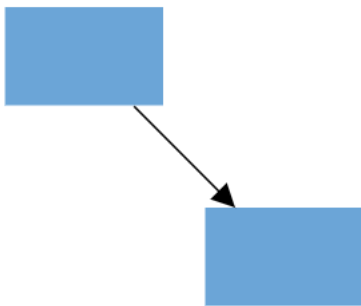
Ports in Angular Diagram component

Diagram provides support to define custom ports for making connections.

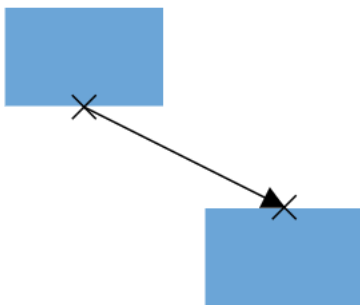


<!-- markdownlint-disable MD033 -->

When a connector is connected between two nodes, its end points are automatically docked to the node's nearest boundary as shown in the following image.



Ports act as the connection points of the node and allows to create connections with only those specific points as shown in the following image.



[Link to the Video](#)

Create port

To create and customize the ports in Angular Diagram, refer to the below video link,

Add ports when initializing nodes

To add a connection port, define the port object and add it to node's ports collection. The `offset` property of port accepts an object of fractions and used to determine the position of ports. The following code illustrates how to add ports when initializing the node.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import { DiagramComponent, NodeModel, PointPortModel, PortVisibility,
ShapeStyleModel } from '@syncfusion/ej2-angular-diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  // specifies the template string for the diagram component
  template: `<ejs-diagram #diagram id="diagram" width="100%" height="580px"
[getNodeDefaults]='getNodeDefaults'>
    <e-nodes>
      <e-node id='node1' [offsetX]=150 [offsetY]=150
[ports]='ports'></e-node>
    </e-nodes>
  </ejs-diagram>`
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram?: DiagramComponent;
  public ports: PointPortModel[] = [{
    // Sets the position for the port
    offset: {
      x: 0.5,
      y: 0.5
    },
    visibility: PortVisibility.Visible
  }]
  public getNodeDefaults(node: NodeModel): NodeModel {
    node.height = 100;
    node.width = 100;
    ((node as NodeModel).style as ShapeStyleModel).fill = "#6BA5D7";
    ((node as NodeModel).style as ShapeStyleModel).strokeColor =
    "White";
    return node;
  }
}
```

MAIN.TS

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


Add ports at runtime

Add ports at runtime by using the client-side method [addPorts](#). The following code illustrates how to add ports to node at runtime.

The port's ID property is used to define the unique ID for the port and its further used to find the port at runtime. If ID is not set, then default ID is automatically set.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import { Diagram, DiagramComponent, NodeModel, PointPortModel,
PortVisibility, ShapeStyleModel } from '@syncfusion/ej2-angular-diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  // specifies the template string for the diagram component
  template: `<ejs-diagram #diagram id="diagram" width="100%" height="580px"
[getNodeDefaults]='getNodeDefaults' (created)='created($event)'>
    <e-nodes>
      <e-node id='node1' [offsetX]=150 [offsetY]=150></e-node>
    </e-nodes>
  </ejs-diagram>`
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram?: DiagramComponent;
  public ports: PointPortModel[] = [
    {
      id: 'port1',
      offset: {
        x: 0,
        y: 0.5
      },
      visibility: PortVisibility.Visible
    },
    {
      id: 'port2',
      offset: {
        x: 1,
        y: 0.5
      },
      visibility: PortVisibility.Visible
    },
    {
      id: 'port3',
      offset: {
        x: 0.5,
        y: 0
      }
    }
  ]
}
```

```

        },
        visibility: PortVisibility.Visible
    },
    {
        id: 'port4',
        offset: {
            x: 0.5,
            y: 1
        },
        visibility: PortVisibility.Visible
    }
]
public getNodeDefaults(node: NodeModel): NodeModel {
    node.height = 100;
    node.width = 100;
    ((node as NodeModel).style as ShapeStyleModel).fill = "#6BA5D7";
    ((node as NodeModel).style as ShapeStyleModel).strokeColor =
    "White";
    return node;
}
public created(args: Object): void {
    // Method to add ports through run time
    (this.diagram as Diagram).addPorts((this.diagram as
Diagram).nodes[0], this.ports);
}
}

```

MAIN.TS

```

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

```

Remove ports at runtime

Remove ports at runtime by using client-side method [removePorts](#). Refer to the following example which shows how to remove ports at runtime.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import { Diagram, DiagramComponent, NodeModel, PointPortModel,
PortVisibility, ShapeStyleModel } from '@syncfusion/ej2-angular-diagrams';
@Component({
    imports: [
        DiagramModule
    ],
    providers: [ ],
    standalone: true,
    selector: "app-container",
    // specifies the template string for the diagram component

```

```

    template: `<ejs-diagram #diagram id="diagram" width="100%" height="580px"
[getNodeDefaults]='getNodeDefaults' (created)='created($event)'>
    <e-nodes>
        <e-node id='node1' [offsetX]=150 [offsetY]=150
[ports]='ports'></e-node>
    </e-nodes>
</ejs-diagram>`
  })
  export class AppComponent {
    @ViewChild("diagram")
    public diagram?: DiagramComponent;
    public ports: PointPortModel[] = [
      {
        id: 'port1',
        offset: {
          x: 0,
          y: 0.5
        },
        visibility: PortVisibility.Visible
      },
      {
        id: 'port2',
        offset: {
          x: 1,
          y: 0.5
        },
        visibility: PortVisibility.Visible
      },
      {
        id: 'port3',
        offset: {
          x: 0.5,
          y: 0
        },
        visibility: PortVisibility.Visible
      },
      {
        id: 'port4',
        offset: {
          x: 0.5,
          y: 1
        },
        visibility: PortVisibility.Visible
      }
    ]
    public getNodeDefaults(node: NodeModel): NodeModel {
      node.height = 100;
      node.width = 100;
      ((node as NodeModel).style as ShapeStyleModel).fill = "#6BA5D7";
      ((node as NodeModel).style as ShapeStyleModel).strokeColor =
"White";
      return node;
    }
    public ports1: PointPortModel[] = [{
      id: 'port1',
    }, {
      id: 'port2',

```

```

    }, {
      id: 'port3',
    }, {
      id: 'port4',
    }
  ]
  public created(args: Object): void {
    // Method to add ports through run time
    (this.diagram as Diagram).removePorts((this.diagram as
Diagram).nodes[0] as any, this.ports1);
  }
}

```

MAIN.TS

```

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

```

Update port at runtime

You can change any port properties at runtime and update it through the client-side method [dataBind](#).

The following code example illustrates how to change the port properties.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import { Diagram, DiagramComponent, NodeModel, PointPortModel,
PortVisibility, ShapeStyleModel } from '@syncfusion/ej2-angular-diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  // specifies the template string for the diagram component
  template: `<ejs-diagram #diagram id="diagram" width="100%" height="580px"
[getNodeDefaults]='getNodeDefaults' (created)= 'created($event)'>
    <e-nodes>
      <e-node id='node1' [offsetX]=150 [offsetY]=150
[ports]='ports'></e-node>
    </e-nodes>
  </ejs-diagram>`
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram?: DiagramComponent;
  public ports: PointPortModel[] = [{
    // Sets the position for the port
    offset: {
      x: 0.5,
      y: 0.5
    }
  }];
}

```

```

    },
    visibility: PortVisibility.Visible
  ]]
  public getNodeDefaults(node: NodeModel): NodeModel {
    node.height = 100;
    node.width = 100;
    ((node as NodeModel).style as ShapeStyleModel).fill = "#6BA5D7";
    ((node as NodeModel).style as ShapeStyleModel).strokeColor =
    "White";
    return node;
  }
  public created(args: Object): void {
    // Method to add ports through run time
    ((this.diagram as Diagram).nodes[0] as any).ports[0].offset = {
      x: 1,
      y: 1
    };
    (this.diagram as Diagram).dataBind();
  }
}

```

MAIN.TS

```

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

```

Appearance

- The shape of port can be changed by using its shape property. To explore the different types of port shapes, refer to Port Shapes. If you need to render a custom shape, then you can set shape as path and define path using path data property of port.
- The appearance of ports can be customized by using [strokeColor](#),

[strokeWidth](#), and [fill](#) properties of the port.

- Customize the port size by using the [width](#) and [height](#) properties of port.
- The ports [visibility](#) property allows you to define, when the port should be visible.

The following code illustrates how to change the appearance of port.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import { DiagramComponent, NodeModel, PointPortModel, PortVisibility,
ShapeStyleModel } from '@syncfusion/ej2-angular-diagrams';
@Component({
  imports: [
    DiagramModule

```

```

    ],
    providers: [ ],
    standalone: true,
    selector: "app-container",
    // specifies the template string for the diagram component
    template: `<ejs-diagram #diagram id="diagram" width="100%" height="580px"
[getNodeDefaults]='getNodeDefaults'>
        <e-nodes>
            <e-node id='node1' [offsetX]=150 [offsetY]=150
[ports]='ports'></e-node>
        </e-nodes>
    </ejs-diagram>`
  })
  export class AppComponent {
    @ViewChild("diagram")
    public diagram?: DiagramComponent;
    public ports: PointPortModel[] = [{
      offset: {
        x: 1,
        y: 0.5
      },
      visibility: PortVisibility.Visible,
      //Set the style for the port
      style: {
        fill: 'red',
        strokeWidth: 2,
        strokeColor: 'black'
      },
      width: 12,
      height: 12,
      // Sets the shape of the port as Circle
      shape: 'Circle'
    }]
    public getNodeDefaults(node: NodeModel): NodeModel {
      node.height = 100;
      node.width = 100;
      ((node as NodeModel).style as ShapeStyleModel).fill = "#6BA5D7";
      ((node as NodeModel).style as ShapeStyleModel).strokeColor =
"White";
      return node;
    }
  }
}

```

MAIN.TS

```

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

```

Offset

The offset property of port is used to align the port based on fractions. 0 represents top/left corner, 1 represents bottom/right corner, and 0.5 represents half of width/height.

Constraints

The constraints property allows to enable/disable certain behaviors of ports. For more information about port constraints, refer to [Port Constraints](#).

Constraints in Angular Diagram component

Constraints are used to enable/disable certain behaviors of the diagram, nodes and connectors. Constraints are provided as flagged enumerations, so that multiple behaviors can be enabled/disabled using Bitwise operators (&, |, ~, <<, etc.).

To know more about Bitwise operators, refer to [Bitwise Operations](#).

Diagram constraints

Diagram constraints allow to enable or disable the following behaviors:

- Page editing
- Bridging
- Zoom and pan
- Undo/redo
- Tooltip

The following example illustrates how to give page editing using the diagram constraints.

```
`typescript
import { Component, ViewEncapsulation, ViewChild } from "@angular/core";
import { DiagramConstraints } from '@syncfusion/ej2-diagrams';

@Component({
  selector: "app-container",
  // specifies the template string for the diagram component
  template: <ejs-diagram id="diagram" width="100%" height="580px"
[constraints]='diagramConstraints'></ejs-diagram>
})
export class AppComponent {
  @ViewChild("diagram")
  public diagramConstraints: DiagramConstraints;
  ngOnInit(): void {
    this.diagramConstraints = DiagramConstraints.Default & ~DiagramConstraints.PageEditable;
  }
}
```

For more information about diagram constraints, refer to [DiagramConstraints](#).

Node constraints

Node constraints allows to enable or disable the following behaviors of node. They are as follows:

- Selection
- Deletion
- Drag
- Resize
- Rotate
- Connect
- Shadow
- Tooltip

The following example illustrates how to disable rotation using the node constraints.

```
`typescript
import { Component, ViewEncapsulation, ViewChild } from "@angular/core";
import { NodeConstraints } from "@syncfusion/ej2-angular-diagrams";

@Component({
  selector: "app-container",
  template: `<ejs-diagram id="diagram" width="100%" height="580px">
    <e-nodes>
      <e-node id='node1' [height]=60 [width]=100 [offsetX]=300 [offsetY]=80 [constraints]='nodeConstraints'>
    </e-node>
    </e-nodes>
  </ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild("diagram")
  public nodeConstraints: NodeConstraints;

  ngOnInit(): void {
    this.nodeConstraints = NodeConstraints.Default & ~NodeConstraints.Rotate;
  }
}
```

For more information about node constraints, refer to [NodeConstraints](#).

Connector constraints

Connector constraints allow to enable or disable certain behaviors of connectors.

- Selection
- Deletion

- Drag
- Segment editing
- Tooltip
- Bridging

The following code illustrates how to disable selection by using connector constraints.

```
`typescript
import { Component, ViewEncapsulation, ViewChild } from "@angular/core";
import { ConnectorConstraints, PointModel } from "@syncfusion/ej2-angular-diagrams";

@Component({
  selector: "app-container",
  template: `<ejs-diagram id="diagram" width="100%" height="580px">
    <e-connectors>
      <e-connector id='connector' type='Straight' [sourcePoint]='sourcePoint' [targetPoint]='targetPoint'
        [constraints]='connectorConstraints'>
    </e-connector>
    </e-connectors>
  </ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild("diagram")
  public connectorConstraints: ConnectorConstraints;
  public sourcePoint: PointModel;
  public targetPoint: PointModel;
  ngOnInit(): void {
    this.sourcePoint: { x: 100, y: 100 },
    this.targetPoint: { x: 200, y: 200 },
    this.connectorConstraints = ConnectorConstraints.Default & ~ConnectorConstraints.Select;
  }
}
```

For more information about connector constraints, refer to [ConnectorConstraints](#).

Port constraints

You can enable or disable certain behaviors of port. They are as follows:

- Connect
- ConnectOnDrag

The following code illustrates how to disable creating connections with a port.

```
`typescript
import { Component, ViewEncapsulation, ViewChild } from "@angular/core";
import { PointPortModel, PortConstraints } from "@syncfusion/ej2-angular-diagrams";
@Component({
  selector: "app-container",
  template: `<ejs-diagram id="diagram" width="100%" height="580px">
<e-nodes>
<e-node id='node1' [height]=60 [width]=100 [offsetX]=300 [offsetY]=80 [ports]='port1'>
</e-node>
</e-nodes>
</ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild("diagram")
  public port1: PointPortModel[];
  ngOnInit(): void {
    this.port1 = [{
      id: 'port1',
      shape: 'Circle',
      offset: { x: 0, y: 0.5 },
      text: 'In - 1',
      constraints: PortConstraints.None
    }];
  }
}
```

For more information about port constraints, refer to [PortConstraints](#).

Annotation constraints

You can enable or disable read-only mode for the annotations by using the annotation constraints.

The following code illustrates how to enable read-only mode for the annotations.

```
`typescript
import { Component, ViewEncapsulation, ViewChild } from "@angular/core";
import { AnnotationConstraints } from "@syncfusion/ej2-angular-diagrams";

@Component({
  selector: "app-container",
  template: `<ejs-diagram id="diagram" width="100%" height="580px">
    <e-nodes>
    <e-node id='node1' [height]=60 [width]=100 [offsetX]=300 [offsetY]=80>
    <e-node-annotations>
    <e-node-annotation content='Start' [constraints]='annotationConstraints'></e-node-annotation>
    </e-node-annotations>
    </e-node>
    </e-nodes>
  </ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild("diagram")
  public annotationConstraints: AnnotationConstraints;

  ngOnInit(): void {
    this.annotationConstraints = AnnotationConstraints.ReadOnly;
  }
}
```

For more details about annotation constraints, refer to [AnnotationConstraints](#).

Selector constraints

Selector visually represents the selected elements with certain editable thumbs. The visibility of the thumbs can be controlled with selector constraints. The part of selector is categorized as follows:

- Resizer
- Rotator
- User handles

The following code illustrates how to hide rotator.

```
`typescript
```

```
import { Component, ViewEncapsulation, ViewChild } from "@angular/core";
import { SelectorModel, SelectorConstraints } from '@syncfusion/ej2-diagrams';
@Component({
  selector: "app-container",
  // specifies the template string for the diagram component
  template: <ejs-diagram id="diagram" width="100%" height="580px"
[selectedItems]='selectedItems'></ejs-diagram>
})
export class AppComponent {
  @ViewChild("diagram")
  public selectedItems: SelectorModel;
  ngOnInit(): void {
    this.selectedItems = { constraints: SelectorConstraints.All & ~SelectorConstraints.Rotate };
  }
}
```

For more information about selector constraints, refer to [SelectorConstraints](#).

Snap constraints

Snap constraints control the visibility of gridlines and enable/disable snapping. Snap constraints allow to set the following behaviors.

- Show only horizontal or vertical gridlines.
- Show both horizontal and vertical gridlines.
- Snap to either horizontal or vertical gridlines.
- Snap to both horizontal and vertical gridlines.

The following code illustrates how to show only horizontal gridlines.

```
`typescript
import { Component, ViewEncapsulation, ViewChild } from "@angular/core";
import { SnapSettingsModel, SnapConstraints } from '@syncfusion/ej2-diagrams';
@Component({
  selector: "app-container",
  // specifies the template string for the diagram component
  template: <ejs-diagram id="diagram" width="100%" height="580px"
[snapSettings]='snapSettings'></ejs-diagram>
})
```

```
export class AppComponent {  
  @ViewChild("diagram")  
  public snapSettings: SnapSettingsModel;  
  ngOnInit(): void {  
    this.snapSettings: {  
      constraints: SnapConstraints.ShowHorizontalLines  
    };  
  }  
}
```

For more information about snap constraints, refer to [SnapConstraints](#).

Boundary constraints

Boundary constraints defines a boundary for the diagram inside which the interaction should be done. Boundary constraints allow to set the following behaviors.

- Infinite boundary
- Diagram sized boundary
- Page sized boundary

The following code illustrates how to limit the interaction done inside a diagram within a page.

```
`typescript  
import { Component, ViewEncapsulation, ViewChild } from "@angular/core";  
import { PageSettingsModel, BoundaryConstraints } from '@syncfusion/ej2-diagrams';  
@Component({  
  selector: "app-container",  
  template: <ejs-diagram id="diagram" width="100%" height="580px"  
    [pageSettings]='pageSettings'></ejs-diagram>  
})  
export class AppComponent {  
  @ViewChild("diagram")  
  public pageSettings: PageSettingsModel;  
  ngOnInit(): void {  
    this.pageSettings: {  
      boundaryConstraints: 'Page'  
    };  
  }  
}
```

```
}
`
```

For more information about selector constraints, refer to [BoundaryConstraints](#).

Inherit behaviors

Some of the behaviors can be defined through both the specific object (node/connector) and diagram. When the behaviors are contradictorily defined through both, the actual behavior is set through inherit options.

The following code example illustrates how to inherit the line bridging behavior from the diagram model.

```
`typescript
import { Component, ViewEncapsulation, ViewChild } from "@angular/core";
import { DiagramConstraints, ConnectorBridging, ConnectorConstraints, PointModel } from
"@syncfusion/ej2-angular-diagrams";
Diagram.Inject(ConnectorBridging);
@Component({
  selector: "app-container",
  template: `

```

```
this.targetPoint: { x: 200, y: 200 },  
this.connectorConstraints = ConnectorConstraints.Default & ConnectorConstraints.InheritBridging;  
}  
}  
`
```

Bitwise operations

Bitwise operations are used to manipulate the flagged enumerations [enum]. In this section, Bitwise operations are illustrated by using node constraints. The same is applicable while working with node constraints, connector constraints, or port constraints.

Add operation

You can add or enable multiple values at a time by using Bitwise `|` (OR) operator.

```
`typescript  
node.constraints = NodeConstraints.Select | NodeConstraints.Rotate;  
`
```

In the previous example, you can do both the selection and rotation operation.

Remove Operation

You can remove or disable values by using Bitwise `&~` (XOR) operator.

```
`typescript  
node.constraints = node.constraints & ~(NodeConstraints.Rotate);  
`
```

In the previous example, rotation is disabled but other constraints are enabled.

Check operation

You can check any value by using Bitwise `&` (AND) operator.

```
`typescript  
if ((node.constraints & (NodeConstraints.Rotate)) == (NodeConstraints.Rotate));  
`
```

In the previous example, check whether the rotate constraints are enabled in a node. When node constraints have rotate constraints, the expression returns a rotate constraint.

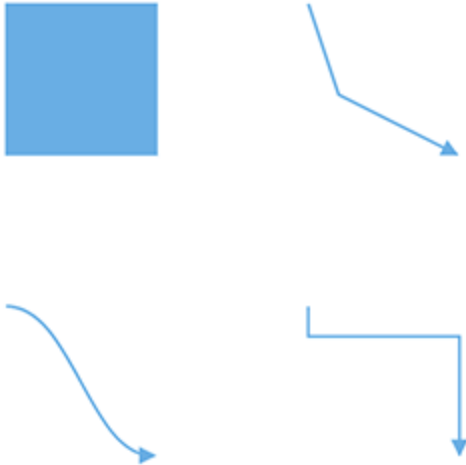
Interaction in Angular Diagram component

Selection

Selector provides a visual representation of selected elements. It behaves like a container and allows to update the size, position, and rotation angle of the selected elements through interaction and by using program. Single or multiple elements can be selected at a time.

Single selection

An element can be selected by clicking that element. During single click, all previously selected items are cleared. The following image shows how the selected elements are visually represented.



- While selecting the diagram elements, the following events can be used to do your customization.
- When selecting/unselecting the diagram elements, the [selectionChange](#) event gets triggered.

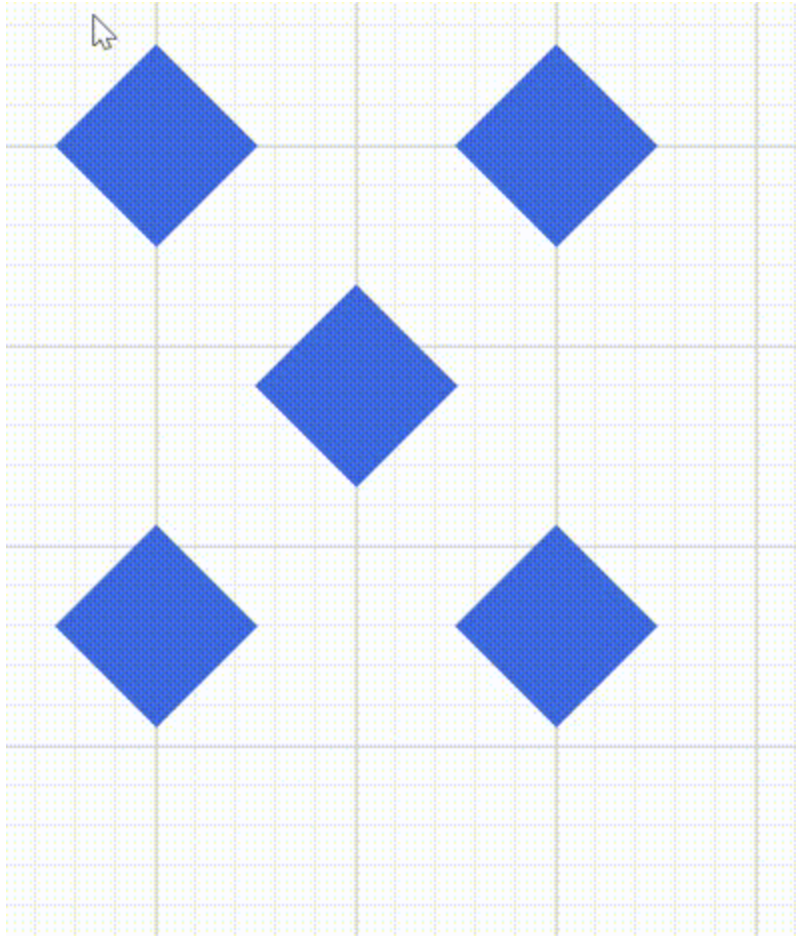
Selecting a group

When a child element of any group is clicked, its contained group is selected instead of the child element. With consecutive clicks on the selected element, selection is changed from top to bottom in the hierarchy of parent group to its children.

Multiple selection

Multiple elements can be selected with the following ways:

- **Ctrl+Click** - During single click, any existing item in the selection list be cleared, and only the item clicked recently is there in the selection list. To avoid cleaning the old selected item, Ctrl key must be on hold when clicking.
- **Selection rectangle/rubber band selection** - Clicking and dragging the diagram area allows to create a rectangular region. The elements that are covered under the rectangular region are selected at the end.



Select/Unselect elements using program

The client-side methods [select](#) and [clearSelection](#) help to select or clear the selection of the elements at runtime. The following code example illustrates how to select or clear the selection of an item using program.

Get the current selected items from the [nodes](#) and [connectors](#) collection of the [selectedItems](#) property of the diagram model.

Select entire elements in diagram programmatically

The client-side method [selectAll](#) used to select all the elements such as nodes/connectors in the diagram. Refer to the following link which shows how to use [selectAll](#) method on the diagram.

Drag

- An object can be dragged by clicking and dragging it. When multiple elements are selected, dragging any one of the selected elements move every selected element.
- When you drag the elements in the diagram, the [positionChange](#) event gets triggered and to do customization in this event.



Resize

- Selector is surrounded by eight thumbs. When dragging these thumbs, selected items can be resized.
- When one corner of the selector is dragged, opposite corner is in a static position.
- When a node is resized, the [sizeChange](#) event gets triggered.



Note: While dragging and resizing, the objects are snapped towards the nearest objects to make better alignments. For better alignments, refer to [Snapping](#).

Rotate

- A rotate handler is placed above the selector. Clicking and dragging the handler in a circular direction lead to rotate the node.
- The node is rotated with reference to the static pivot point.
- Pivot thumb (thumb at the middle of the node) appears while rotating the node to represent the static point.



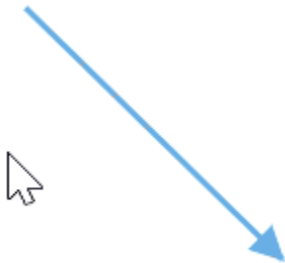
Connection editing

- Each segment of a selected connector is editable with some specific handles/thumbs.

Note: For connector editing, you have to inject the [ConnectorEditing](#) module.

End point handles

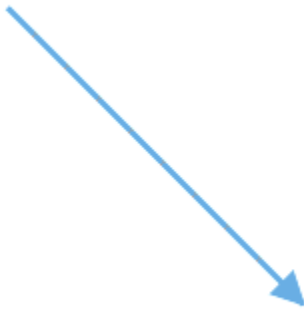
Source and target points of the selected connectors are represented with two handles. Clicking and dragging those handles help you to adjust the source and target points.



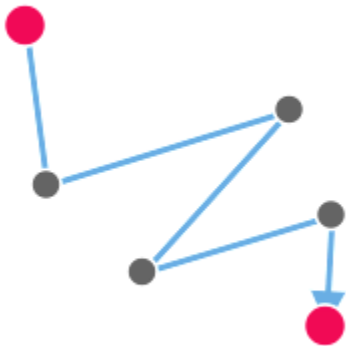
- If you drag the connector end points, then the following events can be used to do your customization.
- When the connector source point is changed, the [sourcePointChange](#) event gets triggered.
- When the connector target point is changed, the [targetPointChange](#) event gets triggered.
- When you connect connector with ports/node or disconnect from it, the [connectionChange](#) event gets triggered.

Straight segment editing

- End point of each straight segment is represented by a thumb that enables to edit the segment.
- Any number of new segments can be inserted into a straight line by clicking, when Shift and Ctrl keys are pressed (Ctrl+Shift+Click).



- Straight segments can be removed by clicking the segment end point, when Ctrl and Shift keys are pressed (Ctrl+Shift+Click).



Orthogonal thumbs

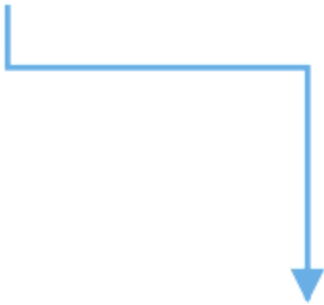
- Orthogonal thumbs allow you to adjust the length of adjacent segments by clicking and dragging it.
- When necessary, some segments are added or removed automatically, when dragging the segment. This is to maintain proper routing of orthogonality between segments.
- The orthogonal segment thumbs can be edited, by injecting the [ConnectorEditing](#) module. The connector constraints has to be set as DragSegmentThumb. The following code example illustrates how to edit an orthogonal segment.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, ViewChild, Inject } from
 '@angular/core';
import { ConnectorModel, Diagram, ConnectorEditing,
ConnectorConstraints, PointModel, DiagramComponent
} from '@syncfusion/ej2-angular-diagrams';
Diagram.Inject(ConnectorEditing)
/**
 * Sample for DrawingTool
 */
@Component({
imports: [
    DiagramModule
],
providers: [ ],
standalone: true,
    selector: 'app-container',
    template: `<ejs-diagram #diagram id="diagram" width="100%"
height="540px" [connectors]="connectors">
</ejs-diagram>`,
    encapsulation: ViewEncapsulation.None
})
export class AppComponent {
    @ViewChild('diagram')
    public diagram?: DiagramComponent;
    ngOnInit(): void {}
    public sourcePoint: PointModel = { x: 100, y: 100 };
    public targetPoint: PointModel = { x: 200, y: 200 };
    public connectors: ConnectorModel[] = [{
        id: 'connector1', type: 'Orthogonal', sourcePoint: this.sourcePoint,
        targetPoint: this.targetPoint,
        constraints: ConnectorConstraints.Default |
ConnectorConstraints.DragSegmentThumb
    }];
}
```

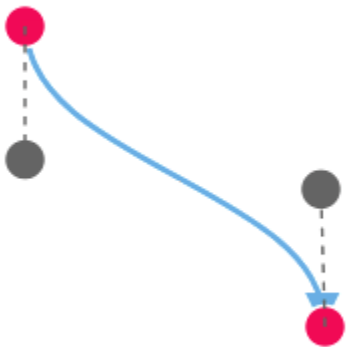
MAIN.TS

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



Bezier thumbs

Bezier segments are annotated with two thumbs to represent the control points. Control points of the curve can be configured by clicking and dragging the control thumbs.



Drag and drop nodes over other elements

Diagram provides support to drop a node/connector over another node/connector. The [drop](#) event is raised to notify that an element is dropped over another one and it is disabled, by default. It can be enabled with the `constraints` property.

User handles

- User handles are used to add some frequently used commands around the selector. To create user handles, define and add them to the [userHandles](#) collection of the [selectedItems](#) property.
- The `name` property of user handle is used to define the name of the user handle and its further used to find the user handle at runtime and do any customization.

Alignment

User handles can be aligned relative to the node boundaries. It has [margin](#), [offset](#), [side](#), [horizontalAlignment](#), and [verticalAlignment](#) settings. It is quite tricky when all four alignments are used together but gives more control over alignment.

Offset

The [offset](#) property of [userHandles](#) is used to align the user handle based on fractions. 0 represents top/left corner, 1 represents bottom/right corner, and 0.5 represents half of width/height.

Side

The [side](#) property of [userHandles](#) is used to align the user handle by using the [Top](#), [Bottom](#), [Left](#), and [Right](#) options.

Horizontal and vertical alignments

The [horizontalAlignment](#) property of [userHandles](#) is used to set how the user handle is horizontally aligned at the position based on the [offset](#). The [verticalAlignment](#) property is used to set how user handle is vertically aligned at the position.

Margin

Margin is an absolute value used to add some blank space in any one of its four sides. The [userHandles](#) can be displaced with the [margin](#) property.

Notification for the mouse button clicked

The diagram component notifies the mouse button clicked. For example, whenever the right mouse button is clicked, the clicked button is notified as right. The mouse click is notified with,

Notification	Description
Left	When the left mouse button is clicked, left is notified
Middle	When the mouse wheel is clicked, middle is notified
Right	When the right mouse button is clicked, right is notified

``typescript`

```
@Component({
  selector: "app-container",
  template: <ejs-diagram id="diagram" width="100%" height="580px"
    (click)="click($event)"></ejs-diagram>,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  ngOnInit(): void {
  }
  public click(args: IClickEventArgs): void {
    // Obtains the mouse button clicked
```

```
var button = args.button
```

```
}
```

```
}
```

```
,
```

Appearance

The appearance of the user handle can be customized by using the [size](#), [borderColor](#), [backgroundColor](#), [visible](#), [pathData](#), and [pathColor](#) properties of the [userHandles](#).

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import { DiagramComponent, SelectorModel, IElement, randomId, cloneObject,
UserHandleModel, SelectorConstraints, ToolBase, NodeModel, Diagram,
MoveTool, ShapeStyleModel, MouseEventArgs, ConnectorModel } from
'@syncfusion/ej2-angular-diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%" height="580px"
[getNodeDefaults]='getNodeDefaults' [getCustomTool]='getCustomTool'
[selectedItems]='selectedItems'>
    <e-nodes>
      <e-node id='node1' [offsetX]=150 [offsetY]=150></e-node>
    </e-nodes>
  </ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram?: DiagramComponent;
  public handles: UserHandleModel[] = [
    {
      name: "clone",
      pathData: "M60.3,18H27.5c-3,0-5.5,2.4-5.5,5.5v38.2h5.5V23.5h32.7V18z
M68.5, 28.9h-30c-3, 0-5.5,2.4-5.5,5.5v38.2c0,3,2.4,5.5,5.5,5.5h30c3,0,5.5-
2.4,5.5-5.5V34.4C73.9,31.4,71.5,28.9,68.5,28.9z M68.5,72.5h-
30V34.4h30V72.5z",
      visible: true,
      offset: 0,
      side: "Bottom",
      margin: { top: 0, bottom: 0, left: 0, right: 0 }
    }
  ];
  public selectedItems: SelectorModel = {
    constraints: SelectorConstraints.UserHandle,
    userHandles: this.handles
  };
};
```



```

public getNodeDefaults(node: NodeModel): NodeModel {
    node.height = 100;
    node.width = 100;
    ((node as NodeModel).style as ShapeStyleModel).fill = "#6BA5D7";
    ((node as NodeModel).style as ShapeStyleModel).strokeColor = "#6BA5D7";
    return node;
}

public getCustomTool: Function = this.getTool.bind(this);
public getTool(action: string): ToolBase {
    let tool: ToolBase = new ToolBase({} as any);
    if (action === "clone") {
        let cloneTool: CloneTool = new CloneTool((this.diagram as
DiagramComponent).commandHandler);
        cloneTool.diagram = this.diagram as Diagram;
        return cloneTool;
    }
    return tool;
}
}

//Defines the clone tool used to copy Node/Connector
class CloneTool extends MoveTool {
    public diagram?: Diagram = undefined;
    public override mouseDown(args: MouseEventArgs): void {
        let newObject: NodeModel | ConnectorModel;
        if (((this.diagram as Diagram).selectedItems.nodes as
NodeModel[]).length > 0) {
            newObject = cloneObject(((this.diagram as Diagram).selectedItems.nodes
as NodeModel[])[0]) as NodeModel;
        } else {
            newObject = cloneObject((((this.diagram as Diagram).selectedItems as
SelectorModel).connectors as ConnectorModel[])[0]) as ConnectorModel;
        }
        newObject.id += randomId();
        (this.diagram as Diagram).paste([newObject]);
        args.source = (this.diagram as Diagram).nodes[(this.diagram as
Diagram).nodes.length - 1] as IElement;
        args.sourceWrapper = args.source.wrapper;
        super.mouseDown(args);
        this.inAction = true;
    }
}
}

```

MAIN.TS

```

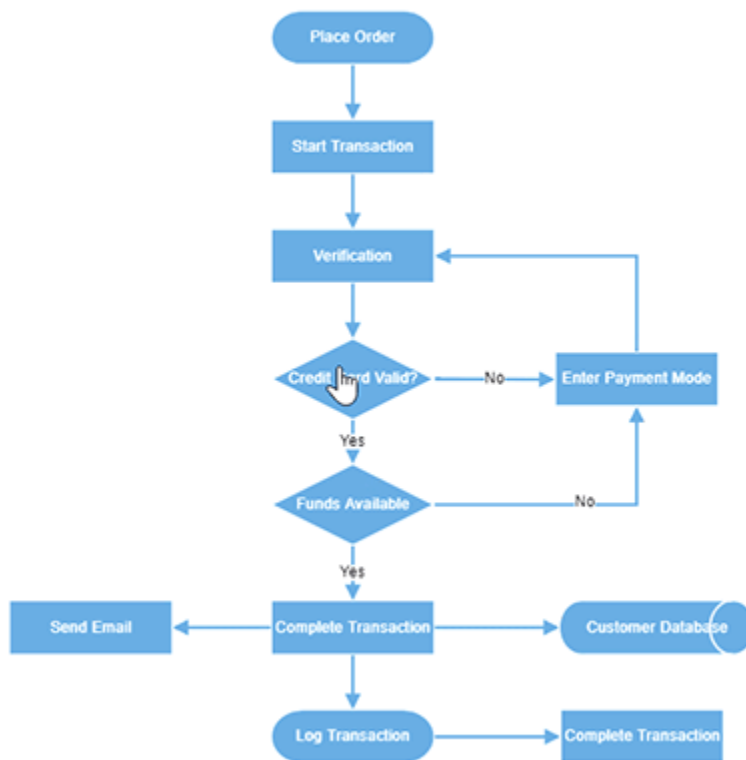
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Zoom pan

- When a large diagram is loaded, only certain portion of the diagram is visible. The remaining portions are clipped. Clipped portions can be explored by scrolling the scrollbars or panning the diagram.

- Diagram can be zoomed in or out by using Ctrl + mouse wheel.
- When the diagram is zoomed or panned, the [scrollChange](#) event gets triggered.



Zoom pan status

Diagram provides the support to notify the pan status of the zoom pan tool. When ever the diagram is panning the [scrollChange](#) event is triggered and hence the pan status can be obtained. The pan status is notified with Start, Progress, and Completed.

Pan Status	Description
Start	When the mouse is clicked and dragged the status is notified as start.
Progress	When the mouse is in motion the status is notified as progress.
Completed	When panning is stopped the status is notified with completed.

```
`typescript
```

```
@Component({
```

```
  selector: "app-container",
```

```
  template: `<ejs-diagram #diagram id="diagram" width="100%" height="580px"
```

```
    [getNodeDefaults]="getNodeDefaults" (created)='created($event)'
```

```
    (scrollChange)='scrollChange($event)'>
```

```
</ejs-diagram>`,
```

```
encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram: DiagramComponent;
  public panStatus: string
  public scrollChange(args: IScrollChangeEventArgs) {
    // Obtains the zoom pan status
    this.panStatus = args.panState;
  }
  public created(args: Object): void {
    //Sets the Zoom pan tool
    this.diagram.tool = DiagramTools.ZoomPan;
    this.diagram.dataBind();
  }
}
```

Keyboard

Diagram provides support to interact with the elements with key gestures. By default, some in-built commands are bound with a relevant set of key combinations.

The following table illustrates those commands with the associated key values.

Shortcut Key	Command	Description
-----	-----	-----
Ctrl + A	selectAll	Select all nodes/connectors in the diagram.
Ctrl + C	copy	Copy the diagram selected elements.
Ctrl + V	paste	Pastes the copied elements.
Ctrl + X	cut	Cuts the selected elements.
Ctrl + Z	undo	Reverses the last editing action performed on the diagram.
Ctrl + Y	redo	Restores the last editing action when no other actions have occurred since the last undo on the diagram.
Delete	delete	Deletes the selected elements.
Ctrl/Shift + Click on object		Multiple selection (Selector binds all selected nodes/connectors).
Up Arrow	nudge("up")	nudgeUp : Moves the selected elements towards up by one pixel.

| Down Arrow | nudge("down") | **nudgeDown**: Moves the selected elements towards down by one pixel. |

| Left Arrow | nudge("left") | **nudgeLeft**: Moves the selected elements towards left by one pixel. |

| Right Arrow | nudge("right") | **nudgeRight**: Moves the selected elements towards right by one pixel. |

| Ctrl + MouseWheel | zoom | Zoom (Zoom in/Zoom out the diagram). |

| F2 | **startLabelEditing** | Starts to edit the label of selected element. |

| Esc | **endLabelEditing** | Sets the label mode as view and stops editing. |

| Tab | Tab to Focus | Select the diagram element based on the rendering order when using the "Tab" key. |

| Shift + Tab | Go to Previous Object | This command is employed to shift the selection to the preceding object based on the z-order. |

| Control + B | Bold | Toggle bold formatting for the selected text. |

| Control + I | Italic | Toggle italic formatting for the selected text. |

| Control + U | Underline | Toggle underline formatting for the selected text. |

| Control + D | Duplicate | Duplicate a selected shape. |

| Control + G | Group | Group together multiple selected shapes, allowing them to be treated as a single shape. |

| Control + Shift + U | UnGroup | Ungroup shapes within a previously grouped selection. |

| Control + R | Rotate clockwise | Rotate the selected nodes in clockwise. |

| Control + L | Rotate anti-clockwise | Rotate the selected nodes in counterclockwise. |

| Control + H | Flip Horizontal | Flip the selected elements horizontally. |

| Control + J | Flip Vertical | Flip the selected elements vertically. |

| Control + 1 | Pointer tool | Activate the pointer tool. |

| Control + 2 | Text tool | Activate the text tool. |

| Control + 3 | Connector tool | Activate the connector tool. |

| Control + 5 | Freeform tool | Activate the freeform tool. |

| Control + 6 | Line tool | Activate the polyline tool. |

| Control + + | Zoom In | Zoom in the diagram. |

| Control + - | Zoom Out | Zoom out the diagram. |

| Shift + Up Arrow | Up | Moves the selected elements towards up by 5 pixel. |

| Shift + Down Arrow | Down | Moves the selected elements towards down by 5 pixel. |

| Shift + Left Arrow | Left | Moves the selected elements towards left by 5 pixel. |

| Shift + Right Arrow | Right | Moves the selected elements towards right by 5 pixel. |

| Control + Shift + L | Align Text Left | Align the selected text to the left. |

- | Control + Shift + C | Center Text Horizontally | Center the selected text horizontally. |
- | Control + Shift + R | Align Text Right | Align the selected text to the right. |
- | Control + Shift + J | Justify Text Horizontally | Justify the selected text, aligning it to both the left and right margins. |
- | Control + Shift + E | Top-align Text Vertically | Align the selected text to the top vertically. |
- | Control + Shift + M | Center Text Vertically | Center the selected text vertically. |
- | Control + Shift + V | Bottom-align Text Vertically | Align the selected text to the bottom vertically. |
- | Control + Shift + B | Send To Back | Send the selected shape backward in the stacking order, making it appear behind other shapes. |
- | Control + Shift + F | Bring To Front | Bring the selected shape forward in the stacking order, making it appear in front of other shapes. |
- | Control + [| Send Backward | Move the selected shape one step backward in the layer order. |
- | Control +] | Bring Forward | Move the selected shape one step forward in the layer order. |

See Also

- [How to create diagram nodes using drawing tools](#)
- [How to create diagram connectors using drawing tools](#)
- [How to disable the diagram interaction](#)
- [How to control the diagram history](#)
- [How to create overview control to the diagram](#)

Tools in Angular Diagram component

Drawing tools

Angular Drawing tool allows you to draw any kind of node/connector during runtime by clicking and dragging on the diagram page.

Shapes

To draw a shape, set the JSON of that shape to the drawType property of the diagram and activate the drawing tool by using the [tool](#) property. The following code example illustrates how to draw a rectangle at runtime.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import { DiagramComponent, Diagram, NodeModel, BasicShapeModel,
DiagramTools, ShapeStyleModel } from '@syncfusion/ej2-angular-diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
```

```

    template: `<ejs-diagram #diagram id="diagram" width="100%"
height="580px" [getNodeDefaults]="getNodeDefaults"
(created)='created($event)'>
    </ejs-diagram>`,
    encapsulation: ViewEncapsulation.None
  })
  export class AppComponent {
    @ViewChild("diagram")
    public diagram?: DiagramComponent;
    public drawingshape?: BasicShapeModel;
    public node?: NodeModel;
    public getNodeDefaults(node: NodeModel): NodeModel {
      node.height = 100;
      node.width = 100;
      ((node as NodeModel).style as ShapeStyleModel).fill = "#6BA5D7";
      ((node as NodeModel).style as ShapeStyleModel).strokeColor =
"White";
      return node;
    }
    public created(args: Object): void {
      //JSON to create a rectangle
      this.drawingshape = { type: 'Basic', shape: 'Rectangle' };
      this.node = {
        shape: this.drawingshape
      };
      (this.diagram as Diagram).drawingObject = this.node;
      //To draw an object once, activate draw once
      (this.diagram as Diagram).tool = DiagramTools.DrawOnce;
      (this.diagram as Diagram).dataBind();
    }
  }
}

```

MAIN.TS

```

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

```

The following code example illustrates how to draw a path.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import { DiagramComponent, Diagram, NodeModel, PathModel, DiagramTools,
ShapeStyleModel } from '@syncfusion/ej2-angular-diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",

```

```

    template: `<ejs-diagram #diagram id="diagram" width="100%"
height="580px" [getNodeDefaults]="getNodeDefaults"
(created)='created($event)'>
    </ejs-diagram>`,
    encapsulation: ViewEncapsulation.None
  })
  export class AppComponent {
    @ViewChild("diagram")
    public diagram?: DiagramComponent;
    public node?: NodeModel;
    public getNodeDefaults(node: NodeModel): NodeModel {
      node.height = 100;
      node.width = 100;
      ((node as NodeModel).style as ShapeStyleModel).fill = "#6BA5D7";
      ((node as NodeModel).style as ShapeStyleModel).strokeColor =
"White";
      return node;
    }
    public created(args: Object): void {
      //JSON to create a path
      this.node = {
        id: "Path",
        style: { fill: "#fbe172" },
        annotations: [{
          content: "Path"
        }],
        shape: {
          type: 'Path',
          data: 'M13.560 67.524 L 21.941 41.731 L 0.000 25.790 L
27.120 25.790 L 35.501 0.000 L 43.882 25.790 L 71.000 25.790 L 49.061 41.731
L 57.441 67.524 L 35.501 51.583 z'
        } as PathModel
      };
      (this.diagram as Diagram).drawingObject = this.node;
      //To draw an object once, activate draw once
      (this.diagram as Diagram).tool = DiagramTools.DrawOnce;
      (this.diagram as Diagram).dataBind();
    }
  }
}

```

MAIN.TS

```

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

```

Connectors

To draw connectors, set the JSON of the connector to the drawType property. The drawing [tool](#) can be activated by using the tool property. The following code example illustrates how to draw a straight line connector.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'

```

```

import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import { DiagramComponent, Diagram, ConnectorModel, DiagramTools, NodeModel,
ShapeStyleModel } from '@syncfusion/ej2-angular-diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%"
height="580px" [getNodeDefaults]="getNodeDefaults"
(created)='created($event)'>
    </ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram?: DiagramComponent;
  public connectors?: ConnectorModel;
  public getNodeDefaults(node: NodeModel): NodeModel {
    node.height = 100;
    node.width = 100;
    ((node as NodeModel).style as ShapeStyleModel).fill = "#6BA5D7";
    ((node as NodeModel).style as ShapeStyleModel).strokeColor =
    "White";
    return node;
  }
  public created(args: Object): void {
    //JSON to create a path
    this.connectors = {
      id: 'connector1',
      type: 'Straight',
      segments: [{ type: "polyline" }]
    } as any as ConnectorModel;
    (this.diagram as Diagram).drawingObject = this.connectors;
    //To draw an object once, activate draw once
    (this.diagram as Diagram).tool = DiagramTools.DrawOnce;
    (this.diagram as Diagram).dataBind();
  }
}

```

MAIN.TS

```

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

```

Text

Diagram allows you to create a textNode, when you click on the diagram page. The following code illustrates how to draw a text.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import { DiagramComponent, Diagram, NodeModel, TextModel, DiagramTools,
ShapeStyleModel } from '@syncfusion/ej2-angular-diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%"
height="580px" [getNodeDefaults]="getNodeDefaults"
(created)="created($event)">
    </ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram?: DiagramComponent;
  public node?: NodeModel;
  public getNodeDefaults(node: NodeModel): NodeModel {
    node.height = 100;
    node.width = 100;
    ((node as NodeModel).style as ShapeStyleModel).fill = "#6BA5D7";
    ((node as NodeModel).style as ShapeStyleModel).strokeColor =
    "White";
    return node;
  }
  public created(args: Object): void {
    //JSON to create a path
    this.node = {
      shape: {
        type: 'Text',
      } as TextModel
    };
    (this.diagram as Diagram).drawingObject = this.node;
    //To draw an object once, activate draw once
    (this.diagram as Diagram).tool = DiagramTools.DrawOnce;
    (this.diagram as Diagram).dataBind();
  }
}

```

MAIN.TS

```

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

```

Once you activate the TextTool, perform label editing of a node/connector.

Polygon

Diagram allows to create the polygon shape by clicking and moving the mouse at runtime on the diagram page.

The following code illustrates how to draw a polygon shape.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import { DiagramComponent, Diagram, NodeModel, BasicShapeModel,
DiagramTools, ShapeStyleModel } from '@syncfusion/ej2-angular-diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%"
height="580px" [getNodeDefaults]="getNodeDefaults"
(created)="created($event)"`>
    </ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram?: DiagramComponent;
  public drawingshape?: BasicShapeModel;
  public node?: NodeModel;
  public getNodeDefaults(node: NodeModel): NodeModel {
    node.height = 100;
    node.width = 100;
    ((node as NodeModel).style as ShapeStyleModel).fill = "#6BA5D7";
    ((node as NodeModel).style as ShapeStyleModel).strokeColor =
    "White";
    return node;
  }
  public created(args: Object): void {
    //JSON to create a rectangle
    this.drawingshape = { type: 'Basic', shape: 'Polygon' };
    this.node = {
      shape: this.drawingshape
    };
    (this.diagram as Diagram).drawingObject = this.node;
    //To draw an object once, activate draw once
    (this.diagram as Diagram).tool = DiagramTools.DrawOnce;
    (this.diagram as Diagram).dataBind();
  }
}
```

MAIN.TS

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

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

Polyline Connector

Diagram allows to create the polyline segments with straight lines and angled vertices at the control points by clicking and moving the mouse at runtime on the diagram page.

The following code illustrates how to draw a polyline connector.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import { DiagramComponent, Diagram, ConnectorModel, DiagramTools,
BasicShapeModel } from '@syncfusion/ej2-angular-diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%"
height="580px" (created)='created($event)'>
</ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram?: DiagramComponent;
  public drawingshape?: BasicShapeModel;
  public connector?: ConnectorModel;
  public created(args: Object): void {
    //JSON to create a rectangle
    this.connector = { id: 'connector1', type: 'Polyline' };
    (this.diagram as Diagram).drawingObject = this.connector;
    //To draw an object once, activate draw once
    (this.diagram as Diagram).tool = DiagramTools.DrawOnce;
    (this.diagram as Diagram).dataBind();
  }
}
```

MAIN.TS

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

Tool selection

There are some functionalities that can be achieved by clicking and dragging on the diagram surface. They are as follows.

- Draw selection rectangle: MultipleSelect tool
- Pan the diagram: Zoom pan
- Draw nodes/connectors: DrawOnce/DrawOnce

As all the three behaviors are completely different, you can achieve only one behavior at a time based on the tool that you choose.

When more than one of those tools are applied, a tool is activated based on the precedence given in the following table.

Precedence	Tools	Description
1st	ContinuesDraw	Allows you to draw the nodes or connectors continuously. Once it is activated, you cannot perform any other interaction in the diagram.
2nd	DrawOnce	Allows you to draw a single node or connector. Once you complete the DrawOnce action, SingleSelect, and MultipleSelect tools are automatically enabled.
3rd	ZoomPan	Allows you to pan the diagram. When you enable both the SingleSelect and ZoomPan tools, you can perform the basic interaction as the cursor hovers node/connector. Panning is enabled when cursor hovers the diagram.
4th	MultipleSelect	Allows you to select multiple nodes and connectors. When you enable both the MultipleSelect and ZoomPan tools, cursor hovers the diagram. When panning is enabled, you cannot select multiple nodes.
5th	SingleSelect	Allows you to select individual nodes or connectors.
6th	None	Disables all tools.

Set the desired [tool](#) to the tool property of the diagram model. The following code illustrates how to enable single/multiple tools.

```
`typescript
```

```
// Enabling Single tool
```

```
@Component({
```

```
  selector: "app-container",
```

```
  template: `<ejs-diagram #diagram id="diagram" width="100%" height="580px">
```

```
</ejs-diagram>`,
```

```
  encapsulation: ViewEncapsulation.None
```

```
})
```

```
export class AppComponent {
```

```
  @ViewChild("diagram")
```

```

public diagram: DiagramComponent;
public tool: DiagramTools
ngOnInit(): void {
this.tool = DiagramTools.DrawOnce
}
}
`typescript
// Enabling multiple tools
@Component({
selector: "app-container",
template: `<ejs-diagram #diagram id="diagram" width="100%" height="580px">
</ejs-diagram>`,
encapsulation: ViewEncapsulation.None
})
export class AppComponent {
@ViewChild("diagram")
public diagram: DiagramComponent;
public tool: DiagramTools
ngOnInit(): void {
// Selects when you click a node and pans when you click the Diagram surface
this.tool = DiagramTools.DrawOnce | DiagramTools.ZoomPan
}
}
`

```

Grid lines in Angular Diagram component

Gridlines are the pattern of lines drawn behind the diagram elements. It provides a visual guidance while dragging or arranging the objects on the diagram surface.

The model's [snapSettings](#) property is used to customize the gridlines and control the snapping behavior in the diagram.

Customize the gridlines visibility

The [snapSettings.snapConstraints](#) enables you to show/hide the gridlines. The following code example illustrates how to show or hide gridlines.

If you need to enable snapping, then inject snapping module into the diagram.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule, SnappingService } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import { DiagramComponent, SnapSettingsModel, SnapConstraints } from
 '@syncfusion/ej2-angular-diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [SnappingService],
  standalone: true,
  selector: "app-container",
  // specifies the template string for the diagram component
  template: `<ejs-diagram #diagram id="diagram" width="100%" height="580px"
[snapSettings]='snapSettings'></ejs-diagram>`
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram?: DiagramComponent;
  public snapSettings: SnapSettingsModel = {
    // Display both Horizontal and Vertical gridlines
    constraints: SnapConstraints.ShowLines
  };
}
```

MAIN.TS

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

To show only horizontal/vertical gridlines or to hide gridlines, refer to [Constraints](#).

Appearance

The appearance of the gridlines can be customized by using a set of predefined properties.

- The [horizontalGridLines](#) and the [verticalGridLines](#) properties allow to customize the appearance of the horizontal and vertical gridlines respectively.
- The horizontal gridlines [lineColor](#) and [lineDashArray](#) properties are used to customize the line color and line style of the horizontal gridlines.
- The vertical gridlines [lineColor](#) and [lineDashArray](#) properties are used to customize the line color and line style of the vertical gridlines.

The following code example illustrates how to customize the appearance of gridlines.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule, SnappingService } from '@syncfusion/ej2-angular-diagrams'
```

```
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import { DiagramComponent, SnapSettingsModel, SnapConstraints, NodeModel }
from '@syncfusion/ej2-angular-diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [SnappingService],
  standalone: true,
  selector: "app-container",
  // specifies the template string for the diagram component
  template: `<ejs-diagram #diagram id="diagram" width="100%" height="580px"
[snapSettings]='snapSettings'></ejs-diagram>`
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram?: DiagramComponent;
  public snapSettings: SnapSettingsModel = {
    // Define the Constraints for gridlines and snapping
    constraints: SnapConstraints.ShowLines,
    // Defines the horizontalGridlines for SnapSettings
    horizontalGridlines: {
      // Sets the line color of gridlines
      lineColor: 'blue',
      // Defines the lineDashArray of gridlines
      lineDashArray: '2 2'
    },
    // Defines the verticalGridlines for SnapSettings
    verticalGridlines: {
      lineColor: 'blue',
      lineDashArray: '2 2'
    }
  };
}
```

MAIN.TS

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

Line intervals

Thickness and the space between gridlines can be customized by using horizontal gridlines's [linesInterval](#) and vertical gridlines's [linesInterval](#) properties. In the lines interval collections, values at the odd places are referred as the thickness of lines and values at the even places are referred as the space between gridlines.

The following code example illustrates how to customize the thickness of lines and the line intervals.

APP.COMPONENT.TS

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

```

import { DiagramModule, SnappingService } from '@syncfusion/ej2-angular-diagrams';
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import { DiagramComponent, SnapSettingsModel, SnapConstraints, NodeModel }
from '@syncfusion/ej2-angular-diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [SnappingService],
  standalone: true,
  selector: "app-container",
  // specifies the template string for the diagram component
  template: `<ejs-diagram #diagram id="diagram" width="100%" height="580px"
[snapSettings]='snapSettings'></ejs-diagram>`
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram?: DiagramComponent;
  public snapSettings: SnapSettingsModel = {
    constraints: SnapConstraints.ShowLines,
    horizontalGridlines: {
      // Sets the lineIntervals of Gridlines
      lineIntervals: [1.25, 14, 0.25, 15, 0.25, 15, 0.25, 15, 0.25,
15],
      lineColor: 'blue',
      lineDashArray: '2 2'
    },
    verticalGridlines: {
      // Sets the lineIntervals of Gridlines
      lineIntervals: [1.25, 14, 0.25, 15, 0.25, 15, 0.25, 15, 0.25,
15],
      lineColor: 'blue',
      lineDashArray: '2 2'
    }
  };
}

```

MAIN.TS

```

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

```

Snapping

Snap to lines

This feature allows the diagram objects to snap to the nearest intersection of gridlines while being dragged or resized. This feature enables easier alignment during layout or design.

Snapping to gridlines can be enabled/disabled with the [snapSettings.snapConstraints](#). The following code example illustrates how to enable/disable the snapping to gridlines.

APP.COMPONENT.TS


```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule, SnappingService } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import { DiagramComponent, SnapSettingsModel, SnapConstraints, NodeModel, ShapeStyleModel } from '@syncfusion/ej2-angular-diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [SnappingService],
  standalone: true,
  selector: "app-container",
  // specifies the template string for the diagram component
  template: `<ejs-diagram #diagram id="diagram" width="100%" height="580px"
[snapSettings]='snapSettings' [getNodeDefaults]='getNodeDefaults'>
    <e-nodes>
      <e-node id='node1' [offsetX]=150 [offsetY]=150></e-node>
    </e-nodes>
  </ejs-diagram>`
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram?: DiagramComponent;
  public snapSettings: SnapSettingsModel = {
    // Enables the object to snap with both horizontal and Vertical
    gridlines
  };
  constraints: SnapConstraints.SnapToLines | SnapConstraints.ShowLines
};
  public getNodeDefaults(node: NodeModel): NodeModel {
    node.height = 100;
    node.width = 100;
    ((node as NodeModel).style as ShapeStyleModel).fill = "#6BA5D7";
    ((node as NodeModel).style as ShapeStyleModel).strokeColor =
    "White";
    return node;
  }
}

```

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 snap intervals

By default, the objects are snapped towards the nearest gridline. The gridline or position towards where the diagram object snaps can be customized with the horizontal gridlines's [snapInterval](#) and the vertical gridlines's [snapInterval](#) properties.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'

```

```

import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule, SnappingService } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import { DiagramComponent, SnapSettingsModel, SnapConstraints, NodeModel, ShapeStyleModel } from '@syncfusion/ej2-angular-diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [SnappingService],
  standalone: true,
  selector: "app-container",
  // specifies the template string for the diagram component
  template: `<ejs-diagram #diagram id="diagram" width="100%" height="580px"
[ snapSettings='snapSettings' [getNodeDefaults]='getNodeDefaults'>
    <e-nodes>
      <e-node id='node1' [offsetX]=150 [offsetY]=150></e-node>
    </e-nodes>
  </ejs-diagram>`
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram?: DiagramComponent;
  public snapSettings: SnapSettingsModel = {
    horizontalGridlines: {
      // Defines the snap interval for object
      snapIntervals: [10]
    },
    verticalGridlines: {
      snapIntervals: [10]
    },
    constraints: SnapConstraints.All
  };
  public getNodeDefaults(node: NodeModel): NodeModel {
    node.height = 100;
    node.width = 100;
    ((node as NodeModel).style as ShapeStyleModel).fill = "#6BA5D7";
    ((node as NodeModel).style as ShapeStyleModel).strokeColor =
    "White";
    return node;
  }
}

```

MAIN.TS

```

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

```

Snap to objects

The snap to object provides visual cues to assist with aligning and spacing diagram elements. A node can be snapped with its neighboring objects based on certain alignments. Such alignments are visually represented as smart guides.

The [snapObjectDistance](#) property allows you to define minimum distance between the selected object and the nearest object.

The [snapAngle](#) property allows you to define the snap angle by which the object needs to be rotated.

The [snapConstraints](#) property allows you to enable or disable the certain features of the snapping, refer to [snapConstraints](#).

The [snapLineColor](#) property allows you to define the color of the snapline.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule, SnappingService } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import { DiagramComponent, SnapSettingsModel, SnapConstraints, NodeModel, ShapeStyleModel } from '@syncfusion/ej2-angular-diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [SnappingService],
  standalone: true,
  selector: "app-container",
  // specifies the template string for the diagram component
  template: `<ejs-diagram #diagram id="diagram" width="100%" height="580px"
[snapSettings]='snapSettings' [getNodeDefaults]='getNodeDefaults'>
    <e-nodes>
      <e-node id='node1' [offsetX]=150 [offsetY]=150></e-node>
      <e-node id='node2' [offsetX]=350 [offsetY]=150></e-node>
    </e-nodes>
  </ejs-diagram>`
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram?: DiagramComponent;
  public snapSettings: SnapSettingsModel = {
    // Enable snap to object constraint
    constraints: SnapConstraints.SnapToObject|SnapConstraints.ShowLines,
    // Sets the Snap object distance
    snapObjectDistance: 10,
    // Snap Angle for object
    snapAngle: 10,
    // Set the Snapline color
    snapLineColor: 'red',
  };
  public getNodeDefaults(node: NodeModel): NodeModel {
    node.height = 100;
    node.width = 100;
    ((node as NodeModel).style as ShapeStyleModel).fill = "#6BA5D7";
    ((node as NodeModel).style as ShapeStyleModel).strokeColor =
    "White";
    return node;
  }
}
```

MAIN.TS

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

Page settings in Angular Diagram component

Page settings enable to customize the appearance, width, and height of the diagram page.

Page size and appearance

- The size and appearance of the diagram pages can be customized with the page settings property.
- The [width](#) and [height](#) properties of page settings define the size of the page and based on the size, the [orientation](#) will be set for the page. In addition to that, the appearance of the page can be customized with [source](#) and set of appearance specific properties.
- The [color](#) property is used to customize the background color and border color of the page.
- The [margin](#) property is used to define the page margin.
- To explore those properties, refer to [Page Settings](#).

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, OnInit, ViewChild } from '@angular/core';
import { DiagramComponent, Diagram, NodeModel, ConnectorModel,
PageSettingsModel, ShapeStyleModel } from '@syncfusion/ej2-angular-diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%"
height="580px" [getNodeDefaults] ='getNodeDefaults'
[getConnectorDefaults]='getConnectorDefaults' [pageSettings]="pageSettings">
  <e-nodes>
    <e-node id='node1' [offsetX]=150 [offsetY]=150>
      <e-node-annotations>
        <e-node-annotation id="label1" content="Rectangle1"
[horizontalAlignment]="horizontalAlignment">
      </e-node-annotation>
    </e-node-annotations>
  </e-node>
  <e-node id='node2' [offsetX]=350 [offsetY]=150>
    <e-node-annotations>
```

```

        <e-node-annotation id="label1" content="Rectangle2"
[horizontalAlignment]="horizontalAlignment">
        </e-node-annotation>
    </e-node-annotations>
    </e-node>
</e-nodes>
<e-connectors>
    <e-connector id='connector' type='Orthogonal' [sourceID]='node1'
[targetID]='node2'>
    </e-connector>
</e-connectors>
</ejs-diagram>`,
    encapsulation: ViewEncapsulation.None
})
export class AppComponent {
    @ViewChild("diagram")
    public diagram?: DiagramComponent;
    public pageSettings?: PageSettingsModel;
    node1: any;
    node2: any;
    horizontalAlignment: any;
    ngOnInit(): void {
        // Defines the pageSettings for the diagram
        this.pageSettings = {
            // Sets the PageOrientation for the diagram to page
            orientation: 'Landscape',
            // Sets the Page Break for diagram
            showPageBreaks: true,
            // Defines the background color and image of diagram
            background: {
                color: 'grey'
            },
            // Sets the width for the Page
            width: 300,
            // Sets the height for the Page
            height: 300,
            // Sets the space to be left between an annotation and its
parent node/connector
            margin: {
                left: 10,
                top: 10,
                bottom: 10
            },
        },
    }
    public getNodeDefaults(node: NodeModel): NodeModel {
        node.height = 100;
        node.width = 100;
        ((node as NodeModel).style as ShapeStyleModel).fill = "#6BA5D7";
        ((node as NodeModel).style as ShapeStyleModel).strokeColor =
"White";
        return node;
    }
    public getConnectorDefaults(obj: ConnectorModel): void {
        obj.style = {
            strokeColor: '#6BA5D7',
            fill: '#6BA5D7',

```

```

        strokeWidth: 2
      }
      obj.targetDecorator = {
        style: {
          fill: '#6BA5D7',
          strokeColor: '#6BA5D7'
        }
      }
    }
  }
}

```

MAIN.TS

```

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

```

Set background image

Stretch and align the background image anywhere over the diagram area. The [source](#) property of [background](#) allows you to set the path of the image. The [scale](#) and the [align](#) properties help to stretch/align the background images.

The following code illustrates how to stretch and align the background image.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, OnInit, ViewChild } from '@angular/core';
import { DiagramComponent, Diagram, NodeModel, ConnectorModel,
PageSettingsModel, ShapeStyleModel } from '@syncfusion/ej2-angular-
diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%"
height="580px" [getNodeDefaults] = 'getNodeDefaults'
[getConnectorDefaults]='getConnectorDefaults' [pageSettings]="pageSettings">
    <e-nodes>
      <e-node id='node1' [offsetX]=150 [offsetY]=150>
        <e-node-annotations>
          <e-node-annotation id="label1" content="Rectangle1"
[horizontalAlignment]="horizontalAlignment">
            </e-node-annotation>
          </e-node-annotations>
        </e-node>
      <e-node id='node2' [offsetX]=350 [offsetY]=150>
        <e-node-annotations>

```

```

        <e-node-annotation id="label1" content="Rectangle2"
[horizontalAlignment]="horizontalAlignment">
        </e-node-annotation>
    </e-node-annotations>
    </e-node>
</e-nodes>
<e-connectors>
    <e-connector id='connector' type='Orthogonal' [sourceID]='node1'
[targetID]='node2'>
    </e-connector>
</e-connectors>
</ejs-diagram>`,
    encapsulation: ViewEncapsulation.None
})
export class AppComponent {
    @ViewChild("diagram")
    public diagram?: DiagramComponent;
    public pageSettings?: PageSettingsModel;
    horizontalAlignment: any;
    node2: any;
    node1: any;
    ngOnInit(): void {
        // Defines the pageSettings for the diagram
        this.pageSettings = {
            orientation: 'Landscape',
            showPageBreaks: true,
            // Defines the background Image source
            background: {
                source:
'https://www.w3schools.com/images/w3schools_green.jpg',
                // Defines the scale values for the background image
                scale: 'Meet',
                // Defines the align values for the background image
                align: 'XMinYMin'
            },
            width: 300,
            height: 300,
            margin: {
                left: 10,
                top: 10,
                bottom: 10
            }
        },
    }
    public getNodeDefaults(node: NodeModel): NodeModel {
        node.height = 100;
        node.width = 100;
        ((node as NodeModel).style as ShapeStyleModel).fill = "#6BA5D7";
        ((node as NodeModel).style as ShapeStyleModel).strokeColor =
"White";
        return node;
    }
    public getConnectorDefaults(obj: ConnectorModel): void {
        obj.style = {
            strokeColor: '#6BA5D7',
            fill: '#6BA5D7',
            strokeWidth: 2

```

```

    }
    obj.targetDecorator = {
      style: {
        fill: '#6BA5D7',
        strokeColor: '#6BA5D7'
      }
    }
  }
}

```

MAIN.TS

```

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

```

Multiple page and page breaks

When multiple page is enabled, the size of the page dynamically increases or decreases in multiples of page width and height and completely fits diagram within the page boundaries. Page breaks is used as a visual guide to see how pages are split into multiple pages.

The [multiplePage](#) and [showPageBreak](#) properties of page settings allow you to enable/disable multiple pages and page breaks respectively.

The following code illustrates how to enable multiple page and page break lines.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, OnInit, ViewChild } from
 '@angular/core';
import { DiagramComponent, Diagram, NodeModel, ConnectorModel,
PageSettingsModel, ShapeStyleModel } from '@syncfusion/ej2-angular-
diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%"
height="580px" [getNodeDefaults] ='getNodeDefaults'
[getConnectorDefaults]='getConnectorDefaults' [pageSettings]="pageSettings">
    <e-nodes>
      <e-node id='node1' [offsetX]=150 [offsetY]=150>
        <e-node-annotations>
          <e-node-annotation id="label1" content="Rectangle1"
[horizontalAlignment]="horizontalAlignment">
            </e-node-annotation>
          </e-node-annotations>
        </e-node>
      <e-node id='node2' [offsetX]=350 [offsetY]=150>

```



```

        <e-node-annotations>
            <e-node-annotation id="label1" content="Rectangle2"
[horizontalAlignment]="horizontalAlignment">
                </e-node-annotation>
            </e-node-annotations>
        </e-node>
    </e-nodes>
    <e-connectors>
        <e-connector id='connector' type='Orthogonal' [sourceID]='node1'
[targetID]='node2'>
            </e-connector>
        </e-connectors>
    </ejs-diagram>`,
    encapsulation: ViewEncapsulation.None
})
export class AppComponent {
    @ViewChild("diagram")
    public diagram?: DiagramComponent;
    public pageSettings?: PageSettingsModel;
    node1: any;
    node2: any;
    horizontalAlignment: any;
    ngOnInit(): void {
        // Defines the pageSettings for the diagram
        this.pageSettings = {
            orientation: 'Landscape',
            // Sets the Multiple page for diagram
            multiplePage: true,
            // Sets the Page Break for diagram
            showPageBreaks: true,
            width: 300,
            height: 300,
            margin: {
                left: 10,
                top: 10,
                bottom: 10
            },
        },
    }
}
    public getNodeDefaults(node: NodeModel): NodeModel {
        node.height = 100;
        node.width = 100;
        ((node as NodeModel).style as ShapeStyleModel).fill = "#6BA5D7";
        ((node as NodeModel).style as ShapeStyleModel).strokeColor =
"White";
        return node;
    }
    public getConnectorDefaults(obj: ConnectorModel): void {
        obj.style = {
            strokeColor: '#6BA5D7',
            fill: '#6BA5D7',
            strokeWidth: 2
        }
        obj.targetDecorator = {
            style: {
                fill: '#6BA5D7',
                strokeColor: '#6BA5D7'
            }
        }
    }
}

```

```

    }
  }
}

```

MAIN.TS

```

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

```

Boundary constraints

The diagram provides support to restrict/customize the interactive region, out of which the elements cannot be dragged, resized, or rotated. The [boundaryConstraints](#) property of page settings allows you to customize the interactive region. To explore the boundary constraints, refer to [Boundary Constraints](#).

The following code example illustrates how to define boundary constraints with respect to the page.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, OnInit, ViewChild } from '@angular/core';
import { DiagramComponent, Diagram, NodeModel, ConnectorModel, PageSettingsModel, ShapeStyleModel } from '@syncfusion/ej2-angular-diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%" height="580px" [getNodeDefaults] ='getNodeDefaults' [getConnectorDefaults]='getConnectorDefaults' [pageSettings]="pageSettings">
    <e-nodes>
      <e-node id='node1' [offsetX]=150 [offsetY]=150>
        <e-node-annotations>
          <e-node-annotation id="label1" content="Rectangle1" [horizontalAlignment]="horizontalAlignment">
            </e-node-annotation>
          </e-node-annotations>
        </e-node>
      <e-node id='node2' [offsetX]=350 [offsetY]=150>
        <e-node-annotations>
          <e-node-annotation id="label1" content="Rectangle2" [horizontalAlignment]="horizontalAlignment">
            </e-node-annotation>
          </e-node-annotations>
        </e-node>
      </e-nodes>
    `
})

```

```

        <e-connectors>
            <e-connector id='connector' type='Orthogonal' [sourceID]='node1'
[targetID]='node2'>
            </e-connector>
        </e-connectors>
    </ejs-diagram>`,
    encapsulation: ViewEncapsulation.None
})
export class AppComponent {
    @ViewChild("diagram")
    public diagram?: DiagramComponent;
    public pageSettings?: PageSettingsModel;
    node1: any;
    node2: any;
    horizontalAlignment: any;
    ngOnInit(): void {
        // Defines the pageSettings for the diagram
        this.pageSettings = {
            // Sets the BoundaryConstraints to page
            boundaryConstraints: 'Page',
            background: {
                color: 'grey'
            },
            width: 400,
            height: 400,
            showPageBreaks: true,
        }
    }
    public getNodeDefaults(node: NodeModel): NodeModel {
        node.height = 100;
        node.width = 100;
        ((node as NodeModel).style as ShapeStyleModel).fill = "#6BA5D7";
        ((node as NodeModel).style as ShapeStyleModel).strokeColor =
"White";
        return node;
    }
    public getConnectorDefaults(obj: ConnectorModel): void {
        obj.style = {
            strokeColor: '#6BA5D7',
            fill: '#6BA5D7',
            strokeWidth: 2
        }
        obj.targetDecorator = {
            style: {
                fill: '#6BA5D7',
                strokeColor: '#6BA5D7'
            }
        }
    }
}

```

MAIN.TS

```

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

```

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

Scroll settings in Angular Diagram component

The diagram can be scrolled by using the vertical and horizontal scrollbars. In addition to the scrollbars, mousewheel can be used to scroll the diagram. Diagram's [scrollSettings](#) enable you to read the current scroll status, view port size, current zoom, and zoom factor. It also allows you to scroll the diagram programmatically.

Get current scroll status

Scroll settings allow you to read the scroll status, [viewPortWidth](#), [viewPortHeight](#), and [currentZoom](#) with a set of properties. To explore those properties, see [Scroll Settings](#).

Define scroll status

Diagram allows you to pan the diagram before loading, so that any desired region of a large diagram is made to view. You can programmatically pan the diagram with the [horizontalOffset](#) and [verticalOffset](#) properties of scroll settings. The following code illustrates how to set pan the diagram programmatically.

In the following example, the vertical scroll bar is scrolled down by 50px and horizontal scroll bar is scrolled to right by 100px.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, OnInit, ViewChild } from
 '@angular/core';
import { DiagramComponent, Diagram, ScrollSettingsModel } from
 '@syncfusion/ej2-angular-diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%"
height="580px" [scrollSettings]="scrollSettings">
</ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram?: DiagramComponent;
  public scrollSettings?: ScrollSettingsModel;
  ngOnInit(): void {
    // Defines the pageSettings for the diagram
    this.scrollSettings = {
      horizontalOffset: 100,
      verticalOffset: 50
    }
  }
}
```

MAIN.TS

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

Update scroll status

You can programmatically change the scroll offsets at runtime by using the client-side method `update`. The following code illustrates how to change the scroll offsets and zoom factor at runtime.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import { DiagramComponent, Diagram } from '@syncfusion/ej2-angular-diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%"
height="580px" (created)='created($event)'>
  </ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram?: DiagramComponent;
  public created(args: Object): void {
    //Updates scroll settings
    (this.diagram as Diagram).scrollSettings.horizontalOffset = 200;
    (this.diagram as Diagram).scrollSettings.verticalOffset = 30;
    (this.diagram as Diagram).dataBind();
  }
}
```

MAIN.TS

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

AutoScroll

Autoscroll feature automatically scrolls the diagram, whenever the node or connector is moved beyond the boundary of the diagram. So that, it is always visible during dragging, resizing, and multiple selection operations. Autoscroll is automatically triggered when any one of the following is done towards the edges of the diagram.

- Node dragging, resizing
- Connection editing
- Rubber band selection
- Label dragging

The diagram client-side event [ScrollChange](#) gets triggered when the autoscroll (scrollbars) is changed and you can do your own customization in this event.

The autoscroll behavior in your diagram can be enabled/disabled by using the [canAutoScroll](#) property of the diagram.

Autoscroll border

The autoscroll border is used to specify the maximum distance between the object and diagram edge to trigger autoscroll. The default value is set as 15 for all sides (left, right, top, and bottom) and it can be changed by using the [autoScrollBorder](#) property of page settings. The following code example illustrates how to set autoscroll border.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, OnInit, ViewChild } from '@angular/core';
import { DiagramComponent, Diagram, NodeModel, ScrollSettingsModel, ShapeStyleModel } from '@syncfusion/ej2-angular-diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%" height="580px" [getNodeDefaults] ='getNodeDefaults' [scrollSettings]="scrollSettings">
    <e-nodes>
      <e-node id='node1' [offsetX]=150 [offsetY]=150>
    </e-node>
    </e-nodes>
  </ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram?: DiagramComponent;
  public scrollSettings?: ScrollSettingsModel;
  ngOnInit(): void {
    // Defines the pageSettings for the diagram
    this.scrollSettings = {
      left: 100,
      right: 100,
      top: 100,
      bottom: 100
    } as ScrollSettingsModel;
  }
}
```

```

    public getNodeDefaults(node: NodeModel): NodeModel {
        node.height = 100;
        node.width = 100;
        ((node as NodeModel).style as ShapeStyleModel).fill = "#6BA5D7";
        ((node as NodeModel).style as ShapeStyleModel).strokeColor =
"White";
        return node;
    }
}

```

MAIN.TS

```

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

```

Scroll limit

The scroll limit allows you to define the scrollable region of the diagram. It includes the following options:

- Allows to scroll in all directions without any restriction.
- Allows to scroll within the diagram content.
- Allows to scroll within the specified scrollable area.
- The [scrollLimit](#) property of scroll settings helps to limit the scrolling.

The scrollSettings [scrollableArea](#) allow to extend the scrollable region that is based on the scroll limit.

The following code example illustrates how to specify the scroll limit.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, OnInit, ViewChild } from
 '@angular/core';
import { DiagramComponent, Diagram, NodeModel, ScrollSettingsModel,
ShapeStyleModel } from '@syncfusion/ej2-angular-diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%"
height="580px" [getNodeDefaults] ='getNodeDefaults'
[scrollSettings]="scrollSettings">
    <e-nodes>
        <e-node id='node1' [offsetX]=150 [offsetY]=150>
        </e-node>
    </e-nodes>
</ejs-diagram>`,

```

```

        encapsulation: ViewEncapsulation.None
    })
    export class AppComponent {
        @ViewChild("diagram")
        public diagram?: DiagramComponent;
        public scrollSettings?: ScrollSettingsModel;
        ngOnInit(): void {
            // Defines the pageSettings for the diagram
            this.scrollSettings = {
                //Sets the scroll limit
                scrollLimit: 'infinity'
            } as any;
        }
        public getNodeDefaults(node: NodeModel): NodeModel {
            node.height = 100;
            node.width = 100;
            ((node as NodeModel).style as ShapeStyleModel).fill = "#6BA5D7";
            ((node as NodeModel).style as ShapeStyleModel).strokeColor =
"White";
            return node;
        }
    }
}

```

MAIN.TS

```

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

```

Scroll padding

The [padding](#) property of scroll settings allows you to extend the scrollable region that is based on the scroll limit.

The following code example illustrates how to set scroll padding to diagram region.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, OnInit, ViewChild } from
 '@angular/core';
import { DiagramComponent, Diagram, NodeModel, ScrollSettingsModel,
MarginModel } from '@syncfusion/ej2-angular-diagrams';
@Component({
    imports: [
        DiagramModule
    ],
    providers: [ ],
    standalone: true,
    selector: "app-container",
    template: `<ejs-diagram #diagram id="diagram" width="100%"
height="580px" [getNodeDefaults] ='getNodeDefaults'
[scrollSettings]="scrollSettings" nodes =nodes>
</ejs-diagram>`,

```



```

        encapsulation: ViewEncapsulation.None
    })
    export class AppComponent {
        public nodes: NodeModel[] = [
            {
                id: 'Start',
                width: 100, height: 100,
                offsetX: 350, offsetY: 350,
                shape: {
                    type: 'Flow',
                    shape: 'Terminator'
                }
            }
        ]
        @ViewChild("diagram")
        public diagram?: DiagramComponent;
        public scrollSettings?: ScrollSettingsModel;
        ngOnInit(): void {
            // Defines the pageSettings for the diagram
            this.scrollSettings = {
                //Sets the scroll limit
                padding: { right: 50, bottom: 50 }
            }
        }
        public getNodeDefaults(node: NodeModel): NodeModel {
            ((node as NodeModel).margin as MarginModel).top = 0;
            ((node as NodeModel).margin as MarginModel).bottom = 0;
            ((node as NodeModel).margin as MarginModel).left = 25;
            ((node as NodeModel).margin as MarginModel).right = 0;
            return node;
        }
    }
}

```

MAIN.TS

```

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

```

Scrollable Area

Scrolling beyond any particular rectangular area can be restricted by using the [scrollableArea](#) property of scroll settings. To restrict scrolling beyond any custom region, set the [scrollLimit](#) as "limited". The following code example illustrates how to customize scrollable area.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, OnInit, ViewChild } from
 '@angular/core';
import { DiagramComponent, Diagram, NodeModel, ScrollSettingsModel,
 ShapeStyleModel } from '@syncfusion/ej2-angular-diagrams';
@Component({
    imports: [

```

```

    DiagramModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%"
height="580px" [getNodeDefaults] ='getNodeDefaults'
[scrollSettings]="scrollSettings">
    <e-nodes>
      <e-node id='node1' [offsetX]=150 [offsetY]=150>
    </e-node>
    </e-nodes>
  </ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram?: DiagramComponent;
  public scrollSettings?: ScrollSettingsModel;
  ngOnInit(): void {
    // Defines the pageSettings for the diagram
    this.scrollSettings = {
      //Sets the scroll limit
      scrollLimit: 'infinity',
      //Sets the scrollable Area
      scrollableArea: {
        x: 0,
        y: 0,
        width: 500,
        height: 500
      }
    } as any;
  }
  public getNodeDefaults(node: NodeModel): NodeModel {
    node.height = 100;
    node.width = 100;
    ((node as NodeModel).style as ShapeStyleModel).fill = "#6BA5D7";
    ((node as NodeModel).style as ShapeStyleModel).strokeColor =
    "White";
    return node;
  }
}

```

MAIN.TS

```

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

```

UpdateViewport

The [updateViewPort](#) method is used to update the diagram page and view size at runtime.

Data binding in Angular Diagram component

- Diagram can be populated with the **nodes** and **connectors** based on the information provided from an external data source.
- Diagram exposes its specific data-related properties allowing you to specify the data source fields from where the node information has to be retrieved from.
- The [dataManager](#) property is used to define the data source either as a collection of objects or as an instance of **DataManager** that needs to be populated in the diagram.
- The [ID](#) property is used to define the unique field of each JSON data.
- The [parentId](#) property is used to defines the parent field which builds the relationship between ID and parent field.
- The [root](#) property is used to define the root node for the diagram populated from the data source.
- To explore those properties, see [DataSourceSettings](#).
- Diagram supports two types of data binding. They are:
 1. Local data
 2. Remote data

Local data

Diagram can be populated based on the user defined JSON data (Local Data) by mapping the relevant data source fields.

To map the user defined JSON data with diagram, configure the fields of [dataSourceSettings](#). The following code example illustrates how to bind local data with the diagram.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule, HierarchicalTreeService, DataBindingService } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation } from '@angular/core';
import { Diagram, NodeModel, ConnectorModel, SnapConstraints, SnapSettingsModel, DiagramTools } from '@syncfusion/ej2-diagrams';
import { DataManager } from '@syncfusion/ej2-data';
import { DecoratorModel, StrokeStyleModel } from '@syncfusion/ej2-angular-diagrams';

let species: object[] = [
  { 'Name': 'Species', 'fillColor': '#3DD94A' },
  { 'Name': 'Plants', 'Category': 'Species' },
  { 'Name': 'Fungi', 'Category': 'Species' },
  { 'Name': 'Lichens', 'Category': 'Species' },
  { 'Name': 'Animals', 'Category': 'Species' },
  { 'Name': 'Mosses', 'Category': 'Plants' },
  { 'Name': 'Ferns', 'Category': 'Plants' },
  { 'Name': 'Gymnosperms', 'Category': 'Plants' },
  { 'Name': 'Dicotyledens', 'Category': 'Plants' },
  { 'Name': 'Monocotyledens', 'Category': 'Plants' },
  { 'Name': 'Invertebrates', 'Category': 'Animals' },
  { 'Name': 'Vertebrates', 'Category': 'Animals' },
  { 'Name': 'Insects', 'Category': 'Invertebrates' },
  { 'Name': 'Molluscs', 'Category': 'Invertebrates' },
  { 'Name': 'Crustaceans', 'Category': 'Invertebrates' },
]
```

```

    { 'Name': 'Others', 'Category': 'Invertebrates' },
    { 'Name': 'Fish', 'Category': 'Vertebrates' },
    { 'Name': 'Amphibians', 'Category': 'Vertebrates' },
    { 'Name': 'Reptiles', 'Category': 'Vertebrates' },
    { 'Name': 'Birds', 'Category': 'Vertebrates' },
    { 'Name': 'Mammals', 'Category': 'Vertebrates' }
  ];
  @Component({
    imports: [
      DiagramModule
    ],
    providers: [HierarchicalTreeService, DataBindingService],
    standalone: true,
    selector: "app-container",
    // specifies the template string for the diagram component
    template: `<ejs-diagram #diagram id="diagram" width="100%" height="490px"
[getConnectorDefaults]='connDefaults' [getNodeDefaults]='nodeDefaults'
[tool]='tool' [layout]='layout' [dataSourceSettings]='data'
[snapSettings]='snapSettings'>
    </ejs-diagram>`
  })
  export class AppComponent {
    public node?: NodeModel;
    public nodeDefaults(node: NodeModel): NodeModel {
      let obj: NodeModel = {};
      obj.shape = { type: 'Basic', shape: 'Rectangle' };
      obj.style = { strokeWidth: 1 };
      obj.width = 95;
      obj.height = 30;
      return obj;
    };
    public data: Object = {
      id: 'Name', parentId: 'Category', dataManager: new
DataManager(species),
      //binds the external data with node
      doBinding: (nodeModel: NodeModel, data: DataInfo, diagram: Diagram)
=> {
        nodeModel.annotations = [{
          /* tslint:disable:no-string-literal */
          content: data['Name'], margin: { top: 10, left: 10, right:
10, bottom: 0 },
          style: { color: 'black' }
        }];
        /* tslint:disable:no-string-literal */
        nodeModel.style = { fill: '#ffeec7', strokeColor: '#f5d897',
strokeWidth: 1 };
      }
    };
    public connDefaults(connector: ConnectorModel): void {
      connector.type = 'Orthogonal';
      ((connector as ConnectorModel).style as
StrokeStyleModel).strokeColor = '#4d4d4d';
      ((connector as ConnectorModel).targetDecorator as
DecoratorModel).shape = 'None';
    };
    public tool: DiagramTools = DiagramTools.ZoomPan;
  }

```

```

    public snapSettings: SnapSettingsModel = { constraints:
SnapConstraints.None };
    public layout: Object = {
        type: 'HierarchicalTree', horizontalSpacing: 40, verticalSpacing:
40,
        margin: { top: 10, left: 10, right: 10, bottom: 0 }
    };
}
export interface DataInfo {
    [key: string]: string;
}

```

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

You can bind the diagram with remote data by using `dataManager`.

It uses two different classes: `DataManager` for processing and `Query` for serving data. `DataManager` communicates with data source and `Query` generates data queries that are read by the [dataManager](#).

To learn more about data manager, refer to [Data Manager](#).

To bind remote data to the diagram, configure the fields of [dataSourceSettings](#). The following code illustrates how to bind remote data to the diagram.

`typescript

```

import { Component, ViewEncapsulation } from '@angular/core';

import { Diagram, NodeModel, ConnectorModel, SnapConstraints, SnapSettingsModel, DiagramTools }
from '@syncfusion/ej2-diagrams';

import { DataManager, Query } from '@syncfusion/ej2-data';

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

    template: `<ejs-diagram #diagram id="diagram" width="100%" height="490px"
[snapSettings]=snapSettings' [getConnectorDefaults]='connDefaults' [getNodeDefaults]='nodeDefaults'
[tool]='tool' [layout]='layout' [dataSourceSettings]='data'>
</ejs-diagram>`,

    encapsulation: ViewEncapsulation.None
})

export class AppComponent {

    public nodeDefaults(obj: NodeModel): NodeModel {

        obj.width = 80;
    }
}

```

```
obj.height = 40;
//Initialize shape
obj.shape = { type: 'Basic', shape: 'Rectangle' };
obj.style = { fill: '#048785', strokeColor: 'Transparent' };
return obj;
};

public data: Object = {
  id: 'EmployeeID', parentId: 'ReportsTo',
  dataManager: new DataManager(
    { url: 'http://mvc.syncfusion.com/Services/Northwnd.svc/', crossDomain: true },
    new Query().from('Employees').select('EmployeeID,ReportsTo,FirstName').take(9),
  ),
  //binds the external data with node
  doBinding: (nodeModel: NodeModel, data: DataInfo, diagram: Diagram) => {
    nodeModel.annotations = [{
      / tslint:disable:no-string-literal /
      content: data['FirstName'],
      style: { color: 'white' }
    }];
  }
};

public connDefaults(connector: ConnectorModel): void {
  connector.type = 'Orthogonal';
  connector.style.strokeColor = '#048785';
  connector.targetDecorator.shape = 'None';
};

public tool: DiagramTools = DiagramTools.ZoomPan;
public snapSettings: SnapSettingsModel = { constraints: SnapConstraints.None };
public layout: Object = {
  type: 'HierarchicalTree', margin: { left: 0, right: 0, top: 100, bottom: 0 },
  verticalSpacing: 40,
  getLayoutInfo: (node: NodeModel, options: TreeInfo) => {
    if (options.level === 3) {
```

```
node.style.fill = '#3c418d';
}
if (options.level === 2) {
node.style.fill = '#108d8d';
options.type = 'Center';
options.orientation = 'Horizontal';
}
if (options.level === 1) {
node.style.fill = '#822b86';
}
}
};
}

export interface EmployeeInfo {
Role: string;
color: string;
}

export interface DataInfo {
[key: string]: string;
}
`
```

CRUD

This feature allows you to read the data source and perform add or edit or delete the data in data source at runtime.

Read DataSource

- This feature allows you to define the nodes and connectors collection in the data source and `connectionDataSource` respectively.
- You can set the data collection in the model's `dataSourceSettings` [dataManager](#) property. The nodes will be generated based on the data specified in the data source.
- You can set the connector collection in the model's `dataSourceSettings` [connectionDataSource](#) property.
- The `dataSourceSettings` `connectionDataSource` [dataManager](#) property is used to set the data source for the connection data source items.
- If you have a data (data will be set in the `dataSource` property) with parent relationship in the database and also defined the connector in the `connectionDataSource` simultaneously, then the connectors set in the `connectionDataSource` will be considered as a priority to render the connector.

- The dataSourceSettings crudAction's [read](#) property specifies the method, which is used to read the data source and its populate the nodes in the diagram.
- The connectionDataSource crudAction's [read](#) specifies the method, which is used to read the data source and its populates the connectors in the diagram.
- The dataSourceSettings's [id](#) and connectionDataSource's [id](#) properties are used to define the unique field of each JSON data.
- The connectionDataSource's [sourceID](#) and [targetID](#) properties are used to set the sourceID and targetID for connection data source item.
- The connectionDataSource's [sourcePointX](#), [sourcePointY](#), [targetPointX](#), and [targetPointY](#) properties are used to define the sourcePoint and targetPoint values for connector from data source.
- The dataSourceSettings crudAction's [customFields](#) property is used to maintain the additional information for nodes.
- Similarly, connectionDataSource's crudAction's [customFields](#) is used to maintain the additional information for connectors.

How to perform Editing at runtime

- The dataSourceSettings crudAction object allows you to define the method, which is used to get the changes done in the data source defined for shapes from the client-side to the server-side.
- Similarly, the connectionDataSource crudAction object allows you to define the method, which is used to get the changes done in the data source defined for connectors from the client-side to the server-side.

InsertData

- The dataSourceSettings crudAction's [create](#) property specifies the method, which is used to get the nodes added from the client-side to the server-side.
- The connectionDataSource crudAction's [create](#) specifies the method, which is used to get the connectors added from the client-side to the server-side.
- The following code example illustrates how to send the newly added or inserted data from the client to server-side.

```
`typescript
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import { DiagramComponent } from '@syncfusion/ej2-angular-diagrams';
@Component({
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%" height="580px"
[dataSourceSettings]='data'>
</ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
```



```

export class AppComponent {
  @ViewChild("diagram")
  public diagram: DiagramComponent;
  public data: Object = {
    crudAction: {
      //Url which triggers the server side AddNodes method
      create: 'https://ej2services.syncfusion.com/development/web-services/api/Crud/AddNodes',
    },
    connectionDataSource: {
      crudAction: {
        //Url which triggers the server side AddConnectors method
        create: 'https://ej2services.syncfusion.com/development/web-services/api/Crud/AddConnectors',
      }
    }
  };
  //Sends the inserted nodes/connectors from client side to the server side through the URL which is
  //specified in server side.
  this.diagram.insertData();
}

```

UpdateData

- The dataSourceSettings crudAction's [update](#) property specifies the method, which is used to get the modified nodes from the client-side to the server-side.
- The connectionDataSource crudAction's [update](#) specifies the method, which is used to get the modified connectors from the client-side to the server-side.
- The following code example illustrates how to send the updated data from the client to the server side.

```

`typescript
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import { DiagramComponent } from '@syncfusion/ej2-angular-diagrams';
@Component({
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%" height="580px"
[dataSourceSettings]='data'>

```

```

</ejs-diagram>`,
encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram: DiagramComponent;
  public data: Object = {
    crudAction: {
      //Url which triggers the server side UpdateNodes method
      update: 'https://ej2services.syncfusion.com/development/web-services/api/Crud/UpdateNodes',
    },
    connectionDataSource: {
      crudAction: {
        //Url which triggers the server side UpdateConnectors method
        update: 'https://ej2services.syncfusion.com/development/web-services/api/Crud/UpdateConnectors',
      }
    }
  };
  //Sends the updated nodes/connectors from client side to the server side through the URL which is
  //specified in server side.
  this.diagram.updateData();
}
`

```

DeleteData

- The dataSourceSettings crudAction's [destroy](#) property specifies the method, which is used to get the deleted nodes from the client-side to the server-side.
- The connectionDataSource crudAction's [destroy](#) specifies the method, which is used to get the deleted connectors from the client-side to the server-side.

```

`typescript
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import { DiagramComponent } from '@syncfusion/ej2-angular-diagrams';
@Component({
  selector: "app-container",

```

```
template: `<ejs-diagram #diagram id="diagram" width="100%" height="580px"
[dataSourceSettings]='data'>
</ejs-diagram>`,
encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram: DiagramComponent;
  public data: Object = {
    crudAction: {
      //Url which triggers the server side DeleteNodes method
      destroy: 'https://ej2services.syncfusion.com/development/web-services/api/Crud/DeleteNodes',
    },
    connectionDataSource: {
      crudAction: {
        //Url which triggers the server side DeleteConnectors method
        destroy: 'https://ej2services.syncfusion.com/development/web-services/api/Crud/DeleteConnectors',
      }
    }
  };
  //Sends the deleted nodes/connectors from client side to the server side through the URL which is
  //specified in server side.
  this.diagram.removeData();
}
```

See Also

- [How to arrange the diagram nodes and connectors using varies layout](#)

Automatic layout in Angular Diagram component

Diagram provides support to auto-arrange the nodes in the diagram area that is referred as **Layout**. It includes the following layout modes:

Layout modes

- Hierarchical layout
- Organization chart

- Radial tree
- Symmetric layout
- Mind Map layout
- Complex hierarchical tree layout

Hierarchical layout

The hierarchical tree layout arranges nodes in a tree-like structure, where the nodes in the hierarchical layout may have multiple parents. There is no need to specify the layout root. To arrange the nodes in a hierarchical structure, specify the layout [type](#) as hierarchical tree.

Note: If you want to use hierarchical tree layout in diagram, you need to inject HierarchicalTree in the diagram.

The following example shows how to arrange the nodes in a hierarchical structure.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule, HierarchicalTreeService, DataBindingService } from
'@syncfusion/ej2-angular-diagrams'
import { Component, OnInit, ViewEncapsulation, ViewChild } from
'@angular/core';
import { DiagramComponent, Diagram, NodeModel, ConnectorModel,
SnapSettingsModel, LayoutModel, DataSourceModel, TextModel, DecoratorModel,
ShapeStyleModel } from '@syncfusion/ej2-angular-diagrams';
import { DataManager, Query } from '@syncfusion/ej2-data';
@Component({
imports: [
    DiagramModule
],
providers: [HierarchicalTreeService, DataBindingService],
standalone: true,
selector: "app-container",
template: `<ejs-diagram #diagram id="diagram" width="100%"
height="580px" [getNodeDefaults]="getNodeDefaults"
[getConnectorDefaults]="getConnectorDefaults" [snapSettings]="snapSettings"
[layout]="layout" [dataSourceSettings]="dataSourceSettings">
</ejs-diagram>`,
encapsulation: ViewEncapsulation.None
})
export class AppComponent {
    @ViewChild("diagram")
    public diagram?: DiagramComponent;
    public snapSettings?: SnapSettingsModel;
    public items?: DataManager;
    public layout?: LayoutModel;
    public dataSourceSettings?: DataSourceModel;
    //Initializes data source
    public data: object[] = [{
        Name: "Steve-Ceo"
    },
    {
        Name: "Kevin-Manager",
        ReportingPerson: "Steve-Ceo"
    },
```

```

        {
            Name: "Peter-Manager",
            ReportingPerson: "Steve-Ceo"
        },
        {
            Name: "John- Manager",
            ReportingPerson: "Peter-Manager"
        },
        {
            Name: "Mary-CSE ",
            ReportingPerson: "Peter-Manager"
        },
        {
            Name: "Jim-CSE ",
            ReportingPerson: "Kevin-Manager"
        },
        {
            Name: "Martin-CSE",
            ReportingPerson: "Kevin-Manager"
        }
    ];
    //Sets the default properties for all the Nodes
    public getNodeDefaults(obj: NodeModel, diagram: Diagram): NodeModel {
        obj.shape = {
            type: 'Text',
            content: (obj.data as {Name: 'string'}).Name
        };
        obj.style = {
            fill: 'None',
            strokeColor: 'none',
            strokeWidth: 2,
            bold: true,
            color: 'white'
        };
        obj.borderColor = 'white';
        obj.width = 100;
        obj.height = 40;
        obj.backgroundColor = '#6BA5D7';
        obj.borderWidth = 1;
        (obj.shape as TextModel).margin = {
            left: 5,
            right: 5,
            top: 5,
            bottom: 5
        };
        return obj;
    }
    //Sets the default properties for all the connectors
    public getConnectorDefaults(connector: ConnectorModel, diagram:
Diagram): ConnectorModel {
        connector.style = {
            strokeColor: '#6BA5D7',
            strokeWidth: 2
        };
        (((connector as ConnectorModel).targetDecorator as
DecoratorModel).style as ShapeStyleModel).fill = '#6BA5D7';
    }

```

```

        (((connector as ConnectorModel).targetDecorator as
DecoratorModel).style as ShapeStyleModel).strokeColor = '#6BA5D7';
        connector.type = 'Orthogonal';
        return connector;
    }
    ngOnInit(): void {
        this.snapSettings = {
            constraints: 0
        }
        this.items = new DataManager(this.data as JSON[], new
Query().take(7));
        //Uses layout to auto-arrange nodes on the Diagram page
        this.layout = {
            //Sets layout type
            type: 'HierarchicalTree'
        }
        //Configures data source for Diagram
        this.dataSourceSettings = {
            id: 'Name',
            parentId: 'ReportingPerson',
            dataSource: this.items
        }
    }
}

```

MAIN.TS

```

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

```

Radial tree layout

The radial tree layout arranges nodes on a virtual concentric circle around a root node. Sub-trees formed by the branching of child nodes are located radially around the child nodes. This arrangement result in an ever-expanding concentric arrangement with radial proximity to the root node indicating the node level in the hierarchy. The layout [root](#) property can be used to define the root node of the layout. When no root node is set, the algorithm automatically considers one of the diagram nodes as the root node.

To arrange nodes in a radial tree structure, set the [type](#) of the layout as **RadialTree**.

Note: If you want to use radial tree layout in diagram, you need to inject DataBinding and RadialTree in the diagram.

The following code illustrates how to arrange the nodes in a radial tree structure.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule, RadialTreeService, DataBindingService } from
'@syncfusion/ej2-angular-diagrams'
import { Component, OnInit, ViewEncapsulation, ViewChild } from
'@angular/core';

```

```

import { DiagramComponent, Diagram, NodeModel, ConnectorModel,
SnapSettingsModel, LayoutModel, DataSourceModel, DecoratorModel,
ShapeStyleModel } from '@syncfusion/ej2-angular-diagrams';
import { DataManager, Query } from '@syncfusion/ej2-data';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [RadialTreeService, DataBindingService],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%"
height="580px" [getNodeDefaults]="getNodeDefaults"
[getConnectorDefaults]="getConnectorDefaults" [snapSettings]="snapSettings"
[layout]="layout" [dataSourceSettings]="dataSourceSettings">
    </ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram?: DiagramComponent;
  public snapSettings?: SnapSettingsModel;
  public items?: DataManager;
  public layout?: LayoutModel;
  public dataSourceSettings?: DataSourceModel;
  //Initializes data source
  public data: object[] = [{
    "Id": 1,
    "Name": "Ana Trujillo",
    "Designation": "Project Manager",
    "RatingColor": "#68C2DE"
  },
  {
    "Id": 2,
    "Name": "Lino Rodri",
    "Designation": "Project Manager",
    "RatingColor": "#68C2DE",
    "ReportingPerson": 1
  },
  {
    "Id": 3,
    "Name": "Philip Cramer",
    "Designation": "Project Manager",
    "RatingColor": "#68C2DE",
    "ReportingPerson": 1
  },
  {
    "Id": 4,
    "Name": "Pedro Afonso",
    "Designation": "Project Manager",
    "RatingColor": "#68C2DE",
    "ReportingPerson": 1
  },
  {
    "Id": 5,
    "Name": "Anto Moreno",
    "Designation": "Project Lead",

```

```

        "RatingColor": "#93B85A",
        "ReportingPerson": 1
    },
    {
        "Id": 6,
        "Name": "Elizabeth Roel",
        "Designation": "Project Lead",
        "RatingColor": "#93B85A",
        "ReportingPerson": 1
    },
    {
        "Id": 7,
        "Name": "Aria Cruz",
        "Designation": "Project Lead",
        "RatingColor": "#93B85A",
        "ReportingPerson": 1
    },
    {
        "Id": 8,
        "Name": "Eduardo Roel",
        "Designation": "Project Lead",
        "RatingColor": "#93B85A",
        "ReportingPerson": 1
    },
    {
        "Id": 9,
        "Name": "Howard Snyd",
        "Designation": "Project Lead",
        "RatingColor": "#68C2DE",
        "ReportingPerson": 1
    },
    {
        "Id": 10,
        "Name": "Daniel Tonini",
        "Designation": "Project Lead",
        "RatingColor": "#93B85A",
        "ReportingPerson": 1
    },
    {
        "Id": 11,
        "Name": "Nardo Batista",
        "Designation": "Project Lead",
        "RatingColor": "#68C2DE",
        "ReportingPerson": 1
    }
];
public getNodeDefaults(obj: NodeModel): NodeModel {
    obj.height = 15;
    obj.width = 15;
    obj.borderWidth = 1;
    obj.style = {
        fill: '#6BA5D7',
        strokeWidth: 2,
        strokeColor: '#6BA5D7'
    };
    return obj;
}

```



```

//Sets the default properties for and connectors
public getConnectorDefaults(connector: ConnectorModel, diagram:
Diagram): ConnectorModel {
    connector.style = {
        strokeColor: '#6BA5D7',
        strokeWidth: 2
    };
    (((connector as ConnectorModel).targetDecorator as
DecoratorModel).style as ShapeStyleModel).fill = '#6BA5D7';
    (((connector as ConnectorModel).targetDecorator as
DecoratorModel).style as ShapeStyleModel).strokeColor = '#6BA5D7';
    (((connector as ConnectorModel).targetDecorator as
DecoratorModel).shape = 'None';
    connector.type = 'Straight';
    return connector;
}
ngOnInit(): void {
    this.snapSettings = {
        constraints: 0
    }
    this.items = new DataManager(this.data as JSON[], new
Query().take(7));
    //Uses layout to auto-arrange nodes on the Diagram page
    this.layout = {
        //set the type as Radial Tree
        type: 'RadialTree',
        root: 'parent'
    }
    //Configures data source for Diagram
    this.dataSourceSettings = {
        id: 'Id',
        parentId: 'ReportingPerson',
        dataSource: this.items
    }
}
}

```

MAIN.TS

```

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

```

Organizational Chart

An organizational chart is a diagram that displays the structure of an organization and relationships. To create an organizational chart, the [type](#) of layout should be set as an `OrganizationalChart`.

The following code example illustrates how to create an organizational chart.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule, HierarchicalTreeService, DataBindingService } from
'@syncfusion/ej2-angular-diagrams'

```

```

import { Component, OnInit, ViewEncapsulation, ViewChild } from
'@angular/core';
import { DiagramComponent, Diagram, NodeModel, ConnectorModel,
SnapSettingsModel, LayoutModel, DataSourceModel, TextModel, DecoratorModel,
ShapeStyleModel } from '@syncfusion/ej2-angular-diagrams';
import { DataManager, Query } from '@syncfusion/ej2-data';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [HierarchicalTreeService, DataBindingService],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%"
height="580px" [getNodeDefaults]="getNodeDefaults"
[getConnectorDefaults]="getConnectorDefaults" [snapSettings]="snapSettings"
[layout]="layout" [dataSourceSettings]="dataSourceSettings">
  </ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram?: DiagramComponent;
  public snapSettings?: SnapSettingsModel;
  public items?: DataManager;
  public layout?: LayoutModel;
  public dataSourceSettings?: DataSourceModel;
  //Initializes data source
  public data: object[] = [{
    Id: "parent",
    Role: "Project Management"
  },
  {
    Id: 1,
    Role: "R&D Team",
    Team: "parent"
  },
  {
    Id: 3,
    Role: "Philosophy",
    Team: "1"
  },
  {
    Id: 4,
    Role: "Organization",
    Team: "1"
  },
  {
    Id: 5,
    Role: "Technology",
    Team: "1"
  },
  {
    Id: 7,
    Role: "Funding",
    Team: "1"
  },
  ],

```

```
{
  Id: 8,
  Role: "Resource Allocation",
  Team: "1"
},
{
  Id: 9,
  Role: "Targeting",
  Team: "1"
},
{
  Id: 11,
  Role: "Evaluation",
  Team: "1"
},
{
  Id: 156,
  Role: "HR Team",
  Team: "parent"
},
{
  Id: 13,
  Role: "Recruitment",
  Team: "156"
},
{
  Id: 113,
  Role: "Training",
  Team: "12"
},
{
  Id: 112,
  Role: "Employee Relation",
  Team: "156"
},
{
  Id: 14,
  Role: "Record Keeping",
  Team: "12"
},
{
  Id: 15,
  Role: "Compensations & Benefits",
  Team: "12"
},
{
  Id: 16,
  Role: "Compliances",
  Team: "12"
},
{
  Id: 17,
  Role: "Production & Sales Team",
  Team: "parent"
},
{
  Id: 119,
```

```

        Role: "Design",
        Team: "17"
    },
    {
        Id: 19,
        Role: "Operation",
        Team: "17"
    },
    {
        Id: 20,
        Role: "Support",
        Team: "17"
    },
    {
        Id: 21,
        Role: "Quality Assurance",
        Team: "17"
    },
    {
        Id: 23,
        Role: "Customer Interaction",
        Team: "17"
    },
    {
        Id: 24,
        Role: "Support and Maintenance",
        Team: "17"
    },
    {
        Id: 25,
        Role: "Task Coordination",
        Team: "17"
    }
];
//Sets the default properties for all the Nodes
public getNodeDefaults(obj: NodeModel, diagram: Diagram): NodeModel {
    obj.shape = {
        type: 'Text',
        content: (obj.data as {
            Role: 'string'
        }).Role
    };
    obj.style = {
        fill: 'None',
        strokeColor: 'none',
        strokeWidth: 2,
        bold: true,
        color: 'white'
    };
    obj.borderColor = 'white';
    obj.backgroundColor = '#6BA5D7';
    obj.borderWidth = 1;
    obj.width = 75;
    obj.height = 40;
    (obj.shape as TextModel).margin = {
        left: 5,
        right: 5,
    }
}

```

```

        top: 5,
        bottom: 5
    };
    return obj;
}
//Sets the default properties for all the connectors
public getConnectorDefaults(connector: ConnectorModel, diagram:
Diagram): ConnectorModel {
    connector.style = {
        strokeColor: '#6BA5D7',
        strokeWidth: 2
    };
    (((connector as ConnectorModel).targetDecorator as
DecoratorModel).style as ShapeStyleModel).fill = '#6BA5D7';
    (((connector as ConnectorModel).targetDecorator as
DecoratorModel).style as ShapeStyleModel).strokeColor = '#6BA5D7';
    connector.type = 'Orthogonal';
    return connector;
}
ngOnInit(): void {
    this.snapSettings = {
        constraints: 0
    }
    this.items = new DataManager(this.data as JSON[], new
Query().take(5));
    //Uses layout to auto-arrange nodes on the Diagram page
    this.layout = {
        //set the type as Organizational Chart
        type: 'OrganizationalChart'
    }
    //Configures data source for Diagram
    this.dataSourceSettings = {
        id: 'Id',
        parentId: 'Team',
        dataSource: this.items
    }
}
}

```

MAIN.TS

```

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

```

Organizational chart layout starts parsing from root and iterate through all its child elements. The `getLayoutInfo` method provides necessary information of a node's children and the way to arrange (direction, orientation, offsets, etc.) them. The arrangements can be customized by overriding this function as explained.

GetLayoutInfo - Set chart orientations, chart types, and offset to be left between parent and child nodes by overriding the method, `diagram.layout.getLayoutInfo`. The [getLayoutInfo](#) method is called to configure every subtree of the organizational chart. It takes the following arguments.

- **node** - Parent node to that options are to be customized.
- **options** - Object to set the customizable properties.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule, HierarchicalTreeService, DataBindingService } from
 '@syncfusion/ej2-angular-diagrams'
import { Component, OnInit, ViewEncapsulation, ViewChild } from
 '@angular/core';
import { DiagramComponent, Diagram, NodeModel, ConnectorModel,
 SnapSettingsModel, LayoutModel, DataSourceModel, DecoratorModel,
 ShapeStyleModel, TreeInfo } from '@syncfusion/ej2-angular-diagrams';
import { DataManager, Query } from '@syncfusion/ej2-data';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [HierarchicalTreeService, DataBindingService],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%"
height="580px" [getNodeDefaults]="getNodeDefaults"
[getConnectorDefaults]="getConnectorDefaults" [snapSettings]="snapSettings"
[layout]="layout" [dataSourceSettings]="dataSourceSettings">
  </ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram?: DiagramComponent;
  public snapSettings?: SnapSettingsModel;
  public items?: DataManager;
  public layout?: LayoutModel;
  public dataSourceSettings?: DataSourceModel;
  //Initializes data source
  public data: object[] = [{
    Id: 1,
    Role: "General Manager"
  },
  {
    Id: 2,
    Role: "Assistant Manager",
    Team: 1
  },
  {
    Id: 3,
    Role: "Human Resource Manager",
    Team: 1
  },
  {
    Id: 4,
    Role: "Design Manager",
    Team: 1
  },
  ],
```

```

    {
        Id: 5,
        Role: "Operation Manager",
        Team: 1
    },
    {
        Id: 6,
        Role: "Marketing Manager",
        Team: 1
    }
];
//Sets the default properties for all the Nodes
public getNodeDefaults(obj: NodeModel | any, diagram: Diagram):
NodeModel {
    obj.width = 150;
    obj.height = 50;
    obj.style.fill = '#6BA5D7';
    obj.annotations = [{
        content: obj.data['Role'],
        style: {
            color: 'white'
        }
    }];
    return obj;
}
//Sets the default properties for all the connectors
public getConnectorDefaults(connector: ConnectorModel, diagram:
Diagram): ConnectorModel {
    connector.style = {
        strokeColor: '#6BA5D7',
        strokeWidth: 2
    };
    (((connector as ConnectorModel).targetDecorator as
DecoratorModel).style as ShapeStyleModel).fill = '#6BA5D7';
    (((connector as ConnectorModel).targetDecorator as
DecoratorModel).style as ShapeStyleModel).strokeColor = '#6BA5D7';
    (((connector as ConnectorModel).targetDecorator as
DecoratorModel).shape = 'None';
    (((connector as ConnectorModel).targetDecorator as
DecoratorModel).shape = 'None';
    connector.type = 'Orthogonal';
    return connector;
}
ngOnInit(): void {
    this.snapSettings = {
        constraints: 0
    }
    this.items = new DataManager(this.data as JSON[], new
Query().take(7));
    //Uses layout to auto-arrange nodes on the Diagram page
    this.layout = {
        //Sets layout type
        type: 'OrganizationalChart',
        getLayoutInfo: (node: Node, options: TreeInfo) => {
            if (!options.hasSubTree) {
                options.type = 'Center';
                options.orientation = 'Horizontal';
            }
        }
    };
}

```

```

    }
  }
}
//Configures data source for Diagram
this.dataSourceSettings = {
  id: 'Id',
  parentId: 'Team',
  dataSource: this.items
}
}
}

```

MAIN.TS

```

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

```

The following table illustrates the properties that “options” argument takes.

Property	Description	Default Value
----------	-------------	---------------

Property	Description	Default Value
----------	-------------	---------------

options.assistants	By default, the collection is empty. When any of the child nodes have to be set as Assistant , you can remove from children collection and have to insert into assistants collection.	Empty array
--------------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	-------------

options.orientation	Gets or sets the organizational chart orientation.	SubTreeOrientation.Vertical
---------------------	----------------------------------------------------	-----------------------------

options.type	Gets or sets the chart organizational chart type.	For horizontal chart orientation:SubTreeAlignments.Center and for vertical chart orientation:SubTreeAlignments.Alternate
--------------	---------------------------------------------------	--------------------------------------------------------------------------------------------------------------------------

options.offset	Offset is the horizontal space to be left between parent and child nodes.	20 pixels applicable only for vertical chart orientations.
----------------	---------------------------------------------------------------------------	------------------------------------------------------------

options.hasSubTree	Gets whether the node contains subtrees.	Boolean
--------------------	------------------------------------------	---------

options.level	Gets the depth of the node from layout root.	Number
---------------	----------------------------------------------	--------

options.enableRouting	By default, connections are routed based on the chart type and orientations. This property gets or sets whether default routing is to be enabled or disabled.	true
-----------------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------	------

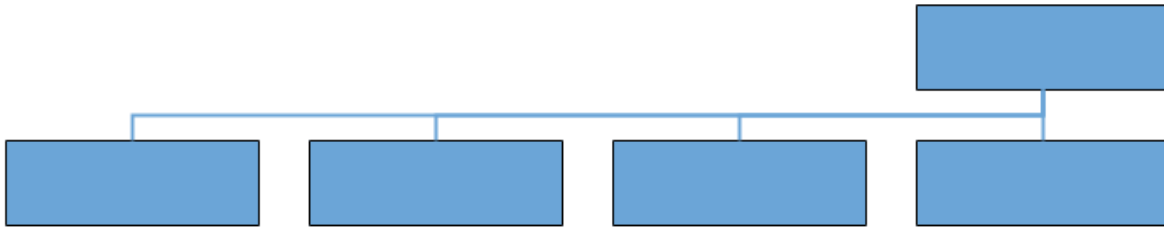
options.rows	Sets the number of rows on which the child nodes will be arranged. Applicable only for balanced type horizontal tree.	Number
--------------	-----------------------------------------------------------------------------------------------------------------------	--------

The following table illustrates the different chart orientations and chart types.

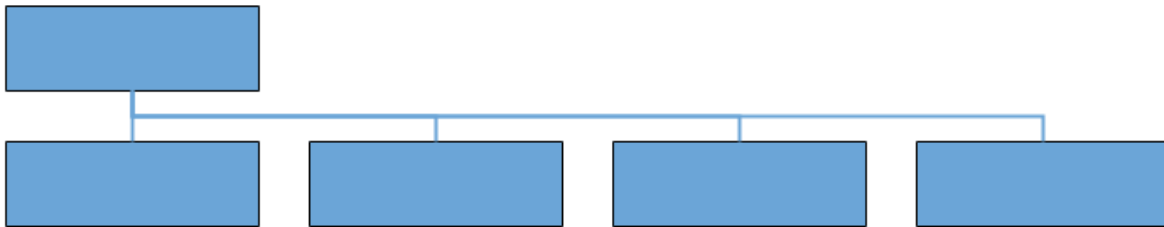
Orientation	Type	Description	Example
-------------	------	-------------	---------

Orientation	Type	Description	Example
-------------	------	-------------	---------

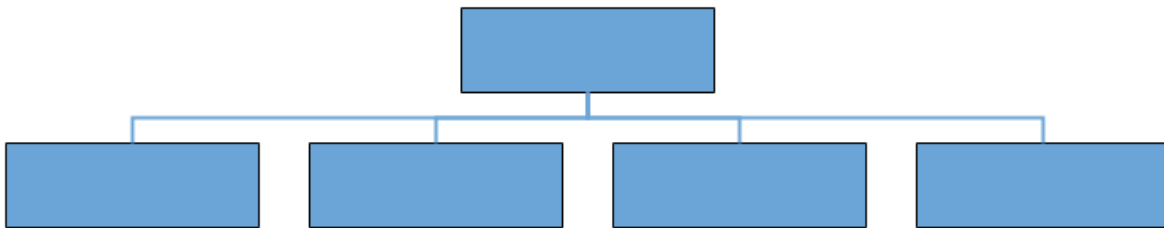
|Horizontal|Left|Arranges the child nodes horizontally at the left side of the parent.|



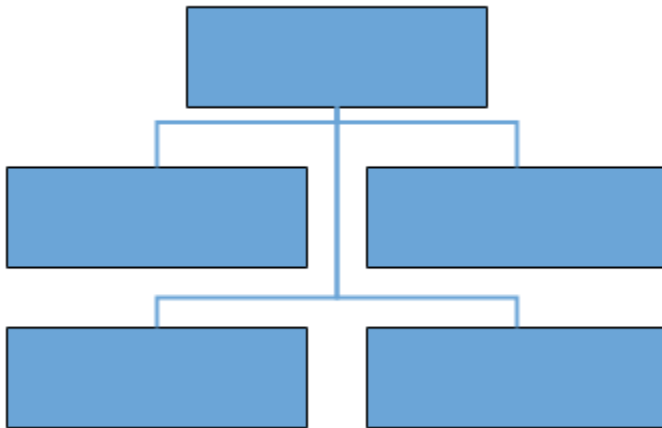
|
|Right|Arranges the child nodes horizontally at the right side of the parent.|



|
|Center|Arranges the children like standard tree layout orientation.|

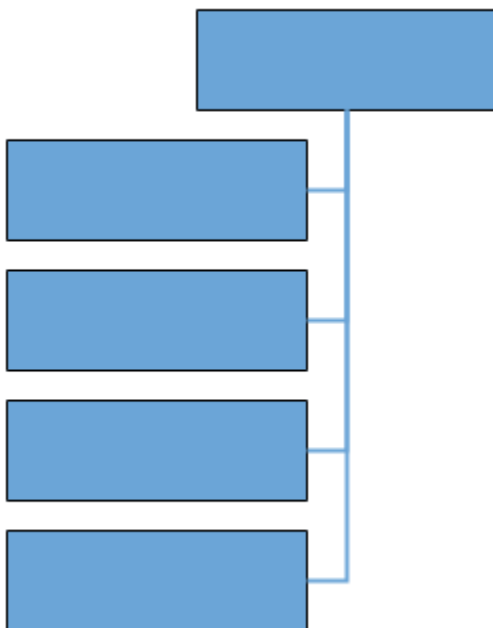


| |Balanced|Arranges the leaf level child nodes in multiple rows.|



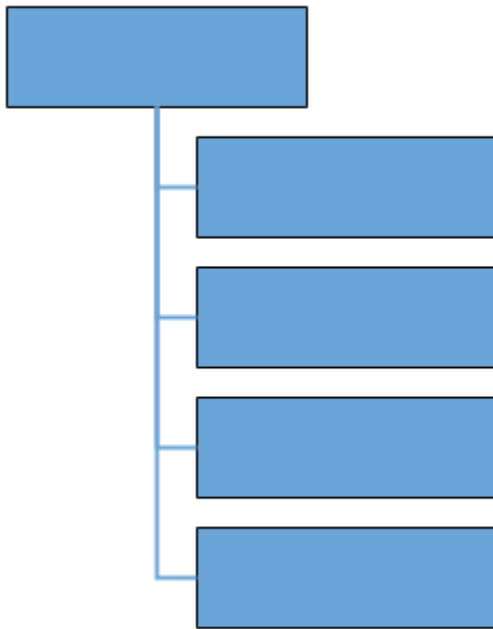
|

|Vertical|Left|Arranges the children vertically at the left side of the parent.|



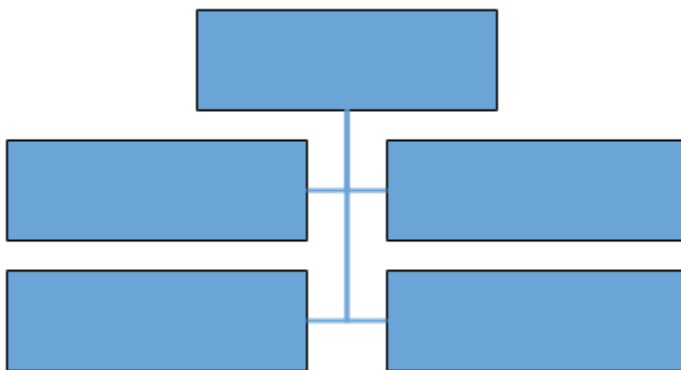
|

||Right|Arranges the children vertically at the right side of the parent.|



|

||Alternate|Arranges the children vertically at both left and right sides of the parent.|



|

The following code example illustrates how to set the vertical right arrangement to the leaf level trees.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule, HierarchicalTreeService, DataBindingService } from
 '@syncfusion/ej2-angular-diagrams'
import { Component, OnInit, ViewEncapsulation, ViewChild } from
 '@angular/core';
import { DiagramComponent, Diagram, NodeModel, ConnectorModel,
 SnapSettingsModel, LayoutModel, DataSourceModel, DecoratorModel,
 ShapeStyleModel, TreeInfo } from '@syncfusion/ej2-angular-diagrams';
import { DataManager, Query } from '@syncfusion/ej2-data';
  
```

```

@Component({
  imports: [
    DiagramModule
  ],
  providers: [HierarchicalTreeService, DataBindingService],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%"
height="580px" [getNodeDefaults]="getNodeDefaults"
[getConnectorDefaults]="getConnectorDefaults" [snapSettings]="snapSettings"
[layout]="layout" [dataSourceSettings]="dataSourceSettings">
    </ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram?: DiagramComponent;
  public snapSettings?: SnapSettingsModel;
  public items?: DataManager;
  public layout?: LayoutModel;
  public dataSourceSettings?: DataSourceModel;
  //Initializes data source
  public data: object[] = [{
    Id: "parent",
    Role: "Board"
  },
  {
    Id: "1",
    Role: "General Manager",
    Manager: "parent"
  },
  {
    Id: "2",
    Role: "Human Resource Manager",
    Manager: "1"
  },
  {
    Id: "3",
    Role: "Trainers",
    Manager: "2"
  },
  {
    Id: "4",
    Role: "Recruiting Team",
    Manager: "2"
  },
  {
    Id: "6",
    Role: "Design Manager",
    Manager: "1"
  },
  {
    Id: "7",
    Role: "Design Supervisor",
    Manager: "6"
  },
  {

```

```

        Id: "8",
        Role: "Development Supervisor",
        Manager: "6"
    },
    {
        Id: "9",
        Role: "Drafting Supervisor",
        Manager: "6"
    },
    {
        Id: "10",
        Role: "Marketing Manager",
        Manager: "1"
    },
    {
        Id: "11",
        Role: "Oversea sales Manager",
        Manager: "10"
    },
    {
        Id: "12",
        Role: "Petroleum Manager",
        Manager: "10"
    },
    {
        Id: "13",
        Role: "Service Dept. Manager",
        Manager: "10"
    },
    ],
    ];
    //Sets the default properties for all the Nodes
    public getNodeDefaults(obj: NodeModel | any, diagram: Diagram):
NodeModel {
        obj.width = 150;
        obj.height = 50;
        obj.borderColor = 'white';
        obj.style.fill = '#6BA5D7';
        obj.borderWidth = 1;
        obj.annotations = [{
            content: obj.data['Role'],
            style: {
                color: 'white'
            }
        }
    ];
    return obj;
}
    //Sets the default properties for all the connectors
    public getConnectorDefaults(connector: ConnectorModel, diagram:
Diagram): ConnectorModel {
        connector.style = {
            strokeColor: '#6BA5D7',
            strokeWidth: 2
        };
        (((connector as ConnectorModel).targetDecorator as
DecoratorModel).style as ShapeStyleModel).fill = '#6BA5D7';
        (((connector as ConnectorModel).targetDecorator as
DecoratorModel).style as ShapeStyleModel).strokeColor = '#6BA5D7';

```

```

        ((connector as ConnectorModel).targetDecorator as
DecoratorModel).shape = 'None';
        connector.type = 'Orthogonal';
        return connector;
    }
    ngOnInit(): void {
        this.snapSettings = {
            constraints: 0
        }
        this.items = new DataManager(this.data as JSON[], new
Query().take(7));
        //Uses layout to auto-arrange nodes on the Diagram page
        this.layout = {
            //Sets layout type
            type: 'OrganizationalChart',
            //Defines getLayoutInfo
            getLayoutInfo: (node: Node | any, options: TreeInfo) => {
                if (node.data['Role'] === 'General Manager') {
                    ((options as TreeInfo).assistants as
string[]).push(((options as TreeInfo).children as string[])[0]);
                    ((options as TreeInfo).children as string[]).splice(0, 1);
                }
                if (!options.hasSubTree) {
                    options.type = 'Right';
                    options.orientation = 'Vertical';
                }
            }
        }
        //Configures data source for Diagram
        this.dataSourceSettings = {
            id: 'Id',
            parentId: 'Manager',
            dataSource: this.items
        }
    }
}

```

MAIN.TS

```

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

```

Assistant

Assistants are child item that have a different relationship with the parent node. They are laid out in a dedicated part of the tree. A node can be specified as an assistant of its parent by adding it to the `assistants` property of the argument “options”.

The following code example illustrates how to add assistants to layout.

APP.COMPONENT.TS

```

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

```

```

import { DiagramModule, HierarchicalTreeService, DataBindingService } from
 '@syncfusion/ej2-angular-diagrams'
import { Component, OnInit, ViewEncapsulation, ViewChild } from
 '@angular/core';
import { DiagramComponent, Diagram, NodeModel, ConnectorModel,
 SnapSettingsModel, LayoutModel, DataSourceModel, DecoratorModel,
 ShapeStyleModel, TreeInfo } from '@syncfusion/ej2-angular-diagrams';
import { DataManager, Query } from '@syncfusion/ej2-data';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [HierarchicalTreeService, DataBindingService],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%"
height="580px" [getNodeDefaults]="getNodeDefaults"
[getConnectorDefaults]="getConnectorDefaults" [snapSettings]="snapSettings"
[layout]="layout" [dataSourceSettings]="dataSourceSettings">
  </ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram?: DiagramComponent;
  public snapSettings?: SnapSettingsModel;
  public items?: DataManager;
  public layout?: LayoutModel;
  public dataSourceSettings?: DataSourceModel;
  //Initializes data source
  public data: object[] = [{
    Id: 1,
    Role: "General Manager"
  },
  {
    Id: 2,
    Role: "Assistant Manager",
    Team: 1
  },
  {
    Id: 3,
    Role: "Human Resource Manager",
    Team: 1
  },
  {
    Id: 4,
    Role: "Design Manager",
    Team: 1
  },
  {
    Id: 5,
    Role: "Operation Manager",
    Team: 1
  },
  {
    Id: 6,
    Role: "Marketing Manager",

```

```

        Team: 1
    }
    ];
    //Sets the default properties for all the Nodes
    public getNodeDefaults(obj: NodeModel | any, diagram: Diagram):
NodeModel {
        obj.width = 150;
        obj.height = 50;
        obj.borderColor = 'white';
        obj.style.fill = '#6BA5D7';
        obj.borderWidth = 1;
        obj.annotations = [{
            content: obj.data['Role'],
            style: {
                color: 'white'
            }
        }
    ]};
    return obj;
}
//Sets the default properties for all the connectors
    public getConnectorDefaults(connector: ConnectorModel, diagram:
Diagram): ConnectorModel {
        connector.style = {
            strokeColor: '#6BA5D7',
            strokeWidth: 2
        };
        (((connector as ConnectorModel).targetDecorator as
DecoratorModel).style as ShapeStyleModel).fill = '#6BA5D7';
        (((connector as ConnectorModel).targetDecorator as
DecoratorModel).style as ShapeStyleModel).strokeColor = '#6BA5D7';
        (((connector as ConnectorModel).targetDecorator as
DecoratorModel).shape = 'None';
        connector.type = 'Orthogonal';
        return connector;
    }
    ngOnInit(): void {
        this.snapSettings = {
            constraints: 0
        }
        this.items = new DataManager(this.data as JSON[], new
Query().take(7));
        //Uses layout to auto-arrange nodes on the Diagram page
        this.layout = {
            //Sets layout type
            type: 'OrganizationalChart',
            // define the getLayoutInfo
            getLayoutInfo: (node: Node | any, options: TreeInfo) => {
                if (node.data['Role'] === 'General Manager') {
                    ((options as TreeInfo).assistants as
string[]).push(((options as TreeInfo).children as string[])[0]);
                    ((options as TreeInfo).children as string[]).splice(0,
1);
                }
                if (!options.hasSubTree) {
                    (options as TreeInfo).type = 'Center';
                    (options as TreeInfo).orientation = 'Horizontal';
                }
            }
        }
    }
}

```



```

    }
  }
  //Configures data source for Diagram
  this.dataSourceSettings = {
    id: 'Id',
    parentId: 'Team',
    dataSource: this.items
  }
}
}

```

MAIN.TS

```

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

```

Symmetric layout

The symmetric layout has been formed using nodes position by closer together or pushing them further apart. This is repeated iteratively until the system comes to an equilibrium state.

The layout's [springLength](#) defined as how long edges should be, ideally. This will be the resting length for the springs. Edge attraction and vertex repulsion forces to be defined by using layout's [springFactor](#), the more sibling nodes repel each other. The relative positions do not change any more from one iteration to the next. The number of iterations can be specified by using layout's [maxIteration](#).

Note: If you want to use symmetric layout in diagram, you need to inject SymmetricLayout in the diagram.

Mind Map layout

A mind map is a diagram that displays the nodes as a spider diagram organizes information around a central concept. To create mind map, the [type](#) of layout should be set as [Link to the Video](#).

To create a Mindmap Layout using Angular Diagram, refer to the below video link,

Tree Orientation in layout

An [Orientation](#) of a `MindMapTreeLayout` is used to arrange the tree layout according to a specific direction. By default, the orientation is set to Horizontal. The following table outlines the various orientation types available:

Orientation Type	Description
Horizontal	Aligns the tree layout from left to right
Vertical	Aligns the tree layout from top to bottom

Note: If you want to use mind map layout in diagram, you need to inject MindMap in the diagram.

The following code example illustrates how to create an mindmap layout

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'

```

```

import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule, MindMapService, DataBindingService } from
 '@syncfusion/ej2-angular-diagrams'
import { Component, OnInit, ViewEncapsulation, ViewChild } from
 '@angular/core';
import { DiagramComponent, DataBinding, Diagram, HierarchicalTree, NodeModel,
 ConnectorModel, MindMap, SnapSettingsModel, LayoutModel, DataSourceModel,
 TextModel, DecoratorModel, ShapeStyleModel } from '@syncfusion/ej2-angular-
 diagrams';
import { DataManager, Query } from '@syncfusion/ej2-data';
Diagram.Inject( MindMap );
@Component({
  imports: [
    DiagramModule
  ],
  providers: [MindMapService, DataBindingService],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%"
height="580px" [getNodeDefaults]="getNodeDefaults"
[getConnectorDefaults]="getConnectorDefaults" [snapSettings]="snapSettings"
[layout]="layout" [dataSourceSettings]="dataSourceSettings">
  </ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram?: DiagramComponent;
  public snapSettings?: SnapSettingsModel;
  public items?: DataManager;
  public layout?: LayoutModel;
  public dataSourceSettings?: DataSourceModel;
  //Initializes data source
  public data: object[] = [{
    id: 1,
    Label: 'StackPanel'
  },
  {
    id: 2,
    Label: 'Label',
    parentId: 1
  },
  {
    id: 3,
    Label: 'ListBox',
    parentId: 1
  },
  {
    id: 4,
    Label: 'StackPanel',
    parentId: 2
  },
  {
    id: 5,
    Label: 'Border',
    parentId: 2
  },
  ],

```

```

        {
            id: 6,
            Label: 'Border',
            parentId: 4
        },
        {
            id: 7,
            Label: 'Button',
            parentId: 4
        },
        {
            id: 8,
            Label: 'ContentPresenter',
            parentId: 5
        },
        {
            id: 9,
            Label: 'Text Block',
            parentId: 5
        },
    ],
};
//Sets the default properties for all the Nodes
public getNodeDefaults(obj: NodeModel, diagram: Diagram): NodeModel {
    obj.shape = {
        type: 'Text',
        content: (obj.data as {
            Label: 'string'
        }).Label,
    };
    obj.style = {
        fill: '#6BA5D7',
        strokeColor: 'none',
        strokeWidth: 2
    };
    obj.borderColor = 'white';
    obj.backgroundColor = '#6BA5D7';
    obj.borderWidth = 1;
    (obj.shape as TextModel).margin = {
        left: 5,
        right: 5,
        top: 5,
        bottom: 5
    };
    return obj;
}
//Sets the default properties for all the connectors
public getConnectorDefaults(connector: ConnectorModel, diagram:
Diagram): ConnectorModel {
    connector.style = {
        strokeColor: '#6BA5D7',
        strokeWidth: 2
    };
    (((connector as ConnectorModel).targetDecorator as
DecoratorModel).style as ShapeStyleModel).fill = '#6BA5D7';
    (((connector as ConnectorModel).targetDecorator as
DecoratorModel).style as ShapeStyleModel).strokeColor = '#6BA5D7';

```

```

        ((connector as ConnectorModel).targetDecorator as
DecoratorModel).shape = 'None';
        connector.type = 'Orthogonal';
        return connector;
    }
    ngOnInit(): void {
        this.snapSettings = {
            constraints: 0
        }
        this.items = new DataManager(this.data as JSON[], new
Query().take(7));
        //Uses layout to auto-arrange nodes on the Diagram page
        this.layout = {
            //Sets layout type
            type: 'MindMap',
            orientation: 'Horizontal'
        }
        //Configures data source for Diagram
        this.dataSourceSettings = {
            id: 'id',
            parentId: 'parentId',
            dataSource: this.items,
            root: String(1)
        }
    }
}

```

MAIN.TS

```

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

```

Complex hierarchical tree

Complex hierarchical tree layout is the extended version of the hierarchical tree layout. The child had been two or more parents. To create a complex hierarchical tree, the [type](#) of layout should be set as `ComplexHierarchicalTree`.

The following code example illustrates how to create a complex hierarchical tree.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule, HierarchicalTreeService, DataBindingService } from
'@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import { DecoratorModel, DiagramComponent, StrokeStyleModel } from
'@syncfusion/ej2-angular-diagrams';
import {
    NodeModel, ConnectorModel, DiagramTools, Diagram, DataBinding,
    ComplexHierarchicalTree,
    SnapConstraints, SnapSettingsModel, LayoutModel, LayoutOrientation
} from '@syncfusion/ej2-diagrams';

```

```

import { DataManager } from '@syncfusion/ej2-data';
import { ChangeEventArgs as NumericChangeEventArgs } from '@syncfusion/ej2-inputs';
Diagram.Inject(DataBinding, ComplexHierarchicalTree);
export interface DataInfo {
    [key: string]: string;
}
/**
 * Sample for Multiple parent sample
 */
@Component({
    imports: [
        DiagramModule
    ],
    providers: [HierarchicalTreeService, DataBindingService],
    standalone: true,
    selector: 'app-container',
    template: `<ejs-diagram #diagram id="diagram" width="100%"
height="580px" [getConnectorDefaults]='connDefaults'
[getNodeDefaults]='nodeDefaults' [tool]='tool' [layout]='layout'
[dataSourceSettings]='data' [snapSettings]='snapSettings'
(created)="created()"></ejs-diagram>`,
    encapsulation: ViewEncapsulation.None
})
export class AppComponent {
    @ViewChild('diagram')
    public diagram?: DiagramComponent;
    public nodeDefaults(obj: NodeModel): NodeModel {
        obj.width = 40; obj.height = 40;
        //Initialize shape
        obj.shape = { type: 'Basic', shape: 'Rectangle', cornerRadius: 7 };
        return obj;
    };
    public data: Object = {
        id: 'Name', parentId: 'ReportingPerson',
        dataSource: new DataManager([
            { "Name": "node11", "fillColor": "#e7704c", "border": "#c15433" },
            { "Name": "node12", "ReportingPerson": ["node114"], "fillColor":
"#efd46e", "border": "#d6b123" },
            { "Name": "node13", "ReportingPerson": ["node12"], "fillColor":
"#58b087", "border": "#16955e" },
            { "Name": "node14", "ReportingPerson": ["node12"], "fillColor":
"#58b087", "border": "#16955e" },
            { "Name": "node15", "ReportingPerson": ["node12"], "fillColor":
"#58b087", "border": "#16955e" },
            { "Name": "node16", "ReportingPerson": [], "fillColor": "#14ad85" },
            { "Name": "node17", "ReportingPerson":
["node13", "node14", "node15"], "fillColor": "#659be5", "border": "#3a6eb5" },
            { "Name": "node18", "ReportingPerson": [], "fillColor": "#14ad85" },
            { "Name": "node19", "ReportingPerson":
["node16", "node17", "node18"], "fillColor": "#8dbe6c", "border": "#489911" },
            { "Name": "node110", "ReportingPerson":
["node16", "node17", "node18"], "fillColor": "#8dbe6c", "border": "#489911" },
            { "Name": "node111", "ReportingPerson":
["node16", "node17", "node18", "node116"], "fillColor": "#8dbe6c", "border":
"#489911" },
            { "Name": "node21", "fillColor": "#e7704c", "border": "#c15433" },

```

```

        {"Name": "node22", "ReportingPerson": ["node114"], "fillColor":
"#efd46e", "border": "#d6b123"},
        {"Name": "node23", "ReportingPerson": ["node22"], "fillColor":
"#58b087", "border": "#16955e"},
        {"Name": "node24", "ReportingPerson": ["node22"], "fillColor":
"#58b087", "border": "#16955e"},
        {"Name": "node25", "ReportingPerson": ["node22"], "fillColor":
"#58b087", "border": "#16955e"},
        {"Name": "node26", "ReportingPerson": [], "fillColor": "#14ad85"},
        {"Name": "node27", "ReportingPerson":
["node23", "node24", "node25"], "fillColor": "#659be5", "border": "#3a6eb5"},
        {"Name": "node28", "ReportingPerson": [], "fillColor": "#14ad85"},
        {"Name": "node29", "ReportingPerson":
["node26", "node27", "node28", "node116"], "fillColor": "#8dbe6c", "border":
"#489911"},
        {"Name": "node210", "ReportingPerson":
["node26", "node27", "node28"], "fillColor": "#8dbe6c", "border": "#489911"},
        {"Name": "node211", "ReportingPerson":
["node26", "node27", "node28"], "fillColor": "#8dbe6c", "border": "#489911"},
        {"Name": "node31", "fillColor": "#e7704c", "border": "#c15433"},
        {"Name": "node114", "ReportingPerson":
["node11", "node21", "node31"], "fillColor": "#f3904a", "border": "#d3722e"},
        {"Name": "node116", "ReportingPerson":
["node12", "node22"], "fillColor": "#58b087", "border": "#16955e"}
    ],),
    //binds the external data with node
    doBinding: (nodeModel: NodeModel, data: DataInfo, diagram: Diagram)
=> {
    /* tslint:disable:no-string-literal */
    nodeModel.style = { fill: data['fillColor'], strokeWidth: 1,
strokeColor: data['border'] };
    }
};
public created(): void {
    (this.diagram as DiagramComponent).fitToPage();
};
public connDefaults(connector: ConnectorModel): void {
    connector.type = 'Orthogonal';
    connector.cornerRadius = 7;
    ((connector as ConnectorModel).targetDecorator as
DecoratorModel).height = 7;
    ((connector as ConnectorModel).targetDecorator as
DecoratorModel).width = 7;
    ((connector as ConnectorModel).style as
StrokeStyleModel).strokeColor = '#6d6d6d';
};
public tool: DiagramTools = DiagramTools.ZoomPan;
public snapSettings: SnapSettingsModel = { constraints:
SnapConstraints.None };
public layout: LayoutModel = {
    type: 'ComplexHierarchicalTree',
    horizontalSpacing: 40, verticalSpacing: 40, orientation:
'TopToBottom',
    margin: { left: 10, right: 0, top: 50, bottom: 0 }
};
}

```

MAIN.TS

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

Note: If you want to use Complex hierarchical layout in diagram, you need to inject ComplexHierarchicalTree in the diagram.

Line Distribution

Line distribution is used to arrange the connectors without overlapping in automatic layout. In some cases, the automatic layout connectors connecting to the nodes will be overlapped with one another. So user can decide whether the segment of each connector from a single parent node should be same point or different point. The [connectionPointOrigin](#) property of layout is used to enable or disable the line distribution in layout. By default connectionPointOrigin will be SamePoint.

The following code example illustrates how to create a complex hierarchical tree with line distribution.

Note: If you want to use line distribution in diagram layout, you need to inject LineDistribution module in the diagram.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule, ComplexHierarchicalTreeService, DataBindingService,
LineDistributionService, LayoutAnimationService } from '@syncfusion/ej2-
angular-diagrams'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import { DecoratorModel, DiagramComponent, ShapeStyleModel } from
 '@syncfusion/ej2-angular-diagrams';
import {
  NodeModel, ConnectorModel, Diagram, DataBinding,
  ComplexHierarchicalTree,
  SnapConstraints, SnapSettingsModel, LayoutModel, LayoutOrientation,
  LineDistribution, ConnectionPointOrigin
} from '@syncfusion/ej2-diagrams';
import { DataManager } from '@syncfusion/ej2-data';
Diagram.Inject(DataBinding, ComplexHierarchicalTree, LineDistribution);
@Component({
  imports: [
    DiagramModule
  ],
  providers: [ComplexHierarchicalTreeService, DataBindingService,
  LineDistributionService, LayoutAnimationService],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-diagram #diagram id="diagram" width="100%"
height="590px" [getConnectorDefaults]='connDefaults'
[getNodeDefaults]='nodeDefaults' [layout]='layout'
[dataSourceSettings]='data' [snapSettings]='snapSettings'
(created)="created()"></ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
```

```

    })
    export class AppComponent {
        @ViewChild('diagram')
        public diagram?: DiagramComponent;
        public nodeDefaults(obj: NodeModel): NodeModel {
            obj.width = 40; obj.height = 40;
            obj.shape = { type: 'Basic', shape: 'Rectangle' };
            obj.style = { fill: '#6BA5D7', strokeColor: 'none', strokeWidth: 2
        };

            obj.borderWidth = 1;
            obj.backgroundColor = '#6BA5D7';
            return obj;
        };
        public data: Object = {
            id: 'Name', parentId: 'ReportingPerson',
            dataSource: new DataManager([
                { "Name": "node11" },
                { "Name": "node12", "ReportingPerson": ["node114"] },
                { "Name": "node13", "ReportingPerson": ["node12"] },
                { "Name": "node14", "ReportingPerson": ["node12"] },
                { "Name": "node15", "ReportingPerson": ["node12"] },
                { "Name": "node16", "ReportingPerson": ["node22", "node12"] },
                { "Name": "node16", "ReportingPerson": [] },
                { "Name": "node18", "ReportingPerson": [] },
                { "Name": "node21" },
                { "Name": "node22", "ReportingPerson": ["node114"] },
                { "Name": "node23", "ReportingPerson": ["node22"] },
                { "Name": "node24", "ReportingPerson": ["node22"] },
                { "Name": "node25", "ReportingPerson": ["node22"] },
                { "Name": "node26", "ReportingPerson": [] },
                { "Name": "node28", "ReportingPerson": [] },
                { "Name": "node31" },
                { "Name": "node114", "ReportingPerson": ["node11", "node21",
"node31"]}
            ],),
        };
        public created(): void {
            (this.diagram as DiagramComponent).fitToPage();
        };
        public connDefaults(connector: ConnectorModel) {
            connector.type = 'Orthogonal';
            connector.cornerRadius = 7;
            ((connector as ConnectorModel).targetDecorator as
DecoratorModel).height = 7;
            ((connector as ConnectorModel).targetDecorator as
DecoratorModel).width = 7;
            connector.style = { strokeColor: '#6BA5D7', strokeWidth: 1 };
            ((connector as ConnectorModel).targetDecorator as
DecoratorModel).style as ShapeStyleModel).fill = '#6BA5D7';
            ((connector as ConnectorModel).targetDecorator as
DecoratorModel).style as ShapeStyleModel).strokeColor = '#6BA5D7';
            return connector;
        };
        public snapSettings: SnapSettingsModel = { constraints:
SnapConstraints.None };
        public layout: LayoutModel = {
            type: 'ComplexHierarchicalTree',

```



```

        connectionPointOrigin: ConnectionPointOrigin.DifferentPoint,
        horizontalSpacing: 40, verticalSpacing: 40, horizontalAlignment:
"Left", verticalAlignment: "Top",
        margin: { left: 0, right: 0, top: 0, bottom: 0 },
        orientation: 'TopToBottom'
    };
}

```

MAIN.TS

```

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

```

Linear Arrangement

Linear arrangement is used to linearly arrange the child nodes in layout, which means the parent node is placed in the center corresponding to its children. When line distribution is enabled, linear arrangement is also activated by default. The [arrangement](#) property of layout is used to enable or disable the linear arrangement in layout. By default arrangement will be **Nonlinear**.

Note: If you want to use linear arrangement in diagram layout, you need to inject `LineDistribution` module in the diagram. Linear arrangement is applicable only for complex hierarchical tree layout.

The following code illustrates how to allow a linear arrangement in diagram layout.

```

`typescript
@Component({
  selector: "app-container",
  template: <ejs-diagram id="diagram" width="100%" height="580px" [layout]='layout'></ejs-diagram>,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  public layout: LayoutModel = {
    type: 'ComplexHierarchicalTree',
    //To arrange the child nodes in a linear manner
    arrangement: ChildArrangement.Linear,
    horizontalSpacing: 40, verticalSpacing: 40,
    orientation: 'TopToBottom',
  };
}
`

```

Prevent connectors overlay

The below constraints prevents the connector segments overlapping nodes with a complex hierarchical layout.

`typescript

```

@Component({
  selector: "app-container",
  template: <ejs-diagram id="diagram" width="100%" height="580px" [layout]='layout'></ejs-diagram>,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  public layout: LayoutModel = {
    //this prevents connector segments overlapping
    enableRouting: true,
  };
}
`

```

Customize layout

Orientation, spacings, and position of the layout can be customized with a set of properties.

To explore layout properties, refer to [Layout Properties](#).

Layout bounds

Diagram provides support to align the layout within any custom rectangular area. For more information about bounds, refer to [bounds](#).

Layout alignment

The layout can be aligned anywhere over the layout bounds/viewport using the [horizontalAlignment](#) and [verticalAlignment](#) properties of the layout.

The following code illustrates how to align the layout at the top-left of the layout bounds.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule, HierarchicalTreeService, DataBindingService } from '@syncfusion/ej2-angular-diagrams'
import { Component, OnInit, ViewEncapsulation, ViewChild } from '@angular/core';
import { DiagramComponent, Diagram, NodeModel, ConnectorModel, SnapSettingsModel, LayoutModel, DataSourceModel, DecoratorModel, ShapeStyleModel } from '@syncfusion/ej2-angular-diagrams';
import { DataManager, Query } from '@syncfusion/ej2-data';
@Component({
  imports: [
    DiagramModule

```

```

    ],
    providers: [HierarchicalTreeService, DataBindingService],
    standalone: true,
    selector: "app-container",
    template: `<ejs-diagram #diagram id="diagram" width="100%"
height="580px" [getNodeDefaults]="getNodeDefaults"
[getConnectorDefaults]="getConnectorDefaults" [snapSettings]="snapSettings"
[layout]="layout" [dataSourceSettings]="dataSourceSettings">
    </ejs-diagram>`,
    encapsulation: ViewEncapsulation.None
  })
}
export class AppComponent {
  @ViewChild("diagram")
  public diagram?: DiagramComponent;
  public snapSettings?: SnapSettingsModel;
  public items?: DataManager;
  public layout?: LayoutModel;
  public dataSourceSettings?: DataSourceModel;
  //Initializes data source
  public data: object[] = [{
    Name: "Steve-Ceo"
  },
  {
    Name: "Kevin-Manager",
    ReportingPerson: "Steve-Ceo"
  },
  {
    Name: "Peter-Manager",
    ReportingPerson: "Kevin-Manager"
  },
  {
    Name: "John- Manager",
    ReportingPerson: "Peter-Manager"
  },
  {
    Name: "Mary-CSE ",
    ReportingPerson: "Peter-Manager"
  }
  ],
  ];
  //Sets the default properties for all the Nodes
  public getNodeDefaults(obj: NodeModel, diagram: Diagram): NodeModel {
    obj.shape = {
      type: 'Text',
      content: (obj.data as {
        Name: 'string'
      }).Name
    };
    obj.style = {
      fill: 'None',
      strokeColor: 'none',
      strokeWidth: 2,
      bold: true,
      color: 'white'
    };
    obj.width = 100;
    obj.height = 40;
    obj.borderColor = 'white';
  }
}

```

```

        obj.backgroundColor = '#6BA5D7';
        obj.borderWidth = 1;
        return obj;
    }
    //Sets the default properties for all the connectors
    public getConnectorDefaults(connector: ConnectorModel, diagram:
Diagram): ConnectorModel {
        connector.style = {
            strokeColor: '#6BA5D7',
            strokeWidth: 2
        };
        (((connector as ConnectorModel).targetDecorator as
DecoratorModel).style as ShapeStyleModel).fill = '#6BA5D7';
        (((connector as ConnectorModel).targetDecorator as
DecoratorModel).style as ShapeStyleModel).strokeColor = '#6BA5D7';
        connector.type = 'Orthogonal';
        return connector;
    }
    ngOnInit(): void {
        this.snapSettings = {
            constraints: 0
        }
        this.items = new DataManager(this.data as JSON[], new
Query().take(7));
        //Uses layout to auto-arrange nodes on the Diagram page
        this.layout = {
            //Sets layout type
            type: 'HierarchicalTree',
            //set layout alignment
            //bounds: new Rect(0, 0, 500, 500),
            horizontalSpacing: 25,
            verticalSpacing: 30,
            horizontalAlignment: 'Left',
            verticalAlignment: 'Top'
        }
        //Configures data source for Diagram
        this.dataSourceSettings = {
            id: 'Name',
            parentId: 'ReportingPerson',
            dataSource: this.items
        }
    }
}

```

MAIN.TS

```

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

```

Layout spacing

Layout provides support to add space horizontally and vertically between the nodes. The [horizontalSpacing](#) and [verticalSpacing](#) properties of the layout allows you to set the space between the nodes in horizontally and vertically.

Layout margin

Layout provides support to add some blank space between the layout bounds/viewport and the layout. The [margin](#) property of the layout allows you to set the blank space.

The following code illustrates how to set the layout margin.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule, HierarchicalTreeService, DataBindingService } from
 '@syncfusion/ej2-angular-diagrams'
import { Component, OnInit, ViewEncapsulation, ViewChild } from
 '@angular/core';
import { DiagramComponent, Diagram, NodeModel, ConnectorModel,
 SnapSettingsModel, LayoutModel, DataSourceModel, DecoratorModel,
 ShapeStyleModel } from '@syncfusion/ej2-angular-diagrams';
import { DataManager, Query } from '@syncfusion/ej2-data';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [HierarchicalTreeService, DataBindingService],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%"
height="580px" [getNodeDefaults]="getNodeDefaults"
[getConnectorDefaults]="getConnectorDefaults" [snapSettings]="snapSettings"
[layout]="layout" [dataSourceSettings]="dataSourceSettings">
  </ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram?: DiagramComponent;
  public snapSettings?: SnapSettingsModel;
  public items?: DataManager;
  public layout?: LayoutModel;
  public dataSourceSettings?: DataSourceModel;
  //Initializes data source
  public data: object[] = [{
    Name: "Steve-Ceo"
  },
  {
    Name: "Kevin-Manager",
    ReportingPerson: "Steve-Ceo"
  },
  {
    Name: "Peter-Manager",
    ReportingPerson: "Kevin-Manager"
  },
  {
    Name: "John- Manager",
    ReportingPerson: "Peter-Manager"
  },
  {
    Name: "Mary-CSE ",
```

```

        ReportingPerson: "Peter-Manager"
    },
];
//Sets the default properties for all the Nodes
public getNodeDefaults(obj: NodeModel, diagram: Diagram): NodeModel {
    obj.shape = {
        type: 'Text',
        content: (obj.data as {
            Name: 'string'
        }).Name
    };
    obj.style = {
        fill: 'None',
        strokeColor: 'none',
        strokeWidth: 2,
        bold: true,
        color: 'white'
    };
    obj.width = 100;
    obj.height = 40;
    obj.borderColor = 'white';
    obj.backgroundColor = '#6BA5D7';
    obj.borderWidth = 1;
    return obj;
}
//Sets the default properties for all the connectors
public getConnectorDefaults(connector: ConnectorModel, diagram:
Diagram): ConnectorModel {
    connector.style = {
        strokeColor: '#6BA5D7',
        strokeWidth: 2
    };
    (((connector as ConnectorModel).targetDecorator as
DecoratorModel).style as ShapeStyleModel).fill = '#6BA5D7';
    (((connector as ConnectorModel).targetDecorator as
DecoratorModel).style as ShapeStyleModel).strokeColor = '#6BA5D7';
    connector.type = 'Orthogonal';
    return connector;
}
ngOnInit(): void {
    this.snapSettings = {
        constraints: 0
    }
    this.items = new DataManager(this.data as JSON[], new
Query().take(7));
    //Uses layout to auto-arrange nodes on the Diagram page
    this.layout = {
        //Sets layout type
        type: 'HierarchicalTree',
        //bounds: new Rect(0, 0, 500, 500),
        horizontalSpacing: 25,
        verticalSpacing: 30,
        horizontalAlignment: 'Left',
        verticalAlignment: 'Top'
    }
    //Configures data source for Diagram
    this.dataSourceSettings = {

```

```

        id: 'Name',
        parentId: 'ReportingPerson',
        dataSource: this.items
    }
}
}

```

MAIN.TS

```

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

```

Layout orientation

The layout orientation can be used to arrange the layout based on the direction. There are different orientation types that are defined in the following table.

Orientation	Description
-----	-----
TopToBottom	Aligns the layout from top to bottom. All the roots are placed at top of diagram.
LeftToRight	Aligns the layout from left to right. All the roots are placed at left of diagram.
BottomToTop	Aligns the layout from bottom to top. All the roots are placed at bottom of the diagram.
RightToLeft	Aligns the layout from right to left. All the roots are placed at right of the diagram.

Diagram provides support to customize the [orientation](#) of layout. You can set the desired orientation using `layout.orientation`.

Note: In the diagram the default orientation is `TopToBottom`.

The following code illustrates how to arrange the nodes in a `BottomToTop` orientation.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { DiagramModule, HierarchicalTreeService, DataBindingService } from '@syncfusion/ej2-angular-diagrams';
import { Component, OnInit, ViewEncapsulation, ViewChild } from '@angular/core';
import { DiagramComponent, Diagram, NodeModel, ConnectorModel, SnapSettingsModel, LayoutModel, DataSourceModel, DecoratorModel, ShapeStyleModel } from '@syncfusion/ej2-angular-diagrams';
import { DataManager, Query } from '@syncfusion/ej2-data';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [HierarchicalTreeService, DataBindingService],
  standalone: true,
  selector: 'app-container',

```

```

    template: `<ejs-diagram #diagram id="diagram" width="100%"
height="580px" [getNodeDefaults]="getNodeDefaults"
[getConnectorDefaults]="getConnectorDefaults" [snapSettings]="snapSettings"
[layout]="layout" [dataSourceSettings]="dataSourceSettings">
    </ejs-diagram>`,
    encapsulation: ViewEncapsulation.None
  })
  export class AppComponent {
    @ViewChild("diagram")
    public diagram?: DiagramComponent;
    public snapSettings?: SnapSettingsModel;
    public items?: DataManager;
    public layout?: LayoutModel;
    public dataSourceSettings?: DataSourceModel;
    //Initializes data source
    public data: object[] = [{
      Name: "Steve-Ceo"
    },
    {
      Name: "Kevin-Manager",
      ReportingPerson: "Steve-Ceo"
    },
    {
      Name: "Peter-Manager",
      ReportingPerson: "Kevin-Manager"
    },
    {
      Name: "John- Manager",
      ReportingPerson: "Peter-Manager"
    },
    {
      Name: "Mary-CSE ",
      ReportingPerson: "Peter-Manager"
    },
  ],
  ];
  //Sets the default properties for all the Nodes
  public getNodeDefaults(obj: NodeModel, diagram: Diagram): NodeModel {
    obj.shape = {
      type: 'Text',
      content: (obj.data as {
        Name: 'string'
      }).Name
    };
    obj.style = {
      fill: 'None',
      strokeColor: 'none',
      strokeWidth: 2,
      bold: true,
      color: 'white'
    };
    obj.width = 100;
    obj.height = 40;
    obj.borderColor = 'white';
    obj.backgroundColor = '#6BA5D7';
    obj.borderWidth = 1;
    return obj;
  }
}

```



```

//Sets the default properties for all the connectors
public getConnectorDefaults(connector: ConnectorModel, diagram:
Diagram): ConnectorModel {
    connector.style = {
        strokeColor: '#6BA5D7',
        strokeWidth: 2
    };
    (((connector as ConnectorModel).targetDecorator as
DecoratorModel).style as ShapeStyleModel).fill = '#6BA5D7';
    (((connector as ConnectorModel).targetDecorator as
DecoratorModel).style as ShapeStyleModel).strokeColor = '#6BA5D7';
    connector.type = 'Orthogonal';
    return connector;
}
ngOnInit(): void {
    this.snapSettings = {
        constraints: 0
    }
    this.items = new DataManager(this.data as JSON[], new
Query().take(7));
    //Uses layout to auto-arrange nodes on the Diagram page
    this.layout = {
        //Sets layout type
        type: 'HierarchicalTree',
        //bounds: new Rect(0, 0, 500, 500),
        horizontalSpacing: 25,
        verticalSpacing: 30,
        horizontalAlignment: 'Left',
        verticalAlignment: 'Top',
        orientation: 'BottomToTop'
    }
    //Configures data source for Diagram
    this.dataSourceSettings = {
        id: 'Name',
        parentId: 'ReportingPerson',
        dataSource: this.items
    }
}
}

```

MAIN.TS

```

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

```

Fixed node

Layout provides support to arrange the nodes with reference to the position of a fixed node and set it to the [fixedNode](#) of the layout property. This is helpful when you try to expand/collapse a node. It might be expected that the position of the double-clicked node should not be changed.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'

```

```

import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule, HierarchicalTreeService, DataBindingService } from
 '@syncfusion/ej2-angular-diagrams'
import { Component, OnInit, ViewEncapsulation, ViewChild } from
 '@angular/core';
import { DiagramComponent, Diagram, NodeModel, ConnectorModel,
 SnapSettingsModel, LayoutModel, DataSourceModel, DecoratorModel,
 ShapeStyleModel } from '@syncfusion/ej2-angular-diagrams';
import { DataManager, Query } from '@syncfusion/ej2-data';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [HierarchicalTreeService, DataBindingService],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%"
height="580px" [getNodeDefaults]="getNodeDefaults"
[getConnectorDefaults]="getConnectorDefaults" [snapSettings]="snapSettings"
[layout]="layout" [dataSourceSettings]="dataSourceSettings">
  </ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram?: DiagramComponent;
  public snapSettings?: SnapSettingsModel;
  public items?: DataManager;
  public layout?: LayoutModel;
  public dataSourceSettings?: DataSourceModel;
  //Initializes data source
  public data: object[] = [{
    Name: "Steve-Ceo",
    //set the offsetX and offsetY for the parent node
    offsetX: 250,
    offsetY: 50
  },
  {
    Name: "Kevin-Manager",
    ReportingPerson: "Steve-Ceo"
  },
  {
    Name: "Peter-Manager",
    ReportingPerson: "Steve-Ceo"
  },
  {
    Name: "John- Manager",
    ReportingPerson: "Peter-Manager"
  },
  {
    Name: "Mary-CSE ",
    ReportingPerson: "Peter-Manager"
  },
  {
    Name: "Jim-CSE ",
    ReportingPerson: "Kevin-Manager"
  }
  ],

```

```

        {
            Name: "Martin-CSE",
            ReportingPerson: "Kevin-Manager"
        }
    ];
    //Sets the default properties for all the Nodes
    public getNodeDefaults(obj: NodeModel, diagram: Diagram): NodeModel {
        obj.shape = {
            type: 'Text',
            content: (obj.data as {
                Name: 'string'
            }).Name
        };
        obj.style = {
            fill: 'None',
            strokeColor: 'none',
            strokeWidth: 2,
            bold: true,
            color: 'white'
        };
        obj.width = 100;
        obj.height = 40;
        obj.borderColor = 'white';
        obj.backgroundColor = '#6BA5D7';
        obj.borderWidth = 1;
        return obj;
    }
    //Sets the default properties for all the connectors
    public getConnectorDefaults(connector: ConnectorModel, diagram:
Diagram): ConnectorModel {
        connector.style = {
            strokeColor: '#6BA5D7',
            strokeWidth: 2
        };
        (((connector as ConnectorModel).targetDecorator as
DecoratorModel).style as ShapeStyleModel).fill = '#6BA5D7';
        (((connector as ConnectorModel).targetDecorator as
DecoratorModel).style as ShapeStyleModel).strokeColor = '#6BA5D7';
        connector.type = 'Orthogonal';
        return connector;
    }
    ngOnInit(): void {
        this.snapSettings = {
            constraints: 0
        }
        this.items = new DataManager(this.data as JSON[], new
Query().take(7));
        //Uses layout to auto-arrange nodes on the Diagram page
        this.layout = {
            type: 'HierarchicalTree',
            //bounds: new Rect(0, 0, 500, 500),
            horizontalSpacing: 25,
            verticalSpacing: 30,
            horizontalAlignment: 'Left',
            verticalAlignment: 'Top'
        }
        //Configures data source for Diagram

```

```

        this.dataSourceSettings = {
            id: 'Name',
            parentId: 'ReportingPerson',
            dataSource: this.items
        }
    }
}

```

MAIN.TS

```

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

```

Expand and collapse

Diagram allows to expand/collapse the subtrees of a layout. The node's `isExpanded` property allows you to expand/collapse its children. The following code example shows how to expand/collapse the children of a node.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule, HierarchicalTreeService, DataBindingService,
LayoutAnimationService } from '@syncfusion/ej2-angular-diagrams'
import { Component, OnInit, ViewEncapsulation, ViewChild } from
'@angular/core';
import { DiagramComponent, Diagram, NodeModel, ConnectorModel,
SelectorModel, SelectorConstraints, SnapSettingsModel, LayoutModel,
DataSourceModel, DecoratorModel, ShapeStyleModel, TreeInfo } from
'@syncfusion/ej2-angular-diagrams';
import { DataManager, Query } from '@syncfusion/ej2-data';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [HierarchicalTreeService, DataBindingService,
LayoutAnimationService],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%"
height="580px" [getNodeDefaults]="getNodeDefaults"
[getConnectorDefaults]="getConnectorDefaults" [snapSettings]="snapSettings"
[selectedItems]="selectedItems" [layout]="layout"
[dataSourceSettings]="dataSourceSettings">
    </ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram?: DiagramComponent;
  public selectedItems?: SelectorModel;
  public snapSettings?: SnapSettingsModel;
  public items?: DataManager;

```

```
public layout?: LayoutModel;
public dataSourceSettings?: DataSourceModel;
//Initializes data source
public data: object[] = [{
    'Id': 'parent1',
    'Name': 'Maria ',
    'Designation': 'Managing Director',
    'RatingColor': '#C34444'
},
{
    'Id': 'parent',
    'Name': ' sam',
    'Designation': 'Managing Director',
    'ReportingPerson': 'parent1',
    'RatingColor': '#C34444'
},
{
    'Id': 'parent3',
    'Name': ' sam geo',
    'Designation': 'Managing Director',
    'ReportingPerson': 'parent1',
    'RatingColor': '#C34444'
},
{
    'Id': '80',
    'Name': ' david',
    'Designation': 'Managing Director',
    'ReportingPerson': 'parent3',
    'RatingColor': '#C34444'
},
{
    'Id': '82',
    'Name': ' pirlo',
    'Designation': 'Managing Director',
    'ReportingPerson': 'parent',
    'RatingColor': '#C34444'
}
];
//Sets the default properties for all the Nodes
public getNodeDefaults(obj: NodeModel | any, diagram: Diagram):
NodeModel {
    obj.expandIcon = {
        height: 15,
        width: 15,
        shape: "Plus",
        fill: 'lightgray',
        offset: {
            x: .5,
            y: .85
        }
    };
    obj.collapseIcon.offset = {
        x: .5,
        y: .85
    };
    obj.collapseIcon.height = 15;
    obj.collapseIcon.width = 15;
```

```

        obj.collapseIcon.shape = "Minus";
        obj.height = 50;
        obj.borderColor = 'white';
        obj.backgroundColor = '#6BA5D7';
        obj.borderWidth = 1;
        obj.style = {
            fill: 'transparent',
            strokeWidth: 2
        };
        return obj;
    }
    //Sets the default properties for all the connectors
    public getConnectorDefaults(connector: ConnectorModel, diagram:
Diagram): ConnectorModel {
        connector.style = {
            strokeColor: '#6BA5D7',
            strokeWidth: 2
        };
        (((connector as ConnectorModel).targetDecorator as
DecoratorModel).style as ShapeStyleModel).fill = '#6BA5D7';
        (((connector as ConnectorModel).targetDecorator as
DecoratorModel).style as ShapeStyleModel).strokeColor = '#6BA5D7';
        (((connector as ConnectorModel).targetDecorator as
DecoratorModel).shape = 'None';
        connector.type = 'Orthogonal';
        return connector;
    }
    ngOnInit(): void {
        this.selectedItems = {
            constraints: ~SelectorConstraints.ResizeAll
        }
        this.snapSettings = {
            constraints: 0
        }
        this.items = new DataManager(this.data as JSON[], new
Query().take(7));
        //Uses layout to auto-arrange nodes on the Diagram page
        this.layout = {
            // set enableAnimation as true
            enableAnimation: true,
            type: 'OrganizationalChart',
            margin: {
                top: 20
            }, // define the getLayoutInfo
            getLayoutInfo: (node: Node, tree: TreeInfo | any) => {
                if (!tree.hasSubTree) {
                    tree.orientation = 'vertical';
                    tree.type = 'alternate';
                }
            }
        }
        //Configures data source for Diagram
        this.dataSourceSettings = {
            id: 'Id',
            parentId: 'ReportingPerson',
            dataSource: this.items
        }
    }

```

```
}
}
```

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 previous example, while expanding/collapsing a node, it is set as fixed node in order to prevent it from repositioning.

Refresh layout

Diagram allows to refresh the layout at runtime. To refresh the layout, refer to [Refresh layout](#).

setNodeTemplate

The [setNodeTemplate](#) function is provided for the purpose of customizing nodes. It will be called for each node on node initialization. In this function, the node style and its properties can be customized and can bind the custom JSON with node.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule, HierarchicalTreeService, DataBindingService } from
 '@syncfusion/ej2-angular-diagrams'
import { Component, OnInit, ViewEncapsulation, ViewChild } from
 '@angular/core';
import { DiagramComponent, Diagram, NodeModel, ConnectorModel,
 SnapSettingsModel, LayoutModel, DataSourceModel, TextModel, DecoratorModel,
 ShapeStyleModel } from '@syncfusion/ej2-angular-diagrams';
import { DataManager, Query } from '@syncfusion/ej2-data';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [HierarchicalTreeService, DataBindingService],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%"
 height="580px" [getNodeDefaults]="getNodeDefaults"
 [getConnectorDefaults]="getConnectorDefaults" [snapSettings]="snapSettings"
 [layout]="layout" [dataSourceSettings]="dataSourceSettings"
 [setNodeTemplate]="setNodeTemplate">
    </ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram?: DiagramComponent;
  public snapSettings?: SnapSettingsModel;
  public items?: DataManager;
  public layout?: LayoutModel;
  public dataSourceSettings?: DataSourceModel;
  //Initializes data source
```

```

public data: object[] = [{
    Name: "Steve-Ceo"
},
{
    Name: "Kevin-Manager",
    ReportingPerson: "Steve-Ceo"
},
{
    Name: "Peter-Manager",
    ReportingPerson: "Kevin-Manager"
},
{
    Name: "John- Manager",
    ReportingPerson: "Peter-Manager"
},
{
    Name: "Mary-CSE ",
    ReportingPerson: "Peter-Manager"
},
];
//Sets the default properties for all the Nodes
public getNodeDefaults(obj: NodeModel, diagram: Diagram): NodeModel {
    obj.shape = {
        type: 'Text',
        content: (obj.data as {
            Name: 'string'
        }).Name
    };
    obj.style = {
        fill: 'None',
        strokeColor: 'none',
        strokeWidth: 2,
        bold: true,
        color: 'white'
    };
    obj.width = 50;
    obj.height = 40;
    obj.borderColor = 'white';
    obj.backgroundColor = '#6BA5D7';
    obj.borderWidth = 1;
    (obj.shape as TextModel).margin = {
        left: 25,
        right: 25,
        top: 25,
        bottom: 25
    };
    return obj;
}
//Sets the default properties for all the connectors
public getConnectorDefaults(connector: ConnectorModel, diagram:
Diagram): ConnectorModel {
    connector.style = {
        strokeColor: '#6BA5D7',
        strokeWidth: 2
    };
    (((connector as ConnectorModel).targetDecorator as
DecoratorModel).style as ShapeStyleModel).fill = '#6BA5D7';

```



```

        (((connector as ConnectorModel).targetDecorator as
DecoratorModel).style as ShapeStyleModel).strokeColor = '#6BA5D7';
        ((connector as ConnectorModel).targetDecorator as
DecoratorModel).shape = 'None';
        connector.type = 'Orthogonal';
        return connector;
    }
    public setNodeTemplate(obj: any): void {
        obj.style.borderColor = (obj.data as {
            color: 'string'
        }).color;
    }
    ngOnInit(): void {
        this.snapSettings = {
            constraints: 0
        }
        this.items = new DataManager(this.data as JSON[], new
Query().take(7));
        //Uses layout to auto-arrange nodes on the Diagram page
        this.layout = {
            type: 'HierarchicalTree',
            //bounds: new Rect(0, 0, 500, 500),
            horizontalSpacing: 25,
            verticalSpacing: 30,
            horizontalAlignment: 'Left',
            verticalAlignment: 'Top'
        }
        //Configures data source for Diagram
        this.dataSourceSettings = {
            id: 'Name',
            parentId: 'ReportingPerson',
            dataSource: this.items
        }
    }
}

```

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 Diagram component

Diagram provides built-in compliance with the [WAI-ARIA](#) specifications. WAI-ARIA Accessibility supports are achieved through the attributes like [aria-label](#). It helps to provides information about elements in a document for assistive technology.

The accessibility compliance for the diagram component is outlined below.

Accessibility Criteria	Compatibility
-----	-----

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

Attributes	Purpose
aria-label	Provides an accessible name for the Diagram Objects.

Aria-label

Attribute provides the text label with some default description for below elements in diagram.

<!-- markdownlint-disable MD033 -->

Element	Default description
ResizeNorthWest	Thumb to resize the selected object on the top-left corner.
ResizeNorthEast	Thumb to resize the selected object on the top-right side direction.
ResizeSouthWest	Thumb to resize the selected object on the bottom-left side direction.
ResizeSouthEast	Thumb to resize the selected object on the bottom-right side direction.
ResizeNorth	Thumb to resize the selected object on the top side direction.
ResizeSouth	Thumb to resize the selected object on the bottom side direction.
ResizeWest	Thumb to resize the selected object on the left side direction.
ResizeEast	Thumb to resize the selected object on the right side direction.
ConnectorSourceThumb	Thumb to move the source point of the connector.
ConnectorTargetThumb	Thumb to move the target point of the connector.
RotateThumb	Thumb to rotate the selected object.

Mobile device support

Syncfusion Diagram component are more user-friendly and accessible to individuals using mobile devices, including those with disabilities. These are designed to be responsive, adaptable to various screen sizes and orientations, and touch-friendly.

Screen Reader Support

The Diagram component supports and its information was dictated properly by the screen readers based on the ARIA attributes and content.

Keyboard navigation support

Syncfusion Diagram component support keyboard navigation, allowing users who rely on alternate methods to effortlessly navigate and interact with the component.

Keyboard interaction

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

| **Command** | **Action** |

| --- | --- |

| Ctrl + A | Select All |

| Ctrl + X | Cut |

| Ctrl + C | Copy |

| Ctrl + V | Paste |

| Ctrl + Z | Undo |

| Ctrl + Y | Redo |

| Delete | Delete |

| Up Arrow | Move selected object to up |

| Down Arrow | Move selected object to down |

| Left Arrow | Move selected object to left |

| Right Arrow | Move selected object to right |

| Enter | Start Annotation Edit |

| Escape | End Annotation Edit |

Ensuring accessibility

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

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

See also

- [Accessibility in Syncfusion Angular components](#)

Commands in Angular Diagram component

<!-- markdownlint-disable MD010 -->

There are several commands available in the diagram as follows.

- Alignment commands
- Spacing commands
- Sizing commands
- Clipboard commands
- Grouping commands
- Z-order commands
- Zoom commands
- Nudge commands
- FitToPage commands
- Undo/Redo commands

Align

Alignment commands enable you to align the selected or defined objects such as nodes and connectors with respect to the selection boundary. Refer to [align](#) commands which shows how to use align methods in the diagram.

<!-- markdownlint-disable MD033 -->

Parameters	Description
:----- :-----:	

`[[`AlignmentOptions`](https://ej2.syncfusion.com/angular/documentation/api/diagram/alignmentOptions#AlignmentOptions) |`

Defines the specific direction, with respect to which the objects to be aligned. The accepted values of the argument "alignment options" are as follows.

Left Aligns all the selected objects at the left of the selection boundary.

Right Aligns all the selected objects at the right of the selection boundary.

Center Aligns all the selected objects at the center of the selection boundary.

Top Aligns all the selected objects at the top of the selection boundary.

Bottom Aligns all the selected objects at the bottom of the selection boundary.

Middle Aligns all the selected objects at the middle of the selection boundary.

|

| **Objects** | `<p align="left">Defines the objects to be aligned. This is an optional parameter. By default, all the nodes and connectors in the selected region of the diagram gets aligned.</p>` |

`[`AlignmentMode`](https://ej2.syncfusion.com/angular/documentation/api/diagram/alignmentMode#AlignmentMode) |`

Defines the specific mode, with respect to which the objects to be aligned. This is an optional parameter. The default alignment mode is ``Object``.

The accepted values of the argument "alignment mode" are as follows.

Object Aligns the objects based on the first object in the selected list.

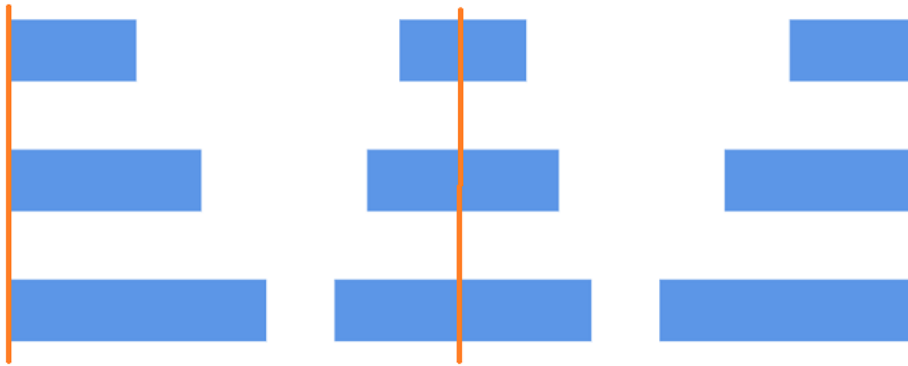
Selector Aligns the objects based on the selection boundary.

|

The following code example illustrates how to align all the selected objects at the left side of the selection boundary.

```
`typescript
@Component({
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%" height="580px" [getNodeDefaults]
    ='getNodeDefaults' (created)='created($event)'>
    <e-nodes>
    <e-node id='node1' [offsetX]=100 [offsetY]=100 [width]=90>
    </e-node>
```

```
<e-node id='node2' [offsetX]=100 [offsetY]=170 [width]=100>
</e-node>
<e-node id='node3' [offsetX]=100 [offsetY]=240 [width]=140>
</e-node>
</e-nodes>
</ejs-diagram>`,
encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram: DiagramComponent;
  public selArray: (NodeModel | ConnectorModel)[] = [];
  public getNodeDefaults(node: NodeModel): NodeModel {
    node.height = 60;
    node.style.fill = "#6BA5D7";
    node.style.strokeColor = "White";
    return node;
  }
  public created(args: Object): void {
    this.selArray = [];
    this.selArray.push(this.diagram.nodes[0], this.diagram.nodes[1], this.diagram.nodes[2]);
    //Selects the nodes
    this.diagram.select(this.selArray);
    //Sets direction as left
    this.diagram.align('Left', this.diagram.selectedItems.nodes, 'Selector');
    this.diagram.dataBind();
  }
}
```



Distribute

The [Distribute](#) commands enable to place the selected objects on the page at equal intervals from each other. The selected objects are equally spaced within the selection boundary.

The factor to distribute the shapes [DistributeOptions](#) are listed as follows:

- RightToLeft: Distributes the objects based on the distance between the right and left sides of the adjacent objects.
- Left: Distributes the objects based on the distance between the left sides of the adjacent objects.
- Right: Distributes the objects based on the distance between the right sides of the adjacent objects.
- Center: Distributes the objects based on the distance between the center of the adjacent objects.
- BottomToTop: Distributes the objects based on the distance between the bottom and top sides of the adjacent objects.
- Top: Distributes the objects based on the distance between the top sides of the adjacent objects.
- Bottom: Distributes the objects based on the distance between the bottom sides of the adjacent objects.
- Middle: Distributes the objects based on the distance between the vertical center of the adjacent objects.

The following code example illustrates how to execute the space commands.

```
`typescript
@Component({
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%" height="580px" [getNodeDefaults]
    ='getNodeDefaults' (created)='created($event)'>
    <e-nodes>
    <e-node id='node1' [offsetX]=100 [offsetY]=100>
    </e-node>
    <e-node id='node2' [offsetX]=100 [offsetY]=170>
```

```
</e-node>
<e-node id='node3' [offsetX]=100 [offsetY]=240>
</e-node>
</e-nodes>
</ejs-diagram>`,
encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram: DiagramComponent;
  public selArray: (NodeModel | ConnectorModel)[] = [];
  public getNodeDefaults(node: NodeModel): NodeModel {
    node.width = 90;
    node.height = 60;
    node.style.fill = "#6BA5D7";
    node.style.strokeColor = "White";
    return node;
  }
  public created(args: Object): void {
    this.selArray = [];
    this.selArray.push(this.diagram.nodes[0], this.diagram.nodes[1], this.diagram.nodes[2]);
    //Selects the nodes
    this.diagram.select(this.selArray);
    //Distributes space between the nodes
    this.diagram.distribute('RightToLeft', this.diagram.selectedItems.nodes);
  }
}
```




Sizing

Sizing [sameSize](#) commands enable to equally size the selected nodes with respect to the first selected object.

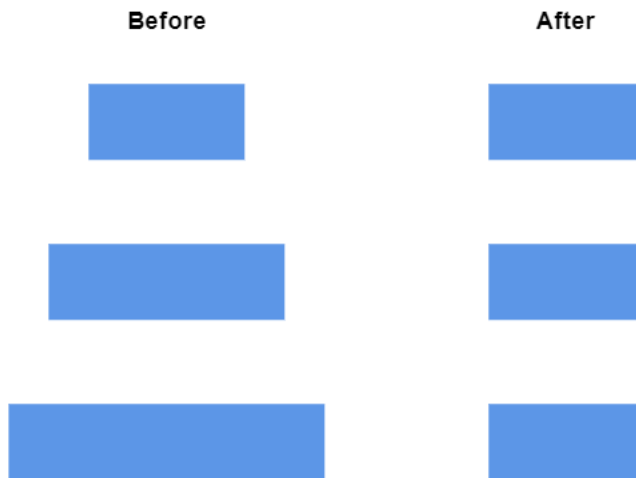
[SizingOptions](#) are as follows:

- Width: Scales the width of the selected objects.
- Height: Scales the height of the selected objects.
- Size: Scales the selected objects both vertically and horizontally.

The following code example illustrates how to execute the size commands.

```
`typescript
@Component({
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%" height="580px" [getNodeDefaults]
    ='getNodeDefaults' (created)='created($event)'>
    <e-nodes>
    <e-node id='node1' [offsetX]=100 [offsetY]=100 [width]=90>
    </e-node>
    <e-node id='node2' [offsetX]=100 [offsetY]=170 [width]=100>
    </e-node>
    <e-node id='node3' [offsetX]=100 [offsetY]=240 [width]=140>
    </e-node>
```

```
</e-nodes>
</ejs-diagram>`,
encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram: DiagramComponent;
  public selArray: (NodeModel | ConnectorModel)[] = [];
  public getNodeDefaults(node: NodeModel): NodeModel {
    node.width = 90;
    node.height = 60;
    node.style.fill = "#6BA5D7";
    node.style.strokeColor = "White";
    return node;
  }
  public created(args: Object): void {
    this.selArray = [];
    this.selArray.push(this.diagram.nodes[0], this.diagram.nodes[1], this.diagram.nodes[2]);
    //Selects the nodes
    this.diagram.select(this.selArray);
    //Resizes the selected nodes with the same width
    this.diagram.sameSize('Width', this.diagram.selectedItems.nodes);
  }
}
`
```



Clipboard

Clipboard commands are used to cut, copy, or paste the selected elements. Refer to the following link which shows how to use clipboard methods in the diagram.

- Cuts the selected elements from the diagram to the diagram's clipboard, [cut](#).
- Copies the selected elements from the diagram to the diagram's clipboard, [copy](#).
- Pastes the diagram's clipboard data (nodes/connectors) into the diagram, [paste](#).

The following code illustrates how to execute the clipboard commands.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import { DiagramComponent, Diagram, NodeModel, ConnectorModel, PointModel,
ShapeStyleModel } from '@syncfusion/ej2-angular-diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%"
height="580px" [getNodeDefaults] ='getNodeDefaults'
[getConnectorDefaults]="getConnectorDefaults" (created)='created($event)'>
  <e-nodes>
    <e-node id='node1' [offsetX]=100 [offsetY]=100>
    </e-node>
    <e-node id='node2' [offsetX]=240 [offsetY]=100>
    </e-node>
  </e-nodes>
  <e-connectors>
    <e-connector id='connector' type='Orthogonal'
[sourcePoint]='sourcePoint1' [targetPoint]='targetPoint1'>
```

```

        </e-connector>
    </e-connectors>
</ejs-diagram>`,
    encapsulation: ViewEncapsulation.None
})
export class AppComponent {
    @ViewChild("diagram")
    public diagram?: DiagramComponent;
    public sourcePoint1?: PointModel;
    public targetPoint1?: PointModel;
    ngOnInit(): void {
        this.sourcePoint1 = { x: 300, y: 100 };
        this.targetPoint1 = { x: 400, y: 300 };
    }
    public getNodeDefaults(node: NodeModel | any): NodeModel {
        node.height = 100;
        node.width = 100;
        ((node as NodeModel).style as ShapeStyleModel).fill = "#6BA5D7";
        ((node as NodeModel).style as ShapeStyleModel).strokeColor =
"White";
        return node;
    }
    public getConnectorDefaults(obj: ConnectorModel): void {
        obj.style = {
            strokeColor: '#6BA5D7',
            fill: '#6BA5D7',
            strokeWidth: 2
        }
        obj.targetDecorator = {
            style: {
                fill: '#6BA5D7',
                strokeColor: '#6BA5D7'
            }
        }
    }
    public created(args: Object): void {
        ((this.diagram as DiagramComponent).select([(this.diagram as
DiagramComponent).nodes[0], (this.diagram as DiagramComponent).nodes[1],
(this.diagram as DiagramComponent).connectors[0]]);
        //copies the selected nodes
        ((this.diagram as DiagramComponent).copy();
        //pastes the copied objects
        ((this.diagram as DiagramComponent).paste((this.diagram as
DiagramComponent).copy() as (NodeModel | ConnectorModel) []));
    }
}

```

MAIN.TS

```

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

```

Grouping

Grouping commands are used to group/ungroup the selected elements on the diagram. Refer to the following link which shows how to use grouping commands in the diagram.

[Group](#) the selected nodes and connectors in the diagram.

[Ungroup](#) the selected nodes and connectors in the diagram.

The following code illustrates how to execute the grouping commands.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import { DiagramComponent, Diagram, NodeModel, ConnectorModel,
ShapeStyleModel } from '@syncfusion/ej2-angular-diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%"
height="580px" [getNodeDefaults] ='getNodeDefaults'
[getConnectorDefaults]='getConnectorDefaults' (created)='created($event)'>
    <e-nodes>
      <e-node id='node1' [offsetX]=100 [offsetY]=100>
      </e-node>
      <e-node id='node2' [offsetX]=240 [offsetY]=100>
      </e-node>
      <e-node id='node3' [offsetX]=240 [offsetY]=100>
      </e-node>
      <e-node id='group1' [offsetX]=240 [offsetY]=100
[children]="children">
      </e-node>
    </e-nodes>
    <e-connectors>
      <e-connector id='connector' type='Orthogonal'
[sourceID]='sourcePoint1' [targetID]='targetPoint1'>
      </e-connector>
    </e-connectors>
  </ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram?: DiagramComponent;
  public children?: string[];
  sourcePoint1: any;
  targetPoint1: any;
  ngOnInit(): void {
    this.children = ['node1', 'node2', 'connector'];
  }
  public getNodeDefaults(node: NodeModel | any): NodeModel {
    node.height = 100;
```

```

        node.width = 100;
        ((node as NodeModel).style as ShapeStyleModel).fill = "#6BA5D7";
        ((node as NodeModel).style as ShapeStyleModel).strokeColor =
"White";
        return node;
    }
    public getConnectorDefaults(obj: ConnectorModel): void {
        obj.style = {
            strokeColor: '#6BA5D7',
            fill: '#6BA5D7',
            strokeWidth: 2
        }
        obj.targetDecorator = {
            style: {
                fill: '#6BA5D7',
                strokeColor: '#6BA5D7'
            }
        }
    }
    public created(args: Object): void {
        //Selects the diagram
        (this.diagram as DiagramComponent).selectAll();
        //Groups the selected elements.
        (this.diagram as DiagramComponent).group();
    }
}

```

MAIN.TS

```

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

```

Z-Order command

Z-Order commands enable you to visually arrange the selected objects such as nodes and connectors on the page.

bringToFront command

The [bringToFront](#) command visually brings the selected element to front over all the other overlapped elements. The following code illustrates how to execute the `bringToFront` command.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import { DiagramComponent, Diagram, NodeModel, ConnectorModel,
ShapeStyleModel } from '@syncfusion/ej2-angular-diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [ ],

```

```

standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%"
height="580px" [getNodeDefaults] = 'getNodeDefaults'
(created)='created($event)'>
    <e-nodes>
      <e-node id='node1' [offsetX]=100 [offsetY]=100>
      </e-node>
      <e-node id='node2' [offsetX]=240 [offsetY]=100>
      </e-node>
      <e-node id='node3' [offsetX]=240 [offsetY]=100>
      </e-node>
    </e-nodes>
  </ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram?: DiagramComponent;
  public selArray: (NodeModel | ConnectorModel)[] = [];
  public getNodeDefaults(node: NodeModel | any): NodeModel {
    node.height = 100;
    node.width = 100;
    ((node as NodeModel).style as ShapeStyleModel).fill = "#6BA5D7";
    ((node as NodeModel).style as ShapeStyleModel).strokeColor =
    "White";
    return node;
  }
  public created(args: Object): void {
    this.selArray = [];
    //((this.diagram as Diagram).appendTo('#element'));
    this.selArray.push((this.diagram as DiagramComponent).nodes[2]);
    //Selects the nodes
    (this.diagram as DiagramComponent).select(this.selArray);
    //Brings to front
    (this.diagram as DiagramComponent).bringToFront();
  }
}

```

MAIN.TS

```

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

```

sendToBack command

The [sendToBack](#) command visually moves the selected element behind all the other overlapped elements. The following code illustrates how to execute the `sendToBack` command.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'

```

```

import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import { DiagramComponent, Diagram, NodeModel, ConnectorModel,
ShapeStyleModel } from '@syncfusion/ej2-angular-diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%"
height="580px" [getNodeDefaults] ='getNodeDefaults'
(created)='created($event)'>
    <e-nodes>
      <e-node id='node1' [offsetX]=100 [offsetY]=100>
      </e-node>
      <e-node id='node2' [offsetX]=240 [offsetY]=100>
      </e-node>
      <e-node id='node3' [offsetX]=240 [offsetY]=100>
      </e-node>
    </e-nodes>
  </ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram?: DiagramComponent;
  public selArray: (NodeModel | ConnectorModel)[] = [];
  public getNodeDefaults(node: NodeModel | any): NodeModel {
    node.height = 100;
    node.width = 100;
    ((node as NodeModel).style as ShapeStyleModel).fill = "#6BA5D7";
    ((node as NodeModel).style as ShapeStyleModel).strokeColor =
    "White";
    return node;
  }
  public created(args: Object): void {
    this.selArray = [];
    //((this.diagram as Diagram).appendTo('#element'));
    this.selArray.push((this.diagram as DiagramComponent).nodes[2]);
    //Selects the nodes
    (this.diagram as DiagramComponent).select(this.selArray);
    //Sends to back
    (this.diagram as DiagramComponent).sendToBack();
  }
}

```

MAIN.TS

```

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

```


moveForward command

The [moveForward](#) command visually moves the selected element over the nearest overlapping element. The following code illustrates how to execute the `moveForward` command.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import { DiagramComponent, Diagram, NodeModel, ConnectorModel,
ShapeStyleModel } from '@syncfusion/ej2-angular-diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%"
height="580px" [getNodeDefaults] ='getNodeDefaults'
(created)='created($event)'>
    <e-nodes>
      <e-node id='node1' [offsetX]=100 [offsetY]=100>
      </e-node>
      <e-node id='node2' [offsetX]=240 [offsetY]=100>
      </e-node>
      <e-node id='node3' [offsetX]=240 [offsetY]=100>
      </e-node>
    </e-nodes>
  </ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram?: DiagramComponent;
  public selArray: (NodeModel | ConnectorModel)[] = [];
  public getNodeDefaults(node: NodeModel | any): NodeModel {
    node.height = 100;
    node.width = 100;
    ((node as NodeModel).style as ShapeStyleModel).fill = "#6BA5D7";
    ((node as NodeModel).style as ShapeStyleModel).strokeColor =
    "White";
    return node;
  }
  public created(args: Object): void {
    this.selArray = [];
    //(this.diagram as Diagram).appendTo('#element');
    this.selArray.push((this.diagram as DiagramComponent).nodes[2]);
    //Selects the nodes
    (this.diagram as DiagramComponent).select(this.selArray);
    //Moves forward
    (this.diagram as DiagramComponent).moveForward();
  }
}
```

MAIN.TS

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

sendBackward command

The [sendBackward](#) command visually moves the selected element behind the underlying element. The following code illustrates how to execute the [sendBackward](#) command.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import { DiagramComponent, Diagram, NodeModel, ConnectorModel,
ShapeStyleModel } from '@syncfusion/ej2-angular-diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%"
height="580px" [getNodeDefaults] ='getNodeDefaults'
(created)='created($event)'>
    <e-nodes>
      <e-node id='node1' [offsetX]=100 [offsetY]=100>
      </e-node>
      <e-node id='node2' [offsetX]=240 [offsetY]=100>
      </e-node>
      <e-node id='node3' [offsetX]=240 [offsetY]=100>
      </e-node>
    </e-nodes>
  </ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram?: DiagramComponent;
  public selArray: (NodeModel | ConnectorModel)[] = [];
  public getNodeDefaults(node: NodeModel | any): NodeModel {
    node.height = 100;
    node.width = 100;
    ((node as NodeModel).style as ShapeStyleModel).fill = "#6BA5D7";
    ((node as NodeModel).style as ShapeStyleModel).strokeColor =
    "White";
    return node;
  }
  public created(args: Object): void {
    this.selArray = [];
    //(this.diagram as Diagram).appendTo('#element');
    this.selArray.push((this.diagram as DiagramComponent).nodes[2]);
```

```
//Selects the nodes
(this.diagram as DiagramComponent).select(this.selArray);
//Sends backward
(this.diagram as DiagramComponent).sendBackward();
}
}
```

MAIN.TS

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

Zoom

The [zoom](#) command is used to zoom-in and zoom-out the diagram view.

The following code illustrates how to zoom-in/zoom out the diagram.

```
`typescript
@Component({
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%" height="580px"
(created)='created($event)'>
</ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram: DiagramComponent;
  public created(args: Object): void {
    // Sets the zoomFactor
    //Defines the focusPoint to zoom the Diagram with respect to any point
    //When you do not set focus point, zooming is performed with reference to the center of current
    Diagram view.
    this.diagram.zoom(1.2, {
      x: 100,
      y: 100
    });
  }
}
```

,

Nudge command

The [nudge](#) commands move the selected elements towards up, down, left, or right by 1 pixel.

[NudgeDirection](#) nudge command moves the selected elements towards the specified direction by 1 pixel, by default.

The accepted values of the argument "direction" are as follows:

- Up: Moves the selected elements towards up by the specified delta value.
- Down: Moves the selected elements towards down by the specified delta value.
- Left: Moves the selected elements towards left by the specified delta value.
- Right: Moves the selected elements towards right by the specified delta value.

The following code illustrates how to execute nudge command.

```
`typescript
@Component({
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%" height="580px"
(created)='created($event)'>
</ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram: DiagramComponent;
  public created(args: Object): void {
    //Nudges to right
    this.diagram.nudge('Right');
  }
}
```

,

Nudge by using arrow keys

The corresponding arrow keys are used to move the selected elements towards up, down, left, or right direction by 1 pixel.



Nudge commands are particularly useful for accurate placement of elements.

BringIntoView

The [bringIntoView](#) command brings the specified rectangular region into the viewport of the diagram.

The following code illustrates how to execute the `bringIntoView` command.

```
`typescript
@Component({
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%" height="580px"
(created)='created($event)'>
</ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram: DiagramComponent;
  public bound: Rect
  public created(args: Object): void {
    //Brings the specified rectangular region of the Diagram content to the viewport of the page.
    this.bound= new Rect(200, 400, 500, 400);
    this.diagram.bringIntoView(this.bound);
  }
}
```

BringToCenter

The [bringToCenter](#) command brings the specified rectangular region of the diagram content to the center of the viewport.

The following code illustrates how to execute the `bringToCenter` command.

```
`typescript
@Component({
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%" height="580px"
(created)='created($event)'>
</ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
```

```
export class AppComponent {
  @ViewChild("diagram")
  public diagram: DiagramComponent;
  public bound: Rect
  public created(args: Object): void {
    //Brings the specified rectangular region of the Diagram content to the center of the viewport.
    this.bound= new Rect(200, 400, 500, 400);
    this.diagram.bringToCenter(this.bound);
  }
}
```

FitToPage command

The [fitToPage](#) command helps to fit the diagram content into the view with respect to either width, height, or at the whole.

The [mode](#) parameter defines whether the diagram has to be horizontally/vertically fit into the viewport with respect to width, height, or entire bounds of the diagram.

The [region](#) parameter defines the region that has to be drawn as an image.

The [margin](#) parameter defines the region/bounds of the diagram content that is to be fit into the view.

The [canZoomIn](#) parameter enables/disables zooming to fit the smaller content into a larger viewport.

The [customBounds](#) parameter the custom region that has to be fit into the viewport.

The following code illustrates how to execute [FitToPage](#) command.

```
`typescript
@Component({
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%" height="580px"
    (created)='created($event)'>
</ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram: DiagramComponent;
  public bound: Rect;
  public created(args: Object): void {
```

//fit the diagram to the page with respect to mode and region

```
this.diagram.fitToPage({  
  mode: 'Page',  
  region: 'Content',  
  margin: {  
    bottom: 50  
  },  
  canZoomIn: false  
});  
}  
}
```

Command manager

Diagram provides support to map/bind command execution with desired combination of key gestures. Diagram provides some built-in commands. [CommandManager](#) provides support to define custom commands. The custom commands are executed, when the specified key gesture is recognized.

Custom command

To define a custom command, specify the following properties:

- [execute](#): A method to be executed.
- [canExecute](#): A method to define whether the command can be executed at the moment.
- [gesture](#): A combination of [keys](#) and [KeyModifiers](#).
- [parameter](#): Defines any additional parameters that are required at runtime.
- [name](#): Defines the name of the command.

To explore the properties of custom commands, refer to [Commands](#).

The following code example illustrates how to define a custom command.

```
`typescript  
@Component({  
  selector: "app-container",  
  template: `<ejs-diagram #diagram id="diagram" width="100%" height="580px"  
[commandManager]="commandManager">  
</ejs-diagram>`,  
  encapsulation: ViewEncapsulation.None  
})  
export class AppComponent {  
  @ViewChild("diagram")
```

```
public diagram: DiagramComponent;
public commandManager: CommandManager;
ngOnInit(): void {
  this.commandManager = {
    commands: [{
      name: 'customCopy',
      parameter: 'node',
      //Method to define whether the command can be executed at the current moment
      canExecute: function() {
        //Defines that the clone command can be executed, if and only if the selection list is not empty.
        if (diagram.selectedItems.nodes.length > 0 || diagram.selectedItems.connectors.length > 0) {
          return true;
        }
        return false;
      },
      //Command handler
      execute: function() {
        //Logic to clone the selected element
        diagram.copy();
        diagram.paste();
        diagram.dataBind();
      },
      //Defines that the clone command has to be executed on the recognition of key press.
      gesture: {
        key: Keys.G,
        keyModifiers: KeyModifiers.Shift | KeyModifiers.Alt
      }
    ]
  },
}
```


Modify the existing command

When any one of the default commands is not desired, they can be disabled. To change the functionality of a specific command, the command can be completely modified.

The following code example illustrates how to disable a command and how to modify the built-in commands.

```
`typescript
@Component({
  selector: "app-container",
  template: `
```

```
,
{
  name: 'nudgeLeft',
  canExecute: (): boolean => {
    if (this.diagram.selectedItems.nodes.length > 0) {
      return true;
    }
    return false;
  },
  execute: (): void => {
    this.diagram.nudge("Left");
  },
  gesture: {
    key: Keys.Left
  }
}
],
},
}
},
`
```

See Also

- [How to create the custom context menu items](#)

Undo redo in Angular Diagram component

Diagram tracks the history of actions that are performed after initializing the diagram and provides support to reverse and restore those changes.

Undo and redo

Diagram provides built-in support to track the changes that are made through interaction and through public APIs. The changes can be reverted or restored either through shortcut keys or through commands.

Note: If you want to use Undo-Redo in diagram, you need to inject UndoRedo in the diagram.

Undo/redo through shortcut keys

Undo/redo commands can be executed through shortcut keys. Shortcut key for undo is Ctrl+z and shortcut key for redo is Ctrl+y.

Undo/redo through public APIs

The client-side methods [undo](#) and [redo](#) help you to revert/restore the changes. The following code example illustrates how to undo/redo the changes through script.

```
`typescript
@Component({
  selector: "app-container",
  // Diagram template
  template: `<ejs-diagram #diagram id="diagram" width="100%" height="580px"
(created)='created($event)'>
</ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram: DiagramComponent;
  public created(args: Object): void {
    // Reverts the last action performed
    this.diagram.undo();
    // Restores the last undone action
    this.diagram.redo();
  }
}
```

When a change in the diagram is reverted or restored (undo/redo), the `historyChange` event gets triggered.

Group multiple changes

History list allows to revert or restore multiple changes through a single undo/redo command. For example, revert/restore the fill color change of multiple elements at a time.

The client-side method [startGroupAction](#) is used to notify the diagram to start grouping the changes. The client-side method [endGroupAction](#) is used to notify to stop grouping the changes. The following code illustrates how to undo/redo fill color change of multiple elements at a time.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule, UndoRedoService } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
```

```

import { DiagramComponent, Diagram, NodeModel, ShapeStyleModel } from
"@syncfusion/ej2-angular-diagrams";
@Component({
  imports: [
    DiagramModule
  ],
  providers: [UndoRedoService],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%"
height="580px" [getNodeDefaults]='getNodeDefaults'
(created)='created($event)'>
    <e-nodes>
      <e-node id='node1' [offsetX]=150 [offsetY]=150>
      </e-node>
    </e-nodes>
  </ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram?: DiagramComponent;
  public getNodeDefaults(node: NodeModel): NodeModel {
    node.height = 100;
    node.width = 100;
    (node as NodeModel).style as ShapeStyleModel).fill = '#6BA5D7';
    (node as NodeModel).style as ShapeStyleModel).strokeColor
= 'white';
    return node;
  };
  public created(args: Object): void {
    //Start to group the changes
    (this.diagram as Diagram).startGroupAction();
    //Makes the changes
    let color: string[] = ['black', 'red', 'green', 'yellow'];
    for (var i = 0; i < color.length; i++) {
      // Updates the fillColor for all the child elements.
      ((this.diagram as Diagram).nodes[0].style as
ShapeStyleModel).fill = color[i];
      (this.diagram as Diagram).dataBind();
    }
    //Ends grouping the changes
    (this.diagram as Diagram).endGroupAction();
  }
}

```

MAIN.TS

```

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

```

Track custom changes

Diagram provides options to track the changes that are made to custom properties. For example, in case of an employee relationship diagram, track the changes in the employee information. The historyList of the diagram enables you to track such changes.

Before changing the employee information, save the existing information to historyList by using the client-side method push of historyList.

The historyList canLog method can be used which takes a history entry as argument and returns whether the specific entry can be added or not.

```
`typescript
@Component({
  selector: "app-container",
  // render initialized Diagram
  template: `<ejs-diagram #diagram id="diagram" width="100%" height="580px"
(created)='create($event)'>
</ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram: DiagramComponent;
  public created(args: Object): void {
    //Creates a custom entry
    let entry: HistoryEntry = {
      undoObject: this.diagram.nodes[0];
    };
    // adds the history to history list
    this.diagram.historyList.push(entry);
    this.diagram.dataBind();
  }
}
```

canLog

canLog in the history list, which takes a history entry as argument and returns whether the specific entry can be added or not.

APP.COMPONENT.TS

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

```

import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule, UndoRedoService } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, ViewChild } from "@angular/core";
import { DiagramComponent, Diagram, NodeModel, HistoryEntry, ShapeStyleModel } from "@syncfusion/ej2-angular-diagrams";
@Component({
  imports: [
    DiagramModule
  ],
  providers: [UndoRedoService],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%" height="580px" [getNodeDefaults]='getNodeDefaults' (created)='created($event)'>
    <e-nodes>
      <e-node id='node1' [offsetX]=150 [offsetY]=150>
      </e-node>
    </e-nodes>
  </ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram?: DiagramComponent;
  public getNodeDefaults(node: NodeModel): NodeModel {
    node.height = 100;
    node.width = 100;
    ((node as NodeModel).style as ShapeStyleModel).fill = '#6BA5D7';
    ((node as NodeModel).style as ShapeStyleModel).strokeColor =
    'white';
    return node;
  };
  public created(args: Object): void {
    // canLog decide whether the entry add or not in history List
    (this.diagram as Diagram | any).historyList.canLog = function(entry:
    HistoryEntry) {
      entry.cancel = true;
      return entry;
    }
  }
}

```

MAIN.TS

```

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

```

Track undo/redo actions

The historyList undoStack property is used to get the collection of undo actions which should be performed in the diagram. The undoStack/redoStack is the read-only property.

```
`typescript
@Component({
  selector: "app-container",
  // render initialized Diagram
  template: `<ejs-diagram #diagram id="diagram" width="100%" height="580px">
</ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram: DiagramComponent;
  //get the collection of undoStack objects
  public undoStack: HistoryEntry[] = this.diagram.historyList.undoStack;
  //get the collection of redoStack objects
  public redoStack: HistoryEntry[] = this.diagram.historyList.redoStack;
}
`
```

History change event

The [historyChange](#) event triggers, whenever the interaction of the node and connector is take place.

```
`typescript
@Component({
  selector: "app-container",
  // render initialized Diagram
  template: `<ejs-diagram #diagram id="diagram" width="100%" height="580px"
(created)='create($event)'>
</ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram: DiagramComponent;
  public created(args: Object): void {
    // history change event
    this.diagram.historyChange = (arg: IHistoryChangeArgs) => {
```

```
//causes of history change
let cause: string = arg.cause;
}
}
}
`
`
```

Stack limit

The [stackLimit](#) property of history manager is used to limits the number of actions to be stored on the history manager.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule, UndoRedoService } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, ViewChild } from "@angular/core";
import { DiagramComponent, Diagram, NodeModel, HistoryEntry, ShapeStyleModel } from "@syncfusion/ej2-angular-diagrams";
@Component({
  imports: [
    DiagramModule
  ],
  providers: [UndoRedoService],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%" height="580px" [getNodeDefaults]='getNodeDefaults' (created)='created($event)'>
    <e-nodes>
      <e-node id='node1' [offsetX]=150 [offsetY]=150>
      </e-node>
    </e-nodes>
  </ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram?: DiagramComponent;
  public getNodeDefaults(node: NodeModel): NodeModel {
    node.height = 100;
    node.width = 100;
    ((node as NodeModel).style as ShapeStyleModel).fill = '#6BA5D7';
    ((node as NodeModel).style as ShapeStyleModel).strokeColor = 'white';
    return node;
  };
  public created(args: Object): void {
    // canLog decide whether the entry add or not in history List
    (this.diagram as Diagram).historyManager.stackLimit = 3;
  }
}
```


MAIN.TS

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

[Retain selection](#)

You can retain a selection at undo/redo operation by using the client-side API Method called **updateSelection**. Using this method, we can select a diagram objects.

`typescript

```
@Component({
  selector: "app-container",
  // render initialized Diagram
  template: `<ejs-diagram #diagram id="diagram" width="100%" height="580px"
(created)=create($event)'>
</ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram: DiagramComponent;
  public created(args: Object): void {
    // history change event
    this.diagram.updateSelection: (object: NodeModel, diagram: Diagram) => {
      let selArr = [];
      selArr.push(object)
      diagram.select(selArr);
    },
  }
}
```

[Virtualization in Angular Diagram component](#)[Virtualization in Diagram](#)

Virtualization is the process of loading the diagramming objects available in the visible area of the Diagram control, that is, only the diagramming objects that lie within the ViewPort of the Scroll Viewer are loaded (remaining objects are loaded only when they come into view).

This feature gives an optimized performance while loading and dragging items to the Diagram that consists of many Nodes and Connectors.

The following code illustrates how to enable Virtualization mode in the diagram.

```
`typescript
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import { DiagramConstraints } from '@syncfusion/ej2-diagrams';
import { DiagramComponent } from '@syncfusion/ej2-angular-diagrams';
@Component({
  selector: "app-container",
  // specifies the template string for the diagram component
  template: <ejs-diagram id="diagram" width="100%" height="580px"
[constraints]='diagramConstraints'></ejs-diagram>
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram: DiagramComponent;
  public diagramConstraints: DiagramConstraints;
  ngOnInit(): void {
    //Enable virtualization in diagram
    this.diagramConstraints = DiagramConstraints.Default | DiagramConstraints.Virtualization;
  }
}
```

Serialization in Angular Diagram component

[Link to the Video](#) is the process of saving and loading for state persistence of the diagram.

To save and load the diagram in Angular, refer to the below video link.

Save

The diagram is serialized as string while saving. The client-side method, [saveDiagram](#) helps to serialize the diagram as a string. The following code illustrates how to save the diagram.

```
`typescript
//returns serialized string of the Diagram
saveData = this.diagram.saveDiagram();
`
```

This string can be converted to JSON data and stored for future use. The following snippet illustrates how to save the serialized string into local storage.

```
`typescript
//Saves the string in to local storage
localStorage.setItem('fileName', saveData);
saveData = localStorage.getItem('fileName');
`
```

Diagram can also be saved as raster or vector image files. For more information about saving the diagram as images, refer to [Print and Export](#).

Load

Diagram is loaded from the serialized string data by client-side method, [loadDiagram](#).

The following code illustrates how to load the diagram from serialized string data.

```
`typescript
//Loads the Diagram from saved json data
this.diagram.loadDiagram(saveData);
`
```

Note: Before loading a new diagram, existing diagram is cleared.

Prevent default values

The diagram provides supports to simplifying the saved JSON object without adding the default properties that are presented in the diagram.

The following code illustrates how to simplify the JSON object.

```
`typescript
let diagram: Diagram = new Diagram({
  serializationSettings: { preventDefaults: true },
});
`
```

Export in Angular Diagram component

Diagram provides support to export its content as image/svg files. The client-side method [exportDiagram](#) helps to export the diagram. The following code illustrates how to export the diagram as image.

Note: To use Print and Export, you need to inject [Link to the Video](#) in the diagram.

To print and export the diagram in Angular, refer to the below video link.

<!-- markdownlint-disable MD033 -->

```
`typescript
public options: IExportOptions;
this.options = {};
this.options.mode = 'Download';
`
```

```
this.diagram.exportDiagram(this.options);
```

```
,
```

Exporting options

Diagram provides support to export the desired region of the diagram to desired formats.

File Name

[FileName](#) is the name of the file to be downloaded. By default, the file name is set to **Diagram**.

Format

[Format](#) is to specify the type/format of the exported file. By default, the diagram is exported as .jpg format. You can export diagram to the following formats:

- JPG
- PNG
- BMP
- SVG

```
`typescript
```

```
public options: IExportOptions;
```

```
this.options = {};
```

```
this.options.mode = 'Download';
```

```
this.options.format = 'SVG';
```

```
this.diagram.exportDiagram(this.options);
```

```
,
```

Margin

[Margin](#) specifies the amount of space that has to be left around the diagram.

<!-- markdownlint-disable MD033 -->

```
`typescript
```

```
public options: IExportOptions;
```

```
this.options = {};
```

```
this.options.mode = 'Download';
```

```
this.options.margin = { left: 10, right: 10, top: 10, bottom: 10};
```

```
this.options.fileName = 'format';
```

```
this.options.format = 'SVG';
```

```
this.diagram.exportDiagram(this.options);
```

```
,
```

Mode

[Mode](#) specifies whether the diagram will be exported as files or get base64 data (ImageURL/SVG). The export options are as follows:

- Download: Exports and downloads the diagram as image/SVG.
- Data: return a base64 string.

The following code example illustrates how to export the diagram as raw data.

```
`typescript
public options: IExportOptions;
this.options = {};
this.options.mode = 'Data';
this.options.margin = { left: 10, right: 10, top: 10, bottom: 10};
this.options.fileName = 'format';
this.options.format = 'SVG';
let base64data = this.diagram.exportDiagram(this.options);
`
```

Region

You can export any particular [region](#) of the diagram and it is categorized into three types as follows.

- PageSettings
- Content
- CustomBounds

PageSettings

Diagram is exported based on the given PageSettings width and height. The Properties available in page settings are as follows.

- width
- height
- margin
- orientation
- boundaryConstraints
- background
- multiplePage
- showPageBreaks
- fitOptions

boundaryConstraints

Defines the editable region of the diagram.

- Infinity - Allow the interactions to take place at infinite height and width.
- Diagram - Allow the interactions to take place around the diagram's height and width.
- Page - Allow the interactions to take place around the page's height and width.

multiplePage

While setting multiple pages as false, the diagram is exported as a single image and while setting multiple pages as true, the diagram is exported as a separate image based on width and height.

The following code example illustrates how to export the region occupied by the diagram elements.

```
`typescript
public options: IExportOptions;

this.options = {};

this.options.mode = 'Download';

this.options.margin = { left: 10, right: 10, top: 10, bottom: 10};

this.options.fileName = 'format';

this.options.format = 'SVG';

this.options.region = 'PageSettings';

this.diagram.exportDiagram(this.options);
`
```

Content

The diagram content alone will be exported as an image.

The following code example illustrates how to export the region occupied by the diagram elements.

```
`javascript
var diagram = new ej.diagrams.Diagram({
width: 1500, height: 1500
}, '#element');

var options = {};

options.mode = 'Download';

options.margin = { left: 10, right: 10, top: 10, bottom: 10};

options.fileName = 'format';

options.format = 'SVG';

options.region = 'Content';

diagram.exportDiagram(options);
`
```

Custom bounds

Diagram provides support to export any specific region of the diagram by using [bounds](#).

The following code example illustrates how to export the region occupied by the diagram elements.

```
`typescript
public options: IExportOptions;
```

```

this.options = {};
this.options.mode = 'Download';
this.options.margin = { left: 10, right: 10, top: 10, bottom: 10};
this.options.fileName = 'region';
this.options.format = 'SVG';
this.options.region = 'CustomBounds';
this.options.bounds.x = 10;
this.options.bounds.y = 10;
this.options.bounds.height = 100;
this.options.bounds.width = 100;
this.diagram.exportDiagram(this.options);
`

```

Export diagram with stretch option

Diagram provides support to export the diagram as image for [stretch](#) option. The exported images will be clearer but larger in file size.

The following code example illustrates how to export the region occupied by the diagram elements.

```

`typescript
public options: IExportOptions;
this.options = {};
this.options.mode = 'Download';
this.options.margin = { left: 10, right: 10, top: 10, bottom: 10};
this.options.fileName = 'region';
this.options.format = 'SVG';
this.options.region = 'Content';
this.options.stretch = 'Stretch';
this.diagram.exportDiagram(this.options);
`

```

Print

The client-side method [print](#) helps to print the diagram as image.

Name	Type	Description
-----	-----	-----
region	enum	Sets the region of the diagram to be printed.
bounds	object	Prints any custom region of diagram.
stretch	enum	Resizes the diagram content to fill its allocated space and printed.

- | multiplePage | boolean | Prints the diagram into multiple pages. |
- | pageWidth | number | Sets the page width of the diagram while printing the diagram into multiple pages. |
- | pageHeight | number | Sets the page height of the diagram while printing the diagram into multiple pages. |
- | pageOrientation | enum | Sets the orientation of the page. |

The following code example illustrates how to export the region occupied by the diagram elements.

```
`typescript
public options: IExportOptions;
this.options = {};
this.options.mode = 'Download';
this.options.region = 'PageSettings';
this.options.multiplePage = true;
this.options.pageHeight = 300;
this.options.pageWidth = 300;
this.diagram.print(this.options);
`
```

Limitations

We have a limitation in exporting the image with HTML and Native node. So, Syncfusion Essential PDF library is used, which supports HTML Content to Image conversion by using the advanced Qt WebKit rendering engine. You can refer to the following KB [link](#) for more details.

Tooltip in Angular Diagram component

```
<!-- markdownlint-disable MD010 -->
```

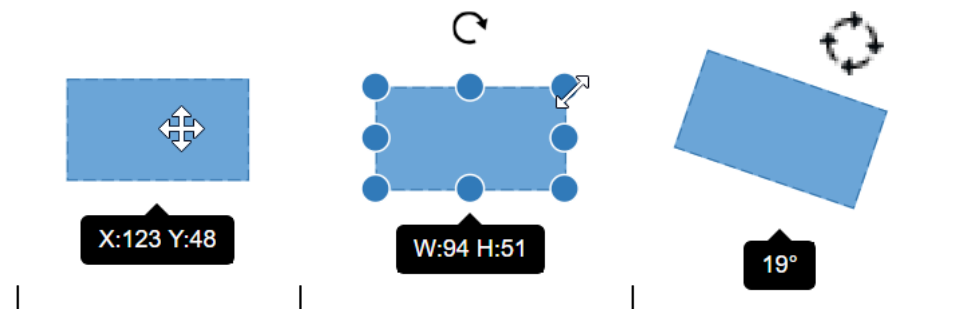
In Graphical User Interface (GUI), the tooltip is a message that is displayed when mouse hovers over an element. The diagram provides tooltip support while dragging, resizing, rotating a node, and when the mouse hovers any diagram element.

Default tooltip

By default, diagram displays a tooltip to provide the size, position, and angle related information while dragging, resizing, and rotating. The following images illustrate how the diagram displays the node information during an interaction.

| Drag | Resize | Rotate |

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



Common tooltip for all nodes and connectors

The diagram provides support to show tooltip when the mouse hovers over any node/connector.

To show tooltip on mouse over, the [tooltip](#) property of diagram model needs to be set with the tooltip [content](#) and [position](#) as shown in the following example.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, OnInit, ViewEncapsulation, ViewChild } from '@angular/core';
import { DiagramComponent, Diagram, NodeModel, DiagramConstraints, DiagramTooltipModel, ShapeStyleModel } from '@syncfusion/ej2-angular-diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%" height="580px" [getNodeDefaults]="getNodeDefaults" [tooltip]="tooltip" [constraints]="constraints">
    <e-nodes>
      <e-node id='node1' [offsetX]=150 [offsetY]=150>
      </e-node>
    </e-nodes>
  </ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram?: DiagramComponent;
  public tooltip?: DiagramTooltipModel;
  public constraints?: DiagramConstraints;
  public getNodeDefaults(node: NodeModel): NodeModel {
    node.height = 100;
    node.width = 100;
    ((node as NodeModel).style as ShapeStyleModel).fill = "#6BA5D7";
    ((node as NodeModel).style as ShapeStyleModel).strokeColor = "White";
    return node;
  }
}
```

```

ngOnInit(): void {
    this.tooltip = {
        content: 'Nodes',
        position: 'TopLeft'
    }
    this.constraints = DiagramConstraints.Default |
DiagramConstraints.Tooltip
    }
}

```

MAIN.TS

```

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

```

Disable tooltip at runtime

The tooltip on mouse over can be disabled by assigning the [tooltip](#) property as `null`. The following code example illustrates how to disable the mouse over tooltip at runtime.

```

`typescript
@Component({
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%" height="580px" [tooltip]="tooltip">
</ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram: DiagramComponent;
  ngOnInit(): void {
    this.tooltip = null
  }
}
`

```

Tooltip for a specific node/connector

The tooltip can be customized for each node and connector. Remove the **InheritTooltip** option from the [constraints](#) of that node/connector. The following code example illustrates how to customize the tooltip for individual elements.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'

```

```

import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, OnInit, ViewEncapsulation, ViewChild } from
 '@angular/core';
import { DiagramComponent, Diagram, NodeModel, NodeConstraints,
DiagramTooltipModel, ShapeStyleModel } from '@syncfusion/ej2-angular-
diagrams';
@Component({
imports: [
    DiagramModule
],
providers: [ ],
standalone: true,
    selector: "app-container",
    template: `<ejs-diagram #diagram id="diagram" width="100%"
height="580px" [getNodeDefaults]="getNodeDefaults">
    <e-nodes>
        <e-node id='node1' [offsetX]=150 [offsetY]=150
[tooltip]="tooltip" [constraints]="constraints">
            </e-node>
        </e-nodes>
    </ejs-diagram>`,
    encapsulation: ViewEncapsulation.None
})
export class AppComponent {
    @ViewChild("diagram")
    public diagram?: DiagramComponent;
    public tooltip?: DiagramTooltipModel;
    public constraints?: NodeConstraints;
    public getNodeDefaults(node: NodeModel): NodeModel {
        node.height = 100;
        node.width = 100;
        ((node as NodeModel).style as ShapeStyleModel).fill = "#6BA5D7";
        ((node as NodeModel).style as ShapeStyleModel).strokeColor =
"White";
        return node;
    }
    ngOnInit(): void {
        this.tooltip = {
            //Sets the content of the Tooltip
            content: 'Node1',
            //Sets the position of the Tooltip
            position: 'BottomRight',
            //Sets the tooltip position relative to the node
            relativeMode: 'Object'
        }
        this.constraints = NodeConstraints.Default | NodeConstraints.Tooltip
    }
}

```

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 Ports

The tooltip feature has been implemented to support Ports, providing the ability to display information or descriptions when the mouse hovers over them.

To display tooltips on mouseover, set the desired tooltip [content](#) by utilizing the [tooltip](#) property.

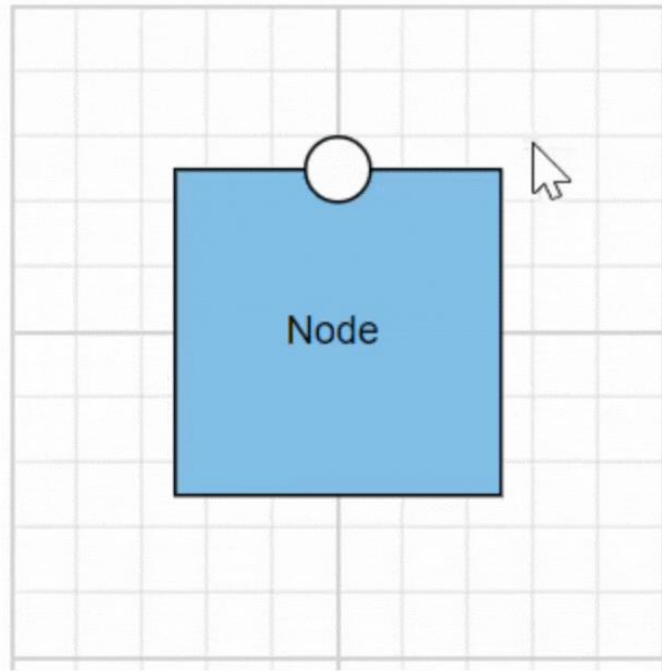
Tooltips for Ports can be enabled or disabled using the [PortConstraints](#) Tooltip property.

```
`ts
let ports: [{
  offset: {x: 1,y: 0.5},
  tooltip: {content: 'Port Tootip'},
  //enable Port Tooltip Constraints
  constraints: PortConstraints.Default | PortConstraints.ToolTip,
  //disable Port Tooltip Constraints
  constraints: PortConstraints.Default ~& PortConstraints.ToolTip
}]
`
```

Dynamic modification of tooltip content is supported, allowing you to change the displayed tooltip content during runtime.

```
`js
{
  //change tooltip content at run time
  diagram.nodes[0].ports[0].tooltip.content = 'New Tooltip Content';
  diagram.databind;
}
`
```

The following image illustrates how the diagram displays tooltips during an interaction with ports:



Here, the code provided below demonstrates the port tooltip Interaction.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, OnInit, ViewEncapsulation, ViewChild } from '@angular/core';
import { DiagramComponent, Diagram, NodeModel, NodeConstraints,
DiagramTooltipModel, PointPortModel, PortVisibility, PortConstraints,
ShapeStyleModel } from '@syncfusion/ej2-angular-diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%"
height="580px" [getNodeDefaults]="getNodeDefaults">
  <e-nodes>
    <e-node id='node1' [offsetX]=150 [offsetY]=150
[tooltip]="tooltip" [constraints]="constraints">
    </e-node>
  </e-nodes>
</ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram?: DiagramComponent;
  public tooltip?: DiagramTooltipModel;
```

```

public constraints?: NodeConstraints;
public port?: PointPortModel;
public getNodeDefaults(node: NodeModel | any): NodeModel {
  node.height = 100;
  node.width = 100;
  node.ports[0]={
    offset: {
      x: 0.5,
      y: 0
    },
    visibility: PortVisibility.Visible,
    //Set the style for the port
    style: {
      fill: '#FFFFFF',
      strokeWidth: 1,
      strokeColor: 'black'
    },
    tooltip:{
      content:'Port Tooltip',
    },
    // Sets the shape of the port as Circle
    shape: 'Circle',
    constraints: PortConstraints.Default
  },
  ((node as NodeModel).style as ShapeStyleModel).fill = "#6BA5D7";
  ((node as NodeModel).style as ShapeStyleModel).strokeColor =
  "White";
  node.p
  return node;
}
ngOnInit(): void {
  this.tooltip = {
    //Sets the content of the Tooltip
    content: 'Node1',
    //Sets the position of the Tooltip
    position: 'BottomRight',
    //Sets the tooltip position relative to the node
    relativeMode: 'Object'
  }
  this.constraints = NodeConstraints.Default | NodeConstraints.Tooltip
}
}

```

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 content

Any text or image can be added to the tooltip, by default. To customize the tooltip layout or to create your own visualized element on the tooltip, template can be used.

The following code example illustrates how to add formatted HTML content to the tooltip.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, OnInit, ViewEncapsulation, ViewChild } from
 '@angular/core';
import { DiagramComponent, Diagram, NodeModel, NodeConstraints,
DiagramTooltipModel, ShapeStyleModel } from '@syncfusion/ej2-angular-
diagrams';
@Component({
imports: [
    DiagramModule
],
providers: [ ],
standalone: true,
    selector: "app-container",
    template: `<ejs-diagram #diagram id="diagram" width="100%"
height="580px" [getNodeDefaults]="getNodeDefaults">
    <e-nodes>
        <e-node id='node1' [offsetX]=150 [offsetY]=150
[tooltip]="tooltip" [constraints]="constraints">
            </e-node>
        </e-nodes>
    </ejs-diagram>`,
    encapsulation: ViewEncapsulation.None
})
export class AppComponent {
    @ViewChild("diagram")
    public diagram?: DiagramComponent;
    public tooltip?: DiagramTooltipModel;
    public constraints?: NodeConstraints;
    public getNodeDefaults(node: NodeModel): NodeModel {
        node.height = 100;
        node.width = 100;
        ((node as NodeModel).style as ShapeStyleModel).fill = "#6BA5D7";
        ((node as NodeModel).style as ShapeStyleModel).strokeColor =
"White";
        return node;
    }
    public getContent(): HTMLElement {
        let tooltipContent: HTMLElement = document.createElement('div');
        tooltipContent.innerHTML = '<div style="background-color: #f4f4f4;
color: black; border-width:1px;border-style: solid;border-color: #d3d3d3;
border-radius: 8px;white-space: nowrap;"> <span style="margin: 10px;">
Tooltip !!! </span> </div>';
        return tooltipContent;
    }
    ngOnInit(): void {
        this.tooltip = {
            //Sets the content of the Tooltip
            content: this.getContent(),
            //Sets the position of the Tooltip
            position: 'TopLeft',
            //Sets the tooltip position relative to the node
            relativeMode: 'Object'
        }
    }
}

```

```

        this.constraints = (NodeConstraints.Default &
~NodeConstraints.InheritTooltip) | NodeConstraints.Tooltip
    }
}

```

MAIN.TS

```

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

```

Tooltip alignments

Tooltip relative to object

The diagram provides support to show tooltip around the node/connector that is hovered by the mouse. The tooltip can be aligned by using the [position](#) property of the tooltip.

The [relativeMode](#) property of the tooltip defines whether the tooltip has to be displayed around the object or at the mouse position.

The following code example illustrates how to position the tooltip around object.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, OnInit, ViewEncapsulation, ViewChild } from
 '@angular/core';
import { DiagramComponent, Diagram, NodeModel, NodeConstraints,
DiagramTooltipModel, ShapeStyleModel } from '@syncfusion/ej2-angular-
diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%"
height="580px" [getNodeDefaults]="getNodeDefaults">
    <e-nodes>
      <e-node id='node1' [offsetX]=150 [offsetY]=150
[tooltip]="tooltip" [constraints]="constraints">
        </e-node>
      </e-nodes>
    </ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram?: DiagramComponent;
  public tooltip?: DiagramTooltipModel;
  public constraints?: NodeConstraints;
  public getNodeDefaults(node: NodeModel): NodeModel {

```



```

        node.height = 100;
        node.width = 100;
        ((node as NodeModel).style as ShapeStyleModel).fill = "#6BA5D7";
        ((node as NodeModel).style as ShapeStyleModel).strokeColor =
"White";
        return node;
    }
    ngOnInit(): void {
        this.tooltip = {
            //Sets the content of the Tooltip
            content: 'Node1',
            //Sets the position of the Tooltip
            position: 'BottomRight',
            //Sets the tooltip position relative to the node
            relativeMode: 'Object'
        }
        this.constraints = (NodeConstraints.Default &
~NodeConstraints.InheritTooltip) | NodeConstraints.Tooltip
    }
}

```

MAIN.TS

```

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

```

Tooltip relative to mouse position

To display the tooltip at mouse position, need to set **mouse** option to the [relativeMode](#) property of the tooltip.

The following code example illustrates how to show tooltip at mouse position.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, OnInit, ViewEncapsulation, ViewChild } from
 '@angular/core';
import { DiagramComponent, Diagram, NodeModel, NodeConstraints,
DiagramTooltipModel, ShapeStyleModel } from '@syncfusion/ej2-angular-
diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%"
height="580px" [getNodeDefaults]="getNodeDefaults">
    <e-nodes>
        <e-node id='node1' [offsetX]=150 [offsetY]=150
[tooltip]="tooltip" [constraints]="constraints">

```

```

        </e-node>
    </e-nodes>
</ejs-diagram>`,
    encapsulation: ViewEncapsulation.None
})
export class AppComponent {
    @ViewChild("diagram")
    public diagram?: DiagramComponent;
    public tooltip?: DiagramTooltipModel;
    public constraints?: NodeConstraints;
    public getNodeDefaults(node: NodeModel): NodeModel {
        node.height = 100;
        node.width = 100;
        ((node as NodeModel).style as ShapeStyleModel).fill = "#6BA5D7";
        ((node as NodeModel).style as ShapeStyleModel).strokeColor =
"White";
        return node;
    }
    ngOnInit(): void {
        this.tooltip = {
            //Sets the content of the Tooltip
            content: 'Node1',
            //Sets the position of the Tooltip
            position: 'BottomRight',
            //Sets the tooltip position relative to the node
            relativeMode: 'Mouse'
        }
        this.constraints = NodeConstraints.Default | NodeConstraints.Tooltip
    }
}

```

MAIN.TS

```

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

```

Tooltip animation

To animate the tooltip, a set of specific animation effects are available, and it can be controlled by using the [animation](#) property. The animation property also allows you to set delay, duration, and various other effects of your choice.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, OnInit, ViewEncapsulation, ViewChild } from
'@angular/core';
import { DiagramComponent, Diagram, NodeModel, NodeConstraints,
DiagramTooltipModel, ShapeStyleModel } from '@syncfusion/ej2-angular-
diagrams';
@Component({
    imports: [

```

```

    DiagramModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%"
height="580px" [getNodeDefaults]="getNodeDefaults">
  <e-nodes>
    <e-node id='node1' [offsetX]=150 [offsetY]=150
[tooltip]="tooltip" [constraints]="constraints">
    </e-node>
  </e-nodes>
</ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram?: DiagramComponent;
  public tooltip?: DiagramTooltipModel;
  public constraints?: NodeConstraints;
  public getNodeDefaults(node: NodeModel): NodeModel {
    node.height = 100;
    node.width = 100;
    ((node as NodeModel).style as ShapeStyleModel).fill = "#6BA5D7";
    ((node as NodeModel).style as ShapeStyleModel).strokeColor =
"White";
    return node;
  }
  ngOnInit(): void {
    this.tooltip = {
      content: 'Node1',
      position: 'BottomCenter',
      relativeMode: 'Object',
      animation: {
        //Animation settings to be applied on the Tooltip, while it
is being shown over the target.
        open: {
          //Animation effect on the Tooltip is applied during open
and close actions.
          effect: 'ZoomIn',
          //Duration of the animation that is completed per
animation cycle.
          duration: 1000,
          //Indicating the waiting time before animation begins.
          delay: 0
        },
        //Animation settings to be applied on the Tooltip, when it
is closed.
        close: {
          effect: 'ZoomOut',
          duration: 500,
          delay: 0
        }
      }
    }
    this.constraints = NodeConstraints.Default | NodeConstraints.Tooltip
  }
}

```

```
}

```

MAIN.TS

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

User handle in Angular Diagram component

- User handles are used to add some frequently used commands around the selector. To create user handles, define and add them to the [userHandles](#) collection of the [selectedItems](#) property.
- The name property of user handle is used to define the name of the user handle and its further used to find the user handle at runtime and do any customization.

Alignment

User handles can be aligned relative to the node boundaries. It has [margin](#), [offset](#), [side](#), [horizontalAlignment](#), and [verticalAlignment](#) settings. It is quite tricky when all four alignments are used together but gives more control over alignment.

Offset for user handle

The [offset](#) property of [userHandles](#) is used to align the user handle based on fractions. 0 represents top/left corner, 1 represents bottom/right corner, and 0.5 represents half of width/height.

Side

The [side](#) property of [userHandles](#) is used to align the user handle by using the [Top](#), [Bottom](#), [Left](#), and [Right](#) options.

Horizontal and vertical alignments

The [horizontalAlignment](#) property of [userHandles](#) is used to set how the user handle is horizontally aligned at the position based on the [offset](#). The [verticalAlignment](#) property is used to set how user handle is vertically aligned at the position.

Margin for user handle

Margin is an absolute value used to add some blank space in any one of its four sides. The [userHandles](#) can be displaced with the [margin](#) property.

Appearance

The appearance of the user handle can be customized by using the [size](#), [borderColor](#), [backgroundColor](#), [visible](#), [pathData](#), and [pathColor](#) properties of the [userHandles](#).

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import { DiagramComponent, SelectorModel, IElement, randomId, cloneObject,
UserHandleModel, SelectorConstraints, ToolBase, NodeModel, Diagram,
MoveTool, ShapeStyleModel, MouseEventArgs, ConnectorModel } from
 '@syncfusion/ej2-angular-diagrams';
@Component({
```

```

imports: [
    DiagramModule
],
providers: [ ],
standalone: true,
selector: "app-container",
template: `<ejs-diagram #diagram id="diagram" width="100%" height="580px"
[getNodeDefaults]='getNodeDefaults' [getCustomTool]='getCustomTool'
[selectedItems]='selectedItems'>
    <e-nodes>
        <e-node id='node1' [offsetX]=150 [offsetY]=150></e-node>
    </e-nodes>
</ejs-diagram>`,
encapsulation: ViewEncapsulation.None
})
export class AppComponent {
    @ViewChild("diagram")
    public diagram?: DiagramComponent;
    public handles: UserHandleModel[] = [
        {
            name: "clone",
            pathData: "M60.3,18H27.5c-3,0-5.5,2.4-5.5,5.5v38.2h5.5V23.5h32.7V18z
M68.5, 28.9h-30c-3, 0-5.5,2.4-5.5,5.5v38.2c0,3,2.4,5.5,5.5,5.5h30c3,0,5.5-
2.4,5.5-5.5V34.4C73.9,31.4,71.5,28.9,68.5,28.9z M68.5,72.5h-
30V34.4h30V72.5z",
            visible: true,
            offset: 0,
            side: "Bottom",
            margin: { top: 0, bottom: 0, left: 0, right: 0 }
        }
    ];
    public selectedItems: SelectorModel = {
        constraints: SelectorConstraints.UserHandle,
        userHandles: this.handles
    };
    public getNodeDefaults(node: NodeModel): NodeModel {
        node.height = 100;
        node.width = 100;
        ((node as NodeModel).style as ShapeStyleModel).fill = "#6BA5D7";
        ((node as NodeModel).style as ShapeStyleModel).strokeColor = "#6BA5D7";
        return node;
    }
    public getCustomTool: Function = this.getTool.bind(this);
    public getTool(action: string): ToolBase {
        let tool: ToolBase = new ToolBase({} as any);
        if (action === "clone") {
            let cloneTool: CloneTool = new CloneTool((this.diagram as
Diagram).commandHandler);
            (cloneTool as CloneTool).diagram = this.diagram as Diagram;
            return cloneTool;
        }
        return tool;
    }
}
//Defines the clone tool used to copy Node/Connector.
class CloneTool extends MoveTool {
    public diagram?: Diagram = undefined;

```

```

public override mouseDown(args: MouseEventArgs): void {
    let newObject: NodeModel | ConnectorModel;
    if (((this.diagram as Diagram).selectedItems as SelectorModel).nodes as
NodeModel[]).length > 0) {
        newObject = cloneObject((((this.diagram as Diagram).selectedItems as
SelectorModel).nodes as NodeModel[])[0]) as NodeModel;
    } else {
        newObject = cloneObject((((this.diagram as Diagram).selectedItems as
SelectorModel).connectors as ConnectorModel[])[0]) as ConnectorModel;
    }
    newObject.id += randomId();
    (this.diagram as Diagram).paste([newObject]);
    args.source = (this.diagram as Diagram).nodes[(this.diagram as
Diagram).nodes.length - 1] as IElement;
    args.sourceWrapper = args.source.wrapper;
    super.mouseDown(args);
    this.inAction = true;
}
}

```

MAIN.TS

```

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

```

Fixed user handles

Fixed user handles are used to add some frequently used commands around the node and connector even without selecting it.

Initialization an fixed user handles

To create fixed user handles, define and add them to the collection of nodes and connectors property. The following code example used to create an fixed user handles for the nodes/connectors.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import { DiagramComponent, Diagram, NodeModel, ShapeStyleModel, MarginModel
} from '@syncfusion/ej2-angular-diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%"
height="580px" [getNodeDefaults] ='getNodeDefaults'>
    <e-nodes>
        <e-node id='node1' [offsetX]=150 [offsetY]=150>
            <e-node-fixeduserhandles>

```

```

                                <e-node-fixeduserhandle [margin]='margin1'
pathData='M60.3,18H27.5c-3,0-5.5,2.4-5.5,5.5v38.2h5.5V23.5h32.7V18z
M68.5,28.9h-30c-3,0-5.5,2.4-5.5,5.5v38.2c0,3,2.4,5.5,5.5,5.5h30c3,0,5.5-
2.4,5.5-5.5V34.4C73.9,31.4,71.5,28.9,68.5,28.9z M68.5,72.5h-
30V34.4h30V72.5z'>
                                </e-node-fixeduserhandle>
                                </e-node-fixeduserhandles>
        </e-node>
    </e-nodes>
</ejs-diagram>`,
    encapsulation: ViewEncapsulation.None
})
export class AppComponent {
    @ViewChild("diagram")
    public margin1?: MarginModel;
    ngOnInit(): void {
        this.margin1 = { right: 20 };
    }
    public diagram?: DiagramComponent;
    public getNodeDefaults(node: NodeModel): NodeModel {
        node.height = 100;
        node.width = 100;
        ((node as NodeModel).style as ShapeStyleModel).fill = "#6BA5D7";
        ((node as NodeModel).style as ShapeStyleModel).strokeColor =
"White";
        return node;
    }
}

```

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 id property of fixed user handle is used to define the unique identification of the fixed user handle and its further used to add custom events to the fixed user handle.
- The fixed user handle can be positioned relative to the node and connector boundaries. It has offset, padding and cornerRadius settings. It is used to position and customize the fixed user handle.
- The **Padding** is used to leave the space that's inside the fixed user handle between the icon and the border.
- Corner radius allows to create fixed user handles with rounded corners. The radius of the rounded corner is set with the **cornerRadius** property.

Note: PathData should need to be provided to render fixed user handle.

Size

Diagram allows you set size for fixed user handles by using the `width` and `height` property. The default value of the width and height property is 10.

Style

- You can change the style of the fixed user handles with the specific properties of `borderColor`, `borderWidth`, and background color using `handleStrokeColor`, `handleStrokeWidth`, and fill properties and the icon `borderColor` and `borderWidth` using `iconStrokeColor` and `iconStrokeWidth`.
- The fixed user handle's `iconStrokeColor` and `iconStrokeWidth` property used to change the stroke color and stroke width of the given `pathData`.
- The fixed user handle `handleStrokeColor`, `fill` properties are used to define the background color and border color of the userhandle and the `handleStrokeWidth` property is used to define the border width of the fixed user handle.
- The `visible` property of the fixed user handle enables or disables the visibility of fixed user handle.

The following code explains how to customize the appearance of the fixed user handles.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, OnInit, ViewChild } from '@angular/core';
import { DiagramComponent, Diagram, NodeModel, ConnectorModel, PointModel, MarginModel, ShapeStyleModel } from '@syncfusion/ej2-angular-diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%" height="580px" [getNodeDefaults] = 'getNodeDefaults' [getConnectorDefaults] = 'getConnectorDefaults'>
    <e-nodes>
      <e-node id='node1' [offsetX]=150 [offsetY]=150>
        <e-node-fixeduserhandles>
          <e-node-fixeduserhandle [width]=20 [height]=20 iconStrokeColor='white' fill='black' [margin]='margin1' [padding]='padding1' pathData='M60.3,18H27.5c-3,0-5.5,2.4-5.5,5.5v38.2h5.5V23.5h32.7V18z M68.5,28.9h-30c-3,0-5.5,2.4-5.5,5.5v38.2c0,3,2.4,5.5,5.5,5.5h30c3,0,5.5-2.4,5.5-5.5V34.4C73.9,31.4,71.5,28.9,68.5,28.9z M68.5,72.5h-30V34.4h30V72.5z'>
        </e-node-fixeduserhandle>
      </e-node-fixeduserhandles>
    </e-node>
  </e-nodes>
</e-connectors>`
})
```



```

        <e-connector id='connector' type='Orthogonal'
[sourcePoint]='sourcePoint1' [targetPoint]='targetPoint1'>
            <e-connector-fixeduserhandles>
                <e-connector-fixeduserhandle iconStrokeColor='white'
[padding]='padding1' fill='black' [width]=20 [height]=20
pathData='M60.3,18H27.5c-3,0-5.5,2.4-5.5,5.5v38.2h5.5V23.5h32.7V18z
M68.5,28.9h-30c-3,0-5.5,2.4-5.5,5.5v38.2c0,3,2.4,5.5,5.5,5.5h30c3,0,5.5-
2.4,5.5-5.5V34.4C73.9,31.4,71.5,28.9,68.5,28.9z M68.5,72.5h-
30V34.4h30V72.5z'>
            </e-connector-fixeduserhandle>
        </e-connector-fixeduserhandles>
    </e-connector>
</e-connectors>
</ejs-diagram>`,
    encapsulation: ViewEncapsulation.None
}))
export class AppComponent {
    @ViewChild("diagram")
    public diagram?: DiagramComponent;
    public sourcePoint1?: PointModel;
    public margin1?: MarginModel;
    public padding1?: MarginModel;
    targetPoint1?: Object;
    ngOnInit(): void {
        this.sourcePoint1 = { x: 300, y: 100 };
        this.targetPoint1 = { x: 400, y: 200 };
        this.margin1 = { right: 20 };
        this.padding1 = { left: 2, right: 2, top: 2, bottom: 2 };
    }
    public getNodeDefaults(node: NodeModel): NodeModel {
        node.height = 100;
        node.width = 100;
        ((node as NodeModel).style as ShapeStyleModel).fill = "#6BA5D7";
        ((node as NodeModel).style as ShapeStyleModel).strokeColor =
"White";
        return node;
    }
    public getConnectorDefaults(obj: ConnectorModel): void {
        obj.style = {
            strokeColor: '#6BA5D7',
            fill: '#6BA5D7',
            strokeWidth: 2
        }
        obj.targetDecorator = {
            style: {
                fill: '#6BA5D7',
                strokeColor: '#6BA5D7'
            }
        }
    }
}

```

MAIN.TS

```

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

```

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

Note: Fixed user handle id should need to be unique.

Customizing node fixed user handle

The node fixed user handle can be aligned relative to the node boundaries. It has `margin` and `offset` settings. It is quite useful to position the node fixed userhandle and used together and gives you more control over the node fixed user handle positioning.




Margin for node fixed user handle

Margin is an absolute value used to add some blank space in any one of its four sides. The fixed user handle can be displaced with the `margin` property.

Offset for node fixed user handle

The `offset` property of fixed user handle is used to align the user handle based on `x` and `y` points. (0,0) represents top/left corner and (1,1) represents bottom/right corner.

The following table shows all the possible alignments visually shows the fixed user handle positions.

Offset	Margin	Output
(0,0)	Right = 20	
(0.5,0)	Bottom = 20	
(1,0)	Left = 20	



| (0,0.5) | Right = 20 |



| (0,1) | Left = 20 |



| (0,1) | Right = 20 |



| (0.5,1) | Top = 20 |



| (1,1) | Left = 20 |

The following code explains how to customize the node fixed user handle.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import { DiagramComponent, Diagram, NodeModel, MarginModel, ShapeStyleModel
} from '@syncfusion/ej2-angular-diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%"
height="580px" [getNodeDefaults] = 'getNodeDefaults'>
  <e-nodes>
    <e-node id='node1' [offsetX]=150 [offsetY]=150>
      <e-node-fixeduserhandles>
        <e-node-fixeduserhandle [width]=20 [height]=20
[margin]='margin1' pathData='M60.3,18H27.5c-3,0-5.5,2.4-
5.5,5.5v38.2h5.5V23.5h32.7V18z M68.5,28.9h-30c-3,0-5.5,2.4-
5.5,5.5v38.2c0,3,2.4,5.5,5.5,5.5h30c3,0,5.5-2.4,5.5-
5.5V34.4C73.9,31.4,71.5,28.9,68.5,28.9z M68.5,72.5h-30V34.4h30V72.5z'>
        </e-node-fixeduserhandle>
      </e-node-fixeduserhandles>
    </e-node>
  </e-nodes>
</ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild("diagram")
  public margin1?: MarginModel;
  ngOnInit(): void {
    this.margin1 = { right: 20 };
  }
  public diagram?: DiagramComponent;
  public getNodeDefaults(node: NodeModel): NodeModel {
    node.height = 100;
    node.width = 100;
    ((node as NodeModel).style as ShapeStyleModel).fill = "#6BA5D7";
    ((node as NodeModel).style as ShapeStyleModel).strokeColor =
    "White";
    return node;
  }
}

```

MAIN.TS

```

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

```

Customizing connector fixed user handle

- The connector fixed user handle can be aligned relative to the connector boundaries. It has alignment, displacement and offset settings. It is useful to position the connector fixed userhandle and used together and gives you more control over the connector fixed user handle positioning.
- The `offset` and `alignment` properties of fixed user handle allows you to align the connector fixed user handles with respect to the segments.

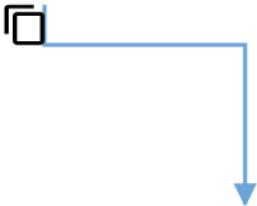
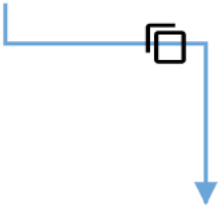
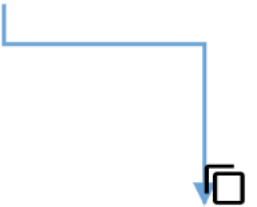
Offset for connector fixed user handle

The `offset` property of connector fixed user handle is used to align the user handle based on fractions. 0 represents the connector source point, 1 represents the connector target point, and 0.5 represents the center point of the connector segment.

Alignment

The connector's fixed user handle can be aligned over its segment path using the `alignment` property of fixed user handle.


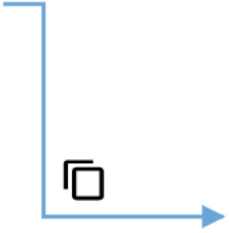
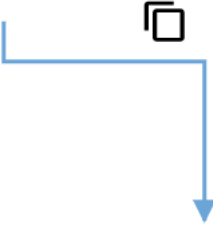
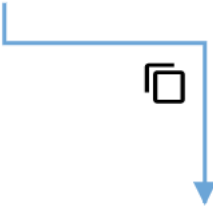
The following table shows all the possible alignments visually shows the fixed user handle positions.

Offset	Alignment	Output
0	Before	
0.5	Center	
1	After	

Displacement

The `displacement` property allows you to specify the space to be left from the connector segment based on the x and y value provided.

The following table shows all the possible alignments visually shows the fixed user handle positions.

Displacment	Alignment	Output
-----	-----	-----
x=10 Before		
x=10 After		
y=10 Before		
y=10 After		

Note: Displacement will not be done if the alignment is set to be center.

The following code explains how to customize the connector fixed user handle.

APP.COMPONENT.TS

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

```

import { DiagramModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, OnInit, ViewChild } from
 '@angular/core';
import { DiagramComponent, Diagram, ConnectorModel, PointModel } from
 '@syncfusion/ej2-angular-diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%"
height="580px" [getConnectorDefaults] ='getConnectorDefaults'>
    <e-connectors>
      <e-connector id='connector' type='Orthogonal'
[sourcePoint]='sourcePoint1' [targetPoint]='targetPoint1'>
        <e-connector-fixeduserhandles>
          <e-connector-fixeduserhandle [width]=20 [height]=20
pathData='M60.3,18H27.5c-3,0-5.5,2.4-5.5,5.5v38.2h5.5V23.5h32.7V18z
M68.5,28.9h-30c-3,0-5.5,2.4-5.5,5.5v38.2c0,3,2.4,5.5,5.5h30c3,0,5.5-
2.4,5.5-5.5V34.4C73.9,31.4,71.5,28.9,68.5,28.9z M68.5,72.5h-
30V34.4h30V72.5z'>
          </e-connector-fixeduserhandle>
        </e-connector-fixeduserhandles>
      </e-connector>
    </e-connectors>
  </ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram?: DiagramComponent;
  public sourcePoint1?: PointModel;
  public targetPoint1?: Object;
  ngOnInit(): void {
    this.sourcePoint1 = { x: 300, y: 100 };
    this.targetPoint1 = { x: 400, y: 200 };
  }
  public getConnectorDefaults(obj: ConnectorModel): void {
    obj.style = {
      strokeColor: '#6BA5D7',
      fill: '#6BA5D7',
      strokeWidth: 2
    }
    obj.targetDecorator = {
      style: {
        fill: '#6BA5D7',
        strokeColor: '#6BA5D7'
      }
    }
  }
}

```

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 Diagram component

Customizing the connector end point handle

Use the following CSS to customize the connector end point handle.

```
`scss
.e-diagram-endpoint-handle {
  fill: red;
  stroke: green;
}
```

Customizing the connector end point handle when connected

Use the following CSS to customize the connector end point handle when connected.

```
`scss
.e-diagram-endpoint-handle.e-connected {
  fill: red;
  stroke: green;
}
```

Customizing the connector end point handle when disabled

Use the following CSS to customize the connector end point handle when disabled.

```
`scss
.e-diagram-endpoint-handle.e-disabled {
  fill: red;
  opacity: 1;
  stroke: green;
}
```

Customizing the bezier connector handle

Use the following CSS to customize the bezier handle properties.

```
`scss
.e-diagram-bezier-handle {
  fill: red;
```



```
stroke: green;  
}
```

,

Customizing the bezier connector line

Use the following CSS to customize the bezier line properties.

```
`scss  
.e-diagram-bezier-line {  
stroke: black;  
}
```

,

Customizing the resize handle

Use the following CSS to customize the resize handle.

```
`scss  
.e-diagram-resize-handle {  
fill: white;  
opacity: 1;  
stroke: white;  
}
```

,

Customizing the selector pivot line

Use the following CSS to customize the line between the selector and rotate handle.

```
`scss  
.e-diagram-pivot-line {  
stroke: red;  
}
```

,

Customizing the selector border

Use the following CSS to customize the selector border.

```
`scss  
.e-diagram-border {  
stroke: red;  
}
```

,

Customizing the rotate handle

Use the following CSS to customize the rotate handle properties.

```
`scss
.e-diagram-rotate-handle {
  fill: red;
  stroke: green;
}
`
```

Customizing the symbolpalette while hovering

Use the following CSS to customize the symbolpalette while hovering.

```
`scss
.e-symbolpalette .e-symbol-hover:hover {
  background: red;
}
`
```

Customizing the symbolpalette when selected

Use the following CSS to customize the symbolpalette when selected.

```
`scss
.e-symbolpalette .e-symbol-selected {
  background: white;
}
`
```

Customizing the ruler

Use the following CSS to customize the ruler properties.

```
`scss
.e-diagram .e-ruler {
  background-color: red;
  font-size: 13px;
}
`
```

Customizing the ruler overlap

Use the following CSS to ruler overlap properties.

```
`scss
.e-diagram .e-ruler-overlap {
```

```
background-color: red;  
}
```

,

Customizing the text edit

Use the following CSS to customize the text edit properties.

```
`scss  
.e-diagram .e-diagram-text-edit {  
background: white;  
border-color: red;  
border-style: dashed;  
border-width: 1px;  
box-sizing: content-box;  
color: black;  
min-width: 50px;  
}
```

,

Customizing the text edit on selection

Use the following CSS to customize the text edit on selection properties.

```
`scss  
.e-diagram-text-edit::selection {  
background: red;  
color: green;  
}
```

,

Ruler in Angular Diagram component

The Ruler provides a horizontal and vertical guide for measuring in the Diagram control. The Ruler can be used to measure the diagram objects, indicate positions, and align diagram elements. This is especially useful in creating scale models.

Adding Rulers to the Diagram

- The [rulerSettings](#) property is used to control the visibility and appearance of the ruler in the diagram.
- The RulerSettings [showRulers](#) property is used to show or hide the rulers in the diagram.
- The RulerSettings [horizontalRuler](#) and [verticalRuler](#) properties are used to customize the rulers appearance in the diagram.

The following code shows how to add a ruler to the diagram.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule, OverviewModule, DataBindingService,
HierarchicalTreeService } from '@syncfusion/ej2-angular-diagrams'
import { Component, OnInit, ViewEncapsulation, ViewChild } from
'@angular/core';
import { DiagramComponent, OverviewComponent, Diagram, NodeModel,
ConnectorModel, OverviewModel, SnapSettingsModel, LayoutModel,
DataSourceModel, RulerSettingsModel, } from '@syncfusion/ej2-angular-
diagrams';
import { DataManager, Query } from '@syncfusion/ej2-data';
@Component({
  imports: [
    DiagramModule, OverviewModule
  ],
  providers: [DataBindingService, HierarchicalTreeService],
  standalone: true,
  selector: "app-container",
  template: `<div><ejs-diagram #diagram id="diagram" width="100%"
height="600px" [rulerSettings]='rulerSettings'></ejs-diagram></div>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild('diagram')
  public diagram?: DiagramComponent;
  public rulerSettings: RulerSettingsModel = { showRulers: 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 Ruler

By default, the ruler segments are arranged based on pixel values.

- The HorizontalRuler's [interval](#) property allows you to define the interval between ruler segments and the [segmentWidth](#) property allows you to define the segment width of the ruler. Similarly, you can use the VerticalRuler's [interval](#) and [segmentWidth](#) properties are used to define the interval and segment width of the vertical ruler
- The HorizontalRuler's [tickAlignment](#) property is used to align the ruler tick either left or right side of the ruler. The VerticalRuler's [tickAlignment](#) property is used to align the ruler tick either top or bottom side of the ruler.
- The HorizontalRuler's [arrangeTick](#) and VerticalRuler's [arrangeTick](#) function is provided for the purpose of customizing the appearance of ruler ticks. It will be called for each tick rendering.
- The HorizontalRuler's [markerColor](#) and VerticalRuler's [markerColor](#) properties are used to define the ruler marker color and marker will be shown when performing the interaction in the diagram.

The following code shows how the diagram ruler can be customized.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule, OverviewModule, DataBindingService,
HierarchicalTreeService } from '@syncfusion/ej2-angular-diagrams'
import { Component, OnInit, ViewEncapsulation, ViewChild } from
'@angular/core';
import { DiagramComponent, OverviewComponent, Diagram, NodeModel,
ConnectorModel, OverviewModel, SnapSettingsModel, LayoutModel,
DataSourceModel, RulerSettingsModel, } from '@syncfusion/ej2-angular-
diagrams';
import { DataManager, Query } from '@syncfusion/ej2-data';
@Component({
  imports: [
    DiagramModule, OverviewModule
  ],
  providers: [DataBindingService, HierarchicalTreeService],
  standalone: true,
  selector: "app-container",
  template: `<div><ejs-diagram #diagram id="diagram" width="100%"
height="600px" [rulerSettings]='rulerSettings'></ejs-diagram></div>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild('diagram')
  public diagram?: DiagramComponent;
  public rulerSettings: RulerSettingsModel = { showRulers: true,
horizontalRuler:{interval:8, segmentWidth:100, thickness:25,
tickAlignment:"LeftOrTop"},verticalRuler:{interval:10, segmentWidth:150,
thickness:35, tickAlignment:"RightOrBottom"} }
}
```

MAIN.TS

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

Note : The MarkerColor property can be customized using the [marker](#) CSS style.

Layers in Angular Diagram component

Layer is used to organize related shapes on a diagram control. A layer is a named category of shapes. By assigning shapes to different layers, you can selectively view, remove, and lock different categories of shapes.

In diagram, [Layers](#) provide a way to change the properties of all shapes that have been assigned to that layer. The following properties can be set.

- Visible
- Lock

- Objects
- AddInfo

Visible

The layer's [visible](#) property is used to control the visibility of the elements assigned to the layer.

`typescript

```
import { Component, ViewEncapsulation, OnInit, ViewChild } from '@angular/core';
import { DiagramComponent, Diagram, NodeModel, ConnectorModel, PointModel, LayerModel } from
 '@syncfusion/ej2-angular-diagrams';

@Component({
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%" height="580px" [layers]="layers">
    <e-connectors>
    <e-connector id='connector' type='Straight' [sourcePoint]='sourcePoint' [targetPoint]='targetPoint'>
    </e-connector>
    </e-connectors>
    <e-nodes>
    <e-node id='node1' [offsetX]=150 [offsetY]=150></e-node>
    </e-nodes>
  </ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram: DiagramComponent;
  public sourcePoint: PointModel;
  public targetPoint: PointModel;
  public pageSettings: PageSettingsModel;
  ngOnInit(): void {
    this.sourcePoint = { x: 100, y: 100 };
    this.targetPoint = { x: 200, y: 200 };
    this.layers = {
      id: 'layer1',
      visible: true,
      objects: ['node1', 'connector']
    }
  }
}
```

```

}
}
}
`

```

Lock

The layer's [lock](#) property is used to prevent or allow changes to the elements dimension and position.

`typescript

```

import { Component, ViewEncapsulation, OnInit, ViewChild } from '@angular/core';
import { DiagramComponent, Diagram, NodeModel, ConnectorModel, PointModel, LayerModel } from
 '@syncfusion/ej2-angular-diagrams';
@Component({
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%" height="580px" [layers]="layers">
    <e-connectors>
    <e-connector id='connector' type='Straight' [sourcePoint]='sourcePoint' [targetPoint]='targetPoint'>
    </e-connector>
    </e-connectors>
    <e-nodes>
    <e-node id='node1' [offsetX]=150 [offsetY]=150></e-node>
    <e-node id='node2' [offsetX]=350 [offsetY]=350></e-node>
    </e-nodes>
  </ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram: DiagramComponent;
  public sourcePoint: PointModel;
  public targetPoint: PointModel;
  public layers: LayerModel[];
  ngOnInit(): void {
    this.sourcePoint = { x: 100, y: 100 };
    this.targetPoint = { x: 200, y: 200 };
    this.layers = [{

```

```

id: 'layer1',
visible: true,
objects: ['node1', 'connector'],
lock: true
},
{
id: 'layer2',
visible: true,
objects: ['node2'],
lock: false
}];
}
}
,

```

Objects

The layer's [objects](#) property defines the diagram elements to the layer.

```

`typescript

import { Component, ViewEncapsulation, OnInit, ViewChild } from '@angular/core';
import { DiagramComponent, Diagram, NodeModel, ConnectorModel, PointModel, LayerModel } from
 '@syncfusion/ej2-angular-diagrams';

@Component({
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%" height="580px" [layers]="layers">
    <e-connectors>
    <e-connector id='connector' type='Straight' [sourcePoint]='sourcePoint' [targetPoint]='targetPoint'>
    </e-connector>
    </e-connectors>
    <e-nodes>
    <e-node id='node1' [offsetX]=150 [offsetY]=150></e-node>
    <e-node id='node2' [offsetX]=350 [offsetY]=350></e-node>
    </e-nodes>
  </ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})

```



```
export class AppComponent {  
  @ViewChild("diagram")  
  public diagram: DiagramComponent;  
  public sourcePoint: PointModel;  
  public targetPoint: PointModel;  
  public layers: LayerModel[];  
  ngOnInit(): void {  
    this.sourcePoint = { x: 100, y: 100 };  
    this.targetPoint = { x: 200, y: 200 };  
    this.layers = [{  
      id: 'layer1',  
      visible: true,  
      objects: ['node1', 'connector'],  
      lock: true  
    },  
    {  
      id: 'layer2',  
      visible: true,  
      objects: ['node2'],  
      lock: false  
    }  
  ];  
}
```

AddInfo

The [addInfo](#) property of layers allow you to maintain additional information to the layers.

The following code illustrates how to add additional information to the layers.

`typescript

```
import { Component, ViewEncapsulation, OnInit, ViewChild } from '@angular/core';  
import { DiagramComponent, Diagram, NodeModel, ConnectorModel, PointModel, LayerModel } from  
'@syncfusion/ej2-angular-diagrams';  
@Component({  
  selector: "app-container",  
  template: `<ejs-diagram #diagram id="diagram" width="100%" height="580px" [layers]="layers">
```

```

<e-connectors>
<e-connector id='connector' type='Straight' [sourcePoint]='sourcePoint' [targetPoint]='targetPoint'>
</e-connector>
</e-connectors>
<e-nodes>
<e-node id='node1' [offsetX]=150 [offsetY]=150></e-node>
<e-node id='node2' [offsetX]=350 [offsetY]=350></e-node>
</e-nodes>
</ejs-diagram>`,
encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram: DiagramComponent;
  public sourcePoint: PointModel;
  public targetPoint: PointModel;
  public layers: LayerModel[];
  public addInfo: Object = { Description: 'Layer1' };
  ngOnInit(): void {
    this.sourcePoint = { x: 100, y: 100 };
    this.targetPoint = { x: 200, y: 200 };
    this.layers = [{
      id: 'layer1',
      visible: true,
      objects: ['node1', 'connector'],
      lock: true,
      addInfo: this.addInfo
    },
    {
      id: 'layer2',
      visible: true,
      objects: ['node2'],
      lock: false
    }
  ]
}

```

```
});  
}  
}  
,
```

Add layer at runtime

Layers can be added at runtime by using the [addLayer](#) public method.

The layer's [ID](#) property defines the ID of the layer, and its further used to find the layer at runtime and do any customization.

The following code illustrates how to add a layer.

```
`typescript  
// add the layers to the existing diagram layer collection  
this.diagram.addLayer({  
  id: 'newlayer',  
  objects: [],  
  visible: true,  
  lock: false,  
  zIndex: -1  
}, [{  
  type: 'Straight',  
  sourcePoint: {  
    x: 100,  
    y: 300  
  },  
  targetPoint: {  
    x: 200,  
    y: 400  
  }  
}]);  
,
```

Remove layer at runtime

Layers can be removed at runtime by using the [removeLayer](#) public method.

The following code illustrates how to remove a layer.

```
`typescript  
// remove the diagram layers
```

```
this.diagram.removeLayer([diagram.model.layers[i]]);  
`
```

moveObjects

Objects of the layers can be moved by using the [moveObjects](#) public method.

The following code illustrates how to move objects from one layer to another layer from the diagram.

```
`typescript  
// move the objects of diagram layers  
this.diagram.moveObjects(['connector1'], 'layer2');  
`
```

bringLayerForward

Layers can be moved forward at runtime by using the [bringLayerForward](#) public method.

The following code illustrates how to bring forward to layer.

```
`typescript  
// move the layer forward  
this.diagram.bringLayerForward('layer1');  
`
```

sendLayerBackward

Layers can be moved backward at runtime by using the [sendLayerBackward](#) public method.

The following code illustrates how to send backward to layer.

```
`typescript  
// move the layer backward  
this.diagram.sendLayerBackward('layer1');  
`
```

cloneLayer

Layers can be cloned with its object by using the [cloneLayer](#) public method.

The following code illustrates how to bring forward to layer.

```
`typescript  
// clone a layer with its object  
this.diagram.cloneLayer('layer2');  
`
```

getActiveLayer

To get the active layers back in diagram, use the [getActiveLayer](#) public method.

The following code illustrates how to bring forward to layer.

```
`typescript  
// gets the active layer back
```

```
this.diagram.getActiveLayer();
```

```
,
```

[setActiveLayer](#)

Set the active layer by using the [setActiveLayer](#) public method.

The following code illustrates how to bring forward to layer.

```
`typescript
```

```
// set the active layer
```

```
//@param layerName defines the name of the layer which is to be active layer
```

```
this.diagram.setActiveLayer('layer2');
```

```
,
```

Context menu in Angular Diagram component

```
<!-- markdownlint-disable MD010 -->
```

In graphical user interface (GUI), a context menu is a type of menu that appears when you perform right-click operation. Nested level of context menu items can be created.

Diagram provides some in-built context menu items and allows to define custom menu items through the [contextMenuSettings](#) property

Customize context menu

The [show](#) property helps you to enable/disable the context menu. Diagram provides some default context menu items to ease the execution of some frequently used commands.

The following code illustrates how to enable the default context menu items.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule, DiagramContextMenuService } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, OnInit, ViewChild } from '@angular/core';
import { DiagramComponent, ShapeStyleModel } from '@syncfusion/ej2-angular-diagrams';
import { ContextMenuSettingsModel, Diagram, NodeModel, ConnectorModel } from '@syncfusion/ej2-diagrams';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [DiagramContextMenuService],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%" height="580px" [getNodeDefaults] ='getNodeDefaults' [getConnectorDefaults]='getConnectorDefaults' [contextMenuSettings]="contextMenuSettings">
    <e-nodes>
      <e-node id='node1' [offsetX]=150 [offsetY]=150>
```

```

        <e-node-annotations>
            <e-node-annotation id="label1" content="Rectangle1"
[horizontalAlignment]="horizontalAlignment">
            </e-node-annotation>
        </e-node-annotations>
    </e-node>
    <e-node id='node2' [offsetX]=350 [offsetY]=150>
        <e-node-annotations>
            <e-node-annotation id="label1" content="Rectangle2"
[horizontalAlignment]="horizontalAlignment">
            </e-node-annotation>
        </e-node-annotations>
    </e-node>
</e-nodes>
<e-connectors>
    <e-connector id='connector' type='Orthogonal' sourceID='node1'
targetID='node2'>
    </e-connector>
</e-connectors>
</ejs-diagram>`,
    encapsulation: ViewEncapsulation.None
})
export class AppComponent {
    @ViewChild("diagram")
    public diagram?: DiagramComponent;
    public contextMenuSettings?: ContextMenuSettingsModel
    horizontalAlignment: any;
    ngOnInit(): void {
        //Enables the context menu
        this.contextMenuSettings = {
            show: true,
        }
    }
    public getNodeDefaults(node: NodeModel): NodeModel {
        node.height = 100;
        node.width = 100;
        ((node as NodeModel).style as ShapeStyleModel).fill = "#6BA5D7";
        ((node as NodeModel).style as ShapeStyleModel).strokeColor =
"White";
        return node;
    }
    public getConnectorDefaults(obj: ConnectorModel): void {
        obj.style = {
            strokeColor: '#6BA5D7',
            fill: '#6BA5D7',
            strokeWidth: 2
        }
        obj.targetDecorator = {
            style: {
                fill: '#6BA5D7',
                strokeColor: '#6BA5D7'
            }
        }
    }
}

```

MAIN.TS

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

Context menu can be defined for individual node with the desired context menu items.

- Apart from the default context menu items, define some additional context menu items. Those additional items have to be defined and added to the [items](#) property of the context menu.
- Set text and ID for context menu item using the context menu [text](#) and [ID](#) properties respectively.
- Set an image for the context menu item using the context menu [url](#) property.
- The [iconCss](#) property defines the class/multiple classes separated by a space for the menu item that is used to include an icon. Menu item can include font icon and sprite image.
- The [target](#) property used to set the target to show the menu item.
- The [separator](#) property defines the horizontal lines that are used to separate the menu items. You cannot select the separators. You can enable separators to group the menu items using the separator property.

The following code example illustrates how to add custom context menu items.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule, DiagramContextMenuService } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, OnInit, ViewChild } from '@angular/core';
import { DiagramComponent, ShapeStyleModel } from '@syncfusion/ej2-angular-diagrams';
import { ContextMenuSettingsModel, Diagram, NodeModel, ConnectorModel } from '@syncfusion/ej2-diagrams';
import { MenuEventArgs } from '@syncfusion/ej2-navigations';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [DiagramContextMenuService],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%"
height="580px" [getNodeDefaults]='getNodeDefaults'
[getConnectorDefaults]='getConnectorDefaults'
[contextMenuSettings]="contextMenuSettings">
  <e-nodes>
    <e-node id='node1' [offsetX]=150 [offsetY]=150>
      <e-node-annotations>
        <e-node-annotation id="label1" content="Rectangle1"
[horizontalAlignment]="horizontalAlignment">
      </e-node-annotation>
    </e-node-annotations>
  </e-nodes>
</template>
```

```

        </e-node>
        <e-node id='node2' [offsetX]=350 [offsetY]=150>
            <e-node-annotations>
                <e-node-annotation id="label1" content="Rectangle2"
[horizontalAlignment]="horizontalAlignment">
                    </e-node-annotation>
                </e-node-annotations>
            </e-node>
        </e-nodes>
        <e-connectors>
            <e-connector id='connector' type='Orthogonal' sourceID='node1'
targetID='node2'>
                </e-connector>
            </e-connectors>
        </ejs-diagram>`,
        encapsulation: ViewEncapsulation.None
    })
    export class AppComponent {
        @ViewChild("diagram")
        public diagram?: DiagramComponent;
        public contextMenuSettings?: ContextMenuSettingsModel
        horizontalAlignment: any;
        ngOnInit(): void {
            //Enables the context menu
            this.contextMenuSettings = {
                //Enables the context menu
                show: true,
                // Defines the custom context menu items
                items: [{
                    // Text to be displayed
                    text: 'Save',
                    //Sets the id for the item
                    id: 'save',
                    //ContextMenu can be visible based on the target in
which you open the ContextMenu.
                    target: '.e-elementcontent',
                    // Sets the css icons for the item
                    iconCss: 'e-save'
                },
                {
                    text: 'Load',
                    id: 'load',
                    target: '.e-elementcontent',
                    iconCss: 'e-load'
                },
                {
                    text: 'Clear',
                    id: 'clear',
                    target: '.e-elementcontent',
                    iconCss: 'e-clear'
                }
            ],
            // Hides the default context menu items
            showCustomMenuOnly: false,
        }
    }
    public getNodeDefaults(node: NodeModel): NodeModel {

```



```

        node.height = 100;
        node.width = 100;
        ((node as NodeModel).style as ShapeStyleModel).fill = "#6BA5D7";
        ((node as NodeModel).style as ShapeStyleModel).strokeColor =
"White";
        return node;
    }
    public getConnectorDefaults(obj: ConnectorModel): void {
        obj.style = {
            strokeColor: '#6BA5D7',
            fill: '#6BA5D7',
            strokeWidth: 2
        }
        obj.targetDecorator = {
            style: {
                fill: '#6BA5D7',
                strokeColor: '#6BA5D7'
            }
        }
    }
}

```

MAIN.TS

```

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

```

To display the custom context menu items alone, set the [showCustomMenuOnly](#) property to true.

Template Support for Context menu

- Diagram provides template support for context menu. The context menu items can be customized by using the `contextMenuBeforeItemRender` event. The `contextMenuBeforeItemRender` event triggers while rendering each menu item.
- In the following sample, the menu item is rendered with key code for specified action in ContextMenu using the template. Here, the key code is specified for the cut and copy at right corner of the menu items by adding a span element in the `contextMenuBeforeItemRender` event.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule, DiagramContextMenuService } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation, OnInit, ViewChild } from
 '@angular/core';
import { createElement } from "@syncfusion/ej2-base";
import { DiagramComponent, ShapeStyleModel } from '@syncfusion/ej2-angular-
diagrams';
import { ContextMenuSettingsModel, Diagram, NodeModel, ConnectorModel } from
 '@syncfusion/ej2-diagrams';

```

```

import { MenuEventArgs } from '@syncfusion/ej2-navigations';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [DiagramContextMenuService],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%"
height="580px" [getNodeDefaults]='getNodeDefaults'
[getConnectorDefaults]='getConnectorDefaults'
[contextMenuSettings]="contextMenuSettings"
(contextMenuBeforeItemRender)='contextMenuBeforeItemRender($event)'>
    <e-nodes>
      <e-node id='node1' [offsetX]=150 [offsetY]=150>
        <e-node-annotations>
          <e-node-annotation id="label1" content="Rectangle1"
[horizontalAlignment]="horizontalAlignment">
        </e-node-annotation>
      </e-node-annotations>
    </e-node>
      <e-node id='node2' [offsetX]=350 [offsetY]=150>
        <e-node-annotations>
          <e-node-annotation id="label1" content="Rectangle2"
[horizontalAlignment]="horizontalAlignment">
        </e-node-annotation>
      </e-node-annotations>
    </e-node>
  </e-nodes>
  <e-connectors>
    <e-connector id='connector' type='Orthogonal' sourceID='node1'
targetID='node2'>
    </e-connector>
  </e-connectors>
</ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram?: DiagramComponent;
  public contextMenuSettings?: ContextMenuSettingsModel
horizontalAlignment: any;
  ngOnInit(): void {
    //Enables the context menu
    this.contextMenuSettings = {
      //Enables the context menu
      show: true,
      items: [{
        text: 'Cut ', id: 'cut', target: '.e-diagramcontent',
        iconCss: 'e-Cut'
      },
      {
        text: 'Copy ', id: 'copy', target: '.e-diagramcontent',
        iconCss: 'e-Copy'
      }
    ],
    // Hides the default context menu items
    showCustomMenuOnly: true,

```

```

    }
  }
  public getNodeDefaults(node: NodeModel): NodeModel {
    node.height = 100;
    node.width = 100;
    ((node as NodeModel).style as ShapeStyleModel).fill = "#6BA5D7";
    ((node as NodeModel).style as ShapeStyleModel).strokeColor =
    "White";
    return node;
  }
  public getConnectorDefaults(obj: ConnectorModel): ConnectorModel {
    obj.style = {
      strokeColor: '#6BA5D7',
      fill: '#6BA5D7',
      strokeWidth: 2
    }
    obj.targetDecorator = {
      style: {
        fill: '#6BA5D7',
        strokeColor: '#6BA5D7'
      }
    }
    return obj;
  }
  public contextMenuBeforeItemRender(args: MenuEventArgs) {
    // To render template in li.
    let shortCutSpan: HTMLElement = createElement('span');
    let text: string = args.item.text as string;
    let shortCutText: string = text === 'Cut ' ? 'Ctrl + S' : (text
    === 'Copy ' ?
    'Ctrl + U' : 'Ctrl + Shift + I');
    shortCutSpan.textContent = shortCutText;
    args.element.appendChild(shortCutSpan);
    shortCutSpan.setAttribute('class', 'shortcut');
  }
}

```

MAIN.TS

```

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

```

Context menu events

You would be notified with events, when you try to open the context menu items [contextMenuOpen](#) and when you click the menu items [contextMenuClick](#).

The following code example illustrates how to define those events.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule, DiagramContextMenuService } from '@syncfusion/ej2-
angular-diagrams'

```

```

import { Component, ViewEncapsulation, OnInit, ViewChild } from
'@angular/core';
import { DiagramComponent, ShapeStyleModel } from '@syncfusion/ej2-angular-
diagrams';
import { ContextMenuSettingsModel, DiagramBeforeMenuOpenEventArgs, Diagram,
NodeModel, ConnectorModel } from '@syncfusion/ej2-diagrams';
import { MenuEventArgs } from '@syncfusion/ej2-navigations';
@Component({
  imports: [
    DiagramModule
  ],
  providers: [DiagramContextMenuService],
  standalone: true,
  selector: "app-container",
  template: `<ejs-diagram #diagram id="diagram" width="100%"
height="580px" [getNodeDefaults] = 'getNodeDefaults'
[getConnectorDefaults]='getConnectorDefaults'
[contextMenuSettings]="contextMenuSettings"
(contextMenuOpen)="contextMenuOpen($event)"
(contextMenuClick)="contextMenuClick($event)">
    <e-nodes>
      <e-node id='node1' [offsetX]=150 [offsetY]=150>
        <e-node-annotations>
          <e-node-annotation id="label1" content="Rectangle1"
[horizontalAlignment]="horizontalAlignment">
        </e-node-annotation>
      </e-node-annotations>
    </e-node>
    <e-node id='node2' [offsetX]=350 [offsetY]=150>
      <e-node-annotations>
        <e-node-annotation id="label1" content="Rectangle2"
[horizontalAlignment]="horizontalAlignment">
      </e-node-annotation>
    </e-node-annotations>
  </e-node>
</e-nodes>
<e-connectors>
  <e-connector id='connector' type='Orthogonal' sourceID='node1'
targetID='node2'>
  </e-connector>
</e-connectors>
</ejs-diagram>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram?: DiagramComponent;
  public contextMenuSettings?: ContextMenuSettingsModel
  horizontalAlignment: any;
  ngOnInit(): void {
    //Enables the context menu
    let $this = this as any;
    this.contextMenuSettings = {
      //Enables the context menu
      show: true,
      items: [{
        text: 'delete',

```

```

        id: 'delete',
      ]],
      // Hides the default context menu items
      showCustomMenuOnly: false,
    } as ContextMenuSettingsModel;
  }

  public contextMenuOpen(args: DiagramBeforeMenuOpenEventArgs): void {
    for (let item of args.items) {
      if (item.text === 'delete') {
        if ((!this.diagram as any).selectedItems.nodes.length &&
          !(this.diagram as any).selectedItems.connectors.length) {
          args.hiddenItems.push(item.id as string);
        }
      }
    }
  }

  public contextMenuClick(args: MenuEventArgs): void {
    if (args.item.id === 'delete') {
      if (((this.diagram as any).selectedItems.nodes.length +
        (this.diagram as any).selectedItems.connectors.length) > 0) {
        (this.diagram as any).cut();
      }
    }
  }

  public getNodeDefaults(node: NodeModel): NodeModel {
    node.height = 100;
    node.width = 100;
    ((node as NodeModel).style as ShapeStyleModel).fill = "#6BA5D7";
    ((node as NodeModel).style as ShapeStyleModel).strokeColor =
    "White";
    return node;
  }

  public getConnectorDefaults(obj: ConnectorModel): void {
    obj.style = {
      strokeColor: '#6BA5D7',
      fill: '#6BA5D7',
      strokeWidth: 2
    }
    obj.targetDecorator = {
      style: {
        fill: '#6BA5D7',
        strokeColor: '#6BA5D7'
      }
    }
  }
}

```

MAIN.TS

```

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

```

Symbol palette in Angular Diagram component

The **SymbolPalette** displays a collection of palettes. The palette shows a set of nodes and connectors. It allows to drag and drop the nodes and connectors into the diagram.

Create symbol palette

The [width](#) and [height](#) properties of the symbol palette allows to define the size of the symbol palette.

```
`typescript
@Component({
  selector: "app-container",
  template: `<ejs-symbolpalette id="symbolpalette"width="100%" height="700px">
</ejs-symbolpalette>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {}
`
```

Add palettes to SymbolPalette

A palette allows to display a group of related symbols and it textually annotates the group with its header. A [Palettes](#) can be added as a collection of symbol groups.

The collection of predefined symbols can be added in palettes using the [symbols](#) property.

To initialize a palette, define a JSON object with the property [ID](#) that is unique ID is set to the palettes.

The [allowDrag](#) property allows the user to drag the symbol from the symbol palette.

The following code example illustrates how to define a palette and how its added to symbol palette.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule, SymbolPaletteModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, OnInit, ViewEncapsulation, ViewChild } from '@angular/core';
import { SymbolPaletteComponent, SymbolPalette, NodeModel, ConnectorModel, PaletteModel } from '@syncfusion/ej2-angular-diagrams';
import { ExpandMode } from '@syncfusion/ej2-navigations';
@Component({
  imports: [
    DiagramModule, SymbolPaletteModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  template: `<ejs-symbolpalette id="symbolpalette"width="100%"
height="700px" [expandMode]="expandMode" [palettes]="palettes"
[symbolHeight]=80 [symbolWidth]=80>
</ejs-symbolpalette>`,
  encapsulation: ViewEncapsulation.None
})
```

```
    })
  export class AppComponent {
    public expandMode?: ExpandMode;
    public palettes?: PaletteModel[];
    public getBasicShapes(): NodeModel[] {
      let basicShapes: NodeModel[] = [{
        id: 'Rectangle',
        shape: {
          type: 'Basic',
          shape: 'Rectangle'
        }
      },
      {
        id: 'Ellipse',
        shape: {
          type: 'Basic',
          shape: 'Ellipse'
        }
      },
      {
        id: 'Hexagon',
        shape: {
          type: 'Basic',
          shape: 'Hexagon'
        }
      }
    ];
    return basicShapes;
  };
  public getFlowShapes(): NodeModel[] {
    let flowShapes: NodeModel[] = [{
      id: 'process',
      shape: {
        type: 'Flow',
        shape: 'Process'
      }
    },
    {
      id: 'document',
      shape: {
        type: 'Flow',
        shape: 'Document'
      }
    },
    {
      id: 'predefinedprocess',
      shape: {
        type: 'Flow',
        shape: 'PreDefinedProcess'
      }
    }
  ];
  return flowShapes;
};
  public getConnectors(): ConnectorModel[] {
    let connectorSymbols: ConnectorModel[] = [{
      id: 'Link1',
```

```

        type: 'Orthogonal',
        sourcePoint: {
            x: 0,
            y: 0
        },
        targetPoint: {
            x: 40,
            y: 40
        },
        targetDecorator: {
            shape: 'Arrow'
        }
    },
    {
        id: 'Link21',
        type: 'Straight',
        sourcePoint: {
            x: 0,
            y: 0
        },
        targetPoint: {
            x: 40,
            y: 40
        },
        targetDecorator: {
            shape: 'Arrow'
        }
    },
    {
        id: 'link33',
        type: 'Bezier',
        sourcePoint: {
            x: 0,
            y: 0
        },
        targetPoint: {
            x: 40,
            y: 40
        },
        style: {
            strokeWidth: 2
        },
        targetDecorator: {
            shape: 'None'
        }
    }
];
return connectorSymbols;
};
ngOnInit(): void {
    this.expandMode = 'Multiple'
    this.palettes = [{
        //Sets the id of the palette
        id: 'flow',
        //Sets whether the palette expands/collapse its children
        expanded: true,
        //Adds the palette items to palette
    }];
}

```



```

        symbols: this.getFlowShapes(),
        //Sets the header text of the palette
        title: 'Flow Shapes',
        iconCss: 'e-ddb-icons e-flow'
    },
    {
        id: 'basic',
        expanded: true,
        symbols: this.getBasicShapes(),
        title: 'Basic Shapes',
        iconCss: 'e-ddb-icons e-basic'
    },
    {
        id: 'connectors',
        expanded: true,
        symbols: this.getConnectors(),
        title: 'Connectors',
        iconCss: 'e-ddb-icons e-connector'
    }
]
}
}

```

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 palette header

Palettes can be annotated with its header texts.

The [title](#) displayed as the header text of palette.

The [expanded](#) property of palette allows to expand/collapse its palette items.

The [height](#) property of palette sets the height of the symbol group.

The [iconCss](#) property sets the content of the symbol group.

The [description](#) defines the text to be displayed and how that is to be handled in `getSymbolInfo`.

Also, any HTML element into a palette header can be embedded by defining the `getSymbolInfo` property.

The following code example illustrates how to customize palette headers.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule, SymbolPaletteModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, OnInit, ViewEncapsulation, ViewChild } from '@angular/core';

```

```

import { SymbolPaletteComponent, SymbolPalette, NodeModel, ConnectorModel,
PaletteModel } from '@syncfusion/ej2-angular-diagrams';
import { ExpandMode } from '@syncfusion/ej2-navigations';
@Component({
  imports: [
    DiagramModule, SymbolPaletteModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  template: `<ejs-symbolpalette id="symbolpalette"width="100%"
height="700px" [expandMode]="expandMode" [palettes]="palettes"
[symbolHeight]=80 [symbolWidth]=80>
  </ejs-symbolpalette>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  public expandMode?: ExpandMode;
  public palettes?: PaletteModel[];
  public getBasicShapes(): NodeModel[] {
    let basicShapes: NodeModel[] = [{
      id: 'Rectangle',
      shape: {
        type: 'Basic',
        shape: 'Rectangle'
      }
    },
    {
      id: 'Ellipse',
      shape: {
        type: 'Basic',
        shape: 'Ellipse'
      }
    },
    {
      id: 'Hexagon',
      shape: {
        type: 'Basic',
        shape: 'Hexagon'
      }
    }
  ];
  return basicShapes;
};
  public getFlowShapes(): NodeModel[] {
    let flowShapes: NodeModel[] = [{
      id: 'process',
      shape: {
        type: 'Flow',
        shape: 'Process'
      }
    },
    {
      id: 'document',
      shape: {
        type: 'Flow',
        shape: 'Document'
      }
    }
  ];
  return flowShapes;
};

```

```

    },
    {
      id: 'predefinedprocess',
      shape: {
        type: 'Flow',
        shape: 'PreDefinedProcess'
      }
    }
  ];
  return flowShapes;
};

public getConnectors(): ConnectorModel[] {
  let connectorSymbols: ConnectorModel[] = [{
    id: 'Link1',
    type: 'Orthogonal',
    sourcePoint: {
      x: 0,
      y: 0
    },
    targetPoint: {
      x: 40,
      y: 40
    },
    targetDecorator: {
      shape: 'Arrow'
    }
  },
  {
    id: 'Link21',
    type: 'Straight',
    sourcePoint: {
      x: 0,
      y: 0
    },
    targetPoint: {
      x: 40,
      y: 40
    },
    targetDecorator: {
      shape: 'Arrow'
    }
  },
  {
    id: 'link33',
    type: 'Bezier',
    sourcePoint: {
      x: 0,
      y: 0
    },
    targetPoint: {
      x: 40,
      y: 40
    },
    style: {
      strokeWidth: 2
    }
  },

```

```

        targetDecorator: {
            shape: 'None'
        }
    };
    return connectorSymbols;
};
ngOnInit(): void {
    this.expandMode = 'Multiple'
    this.palettes = [{
        //Sets the id of the palette
        id: 'flow',
        //Sets whether the palette expands/collapse its children
        expanded: true,
        //Adds the palette items to palette
        symbols: this.getFlowShapes(),
        //Sets the header text of the palette
        title: 'Flow Shapes',
        iconCss: 'e-ddb-icons e-flow'
    },
    {
        id: 'basic',
        expanded: true,
        symbols: this.getBasicShapes(),
        title: 'Basic Shapes',
        iconCss: 'e-ddb-icons e-basic'
    },
    {
        id: 'connectors',
        expanded: true,
        symbols: this.getConnectors(),
        title: 'Connectors',
        iconCss: 'e-ddb-icons e-connector'
    }
    ]
}
}

```

MAIN.TS

```

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

```

Restrict expansion of the palette panel

The symbol palette panel can be restricted from getting expanded. The `cancel` argument of the `paletteExpanding` property defines whether the palette's panel should be expanded or collapsed. By default, the panel is expanded. This restriction can be done for each of the palettes in the symbol palette as desired.

In the following code example, the basic shapes palette is restricted from getting collapsed whereas the swimlane shapes palette can be expanded or collapsed.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule, SymbolPaletteModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation } from '@angular/core';
import { NodeModel, PaletteModel, SymbolPreviewModel } from '@syncfusion/ej2-angular-diagrams';
import { ExpandMode } from '@syncfusion/ej2-navigations';
@Component({
  imports: [
    DiagramModule, SymbolPaletteModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  template: `<ejs-symbolpalette id="symbolpalette"width="100%"
height="100%" [expandMode]="expandMode" [enableSearch]=true
[palettes]='palettes'
[symbolHeight]=80 [symbolWidth]=80 [symbolPreview]='symbolPreview'
(paletteExpanding)=paletteExpanding($event)`>
    </ejs-symbolpalette>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  public expandMode?: ExpandMode;
  public palettes?: PaletteModel[];
  public symbolPreview?: SymbolPreviewModel[];
  public paletteExpanding(args: any) {
    if(args.palette.id === 'basic') {
      // Basic shapes panel does not collapse
      args.cancel = true;
    } else {
      // Swimlane shapes panel collapse and expand
      args.cancel = false;
    }
  };
  public getBasicShapes(): NodeModel[] {
    let basicShapes: NodeModel[] = [{
      id: 'Rectangle',
      shape: {
        type: 'Basic',
        shape: 'Rectangle'
      }
    },
    {
      id: 'Ellipse',
      shape: {
        type: 'Basic',
        shape: 'Ellipse'
      }
    },
    {
      id: 'Hexagon',
      shape: {
        type: 'Basic',
        shape: 'Hexagon'
      }
    }
  ]
}

```

```

    }
    ];
    return basicShapes;
  };
  public getFlowShapes(): NodeModel[] {
    let flowShapes: NodeModel[] = [{
      id: 'process',
      shape: {
        type: 'Flow',
        shape: 'Process'
      }
    },
    {
      id: 'document',
      shape: {
        type: 'Flow',
        shape: 'Document'
      }
    }
  ];
  return flowShapes;
};
ngOnInit(): void {
  this.expandMode = 'Multiple'
  this.palettes = [{
    //Sets the id of the palette
    id: 'flow',
    //Sets whether the palette expands/collapse its children
    expanded: true,
    //Adds the palette items to palette
    symbols: this.getFlowShapes(),
    //Sets the header text of the palette
    title: 'Flow Shapes',
    iconCss: 'e-ddb-icons e-flow'
  },
  {
    id: 'basic',
    expanded: true,
    symbols: this.getBasicShapes(),
    title: 'Basic Shapes',
    iconCss: 'e-ddb-icons e-basic'
  }
  ],
  this.symbolPreview = [{
    height: 100,
    width: 100
  }]
}
}

```

MAIN.TS

```

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

```

Stretch the symbols into the palette

The [fit](#) property defines whether the symbol has to be fit inside the size, that is defined by the symbol palette. For example, when you resize the rectangle in the symbol, ratio of the rectangle size has to be maintained rather changing into square shape. The following code example illustrates how to customize the symbol size.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule, SymbolPaletteModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, OnInit, ViewEncapsulation, ViewChild } from '@angular/core';
import { SymbolPaletteComponent, SymbolPalette, NodeModel, ConnectorModel, PaletteModel } from '@syncfusion/ej2-angular-diagrams';
import { ExpandMode } from '@syncfusion/ej2-navigations';
@Component({
  imports: [
    DiagramModule, SymbolPaletteModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  template: `<ejs-symbolpalette id="symbolpalette"width="100%"
height="700px" [symbolHeight]=80 [symbolWidth]=80 [expandMode]="expandMode"
[palettes]="palettes" [getSymbolInfo]="getSymbolInfo">
  </ejs-symbolpalette>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  public expandMode?: ExpandMode;
  public palettes?: PaletteModel[];
  public getBasicShapes(): NodeModel[] {
    let basicShapes: NodeModel[] = [{
      id: 'Rectangle',
      shape: {
        type: 'Basic',
        shape: 'Rectangle'
      }
    },
    {
      id: 'Ellipse',
      shape: {
        type: 'Basic',
        shape: 'Ellipse'
      }
    },
    {
      id: 'Hexagon',
      shape: {
        type: 'Basic',
        shape: 'Hexagon'
      }
    }
  ]
}
```

```

    ];
    return basicShapes;
  };
  public getSymbolInfo() {
    // Enables to fit the content into the specified palette item size
    return {
      fit: true
    };
    // When it is set as false, the element is rendered with actual node
    size
  };
  ngOnInit(): void {
    this.expandMode = 'Multiple'
    this.palettes = [{
      id: 'basic',
      expanded: true,
      symbols: this.getBasicShapes(),
      title: 'Basic Shapes',
      iconCss: 'e-ddb-icons e-basic'
    }]
  }
}

```

MAIN.TS

```

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

```

Add/Remove symbols to palette at runtime

- Symbols can be added to palette at runtime by using public method, [addPaletteltem](#).
- Symbols can be removed from palette at runtime by using public method, [removePaletteltem](#).

Customize the size of symbols

The size of the individual symbol can be customized. The [symbolWidth](#) and [symbolHeight](#) properties of node enables you to define the size of the symbols. The following code example illustrates how to change the size of a symbol.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule, SymbolPaletteModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, OnInit, ViewEncapsulation, ViewChild } from
 '@angular/core';
import { SymbolPaletteComponent, SymbolPalette, NodeModel, MarginModel,
 PaletteModel } from '@syncfusion/ej2-angular-diagrams';
import { ExpandMode } from '@syncfusion/ej2-navigations';
@Component({
  imports: [
    DiagramModule, SymbolPaletteModule

```



```

    ],
    providers: [ ],
    standalone: true,
    selector: "app-container",
    template: `<ejs-symbolpalette id="symbolpalette"width="100%"
height="700px" [symbolHeight]=80 [symbolWidth]=80 [expandMode]="expandMode"
[palettes]="palettes" [symbolMargin]="symbolMargin">
    </ejs-symbolpalette>`,
    encapsulation: ViewEncapsulation.None
  ))
}
export class AppComponent {
  public expandMode?: ExpandMode;
  public palettes?: PaletteModel[];
  public symbolMargin?: MarginModel;
  public getBasicShapes(): NodeModel[] {
    let basicShapes: NodeModel[] = [{
      id: 'Rectangle',
      shape: {
        type: 'Basic',
        shape: 'Rectangle'
      }
    },
    {
      id: 'Ellipse',
      shape: {
        type: 'Basic',
        shape: 'Ellipse'
      }
    },
    {
      id: 'Hexagon',
      shape: {
        type: 'Basic',
        shape: 'Hexagon'
      }
    }
  ];
  return basicShapes;
};
ngOnInit(): void {
  this.expandMode = 'Multiple'
  this.palettes = [{
    id: 'basic',
    expanded: true,
    symbols: this.getBasicShapes(),
    title: 'Basic Shapes',
    iconCss: 'e-ddb-icons e-basic'
  }]
  //Sets the space to be left around a symbol
  this.symbolMargin = {
    left: 15,
    right: 15,
    top: 15,
    bottom: 15
  }
}
}

```

MAIN.TS

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

The [symbolMargin](#) property is used to create the space around elements, outside of any defined borders.

Symbol preview

The symbol preview size of the palette items can be customized using [symbolPreview](#).

The [width](#) and [height](#) properties of SymbolPalette enables you to define the preview size to all the symbol palette items. The [offset](#) of the dragging helper relative to the mouse cursor.

The following code example illustrates how to change the preview size of a palette item.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule, SymbolPaletteModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, OnInit, ViewEncapsulation, ViewChild } from
'@angular/core';
import { SymbolPaletteComponent, SymbolPalette, SymbolPreviewModel,
NodeModel, ConnectorModel, PaletteModel, MarginModel } from
'@syncfusion/ej2-angular-diagrams';
import { ExpandMode } from '@syncfusion/ej2-navigations';
@Component({
imports: [
DiagramModule, SymbolPaletteModule
],
providers: [ ],
standalone: true,
selector: "app-container",
template: `<ejs-symbolpalette id="symbolpalette"width="100%"
height="700px" [symbolHeight]=80 [symbolWidth]=80 [expandMode]="expandMode"
[palettes]="palettes" [getSymbolInfo]="getSymbolInfo"
[symbolMargin]="symbolMargin" [symbolPreview]="symbolPreview">
</ejs-symbolpalette>`,
encapsulation: ViewEncapsulation.None
})
export class AppComponent {
public expandMode?: ExpandMode;
public palettes?: PaletteModel[];
public symbolMargin?: MarginModel;
public symbolPreview?: SymbolPreviewModel;
public getBasicShapes(): NodeModel[] {
let basicShapes: NodeModel[] = [{
id: 'Rectangle',
shape: {
type: 'Basic',
shape: 'Rectangle'
}
```

```

        },
        {
            id: 'Ellipse',
            shape: {
                type: 'Basic',
                shape: 'Ellipse'
            }
        },
        {
            id: 'Hexagon',
            shape: {
                type: 'Basic',
                shape: 'Hexagon'
            }
        }
    ];
    return basicShapes;
};

public getSymbolInfo() {
    // Enables to fit the content into the specified palette item size
    return {
        fit: true
    };
    // When it is set as false, the element is rendered with actual node
size
};

ngOnInit(): void {
    this.expandMode = 'Multiple'
    this.palettes = [{
        id: 'basic',
        expanded: true,
        symbols: this.getBasicShapes(),
        title: 'Basic Shapes',
        iconCss: 'e-ddb-icons e-basic'
    }],
    this.symbolMargin = {
        left: 15,
        right: 15,
        top: 15,
        bottom: 15
    },
    //Specifies the preview size and position to symbol palette items.
    this.symbolPreview = {
        height: 100,
        width: 100,
        offset: {
            x: 0.5,
            y: 0.5
        }
    }
}
}
}

```

MAIN.TS

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

Default settings

While adding more number of symbols such as nodes and connectors to the palette, define the default settings for those objects through the [getNodeDefaults](#) and the [getConnectorDefaults](#) properties of diagram allows to define the default settings for nodes and connectors.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule, SymbolPaletteModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, OnInit, ViewEncapsulation, ViewChild } from
 '@angular/core';
import { SymbolPaletteComponent, SymbolPalette, SymbolPreviewModel,
NodeModel, ConnectorModel, PaletteModel, MarginModel, ShapeStyleModel } from
 '@syncfusion/ej2-angular-diagrams';
import { ExpandMode } from '@syncfusion/ej2-navigations';
@Component({
imports: [
DiagramModule, SymbolPaletteModule
],
providers: [ ],
standalone: true,
selector: "app-container",
template: `<ejs-symbolpalette id="symbolpalette"width="100%"
height="700px" [symbolHeight]=80 [symbolWidth]=80 [expandMode]="expandMode"
[palettes]="palettes" [getSymbolInfo]="getSymbolInfo"
[symbolMargin]="symbolMargin" [symbolPreview]="symbolPreview"
[getNodeDefaults]="">
</ejs-symbolpalette>`,
encapsulation: ViewEncapsulation.None
})
export class AppComponent {
public expandMode?: ExpandMode;
public palettes?: PaletteModel[];
public symbolMargin?: MarginModel;
public symbolPreview?: SymbolPreviewModel;
public getBasicShapes(): NodeModel[] {
let basicShapes: NodeModel[] = [{
id: 'Rectangle',
shape: {
type: 'Basic',
shape: 'Rectangle'
}
},
{
id: 'Ellipse',
shape: {
type: 'Basic',
shape: 'Ellipse'
}
}
```

```

        },
        {
            id: 'Hexagon',
            shape: {
                type: 'Basic',
                shape: 'Hexagon'
            }
        }
    ];
    return basicShapes;
};

public getPaletteNodeDefaults(node: NodeModel): void {
    node.width = 100;
    node.height = 100;
    ((node as NodeModel).style as ShapeStyleModel).strokeColor =
'#3A3A3A';
}

public getSymbolInfo() {
    // Enables to fit the content into the specified palette item size
    return {
        fit: true
    };
    // When it is set as false, the element is rendered with actual node
size
};

ngOnInit(): void {
    this.expandMode = 'Multiple'
    this.palettes = [{
        id: 'basic',
        expanded: true,
        symbols: this.getBasicShapes(),
        title: 'Basic Shapes',
        iconCss: 'e-ddb-icons e-basic'
    }],
    this.symbolMargin = {
        left: 15,
        right: 15,
        top: 15,
        bottom: 15
    },
    //Specifies the preview size and position to symbol palette items.
    this.symbolPreview = {
        height: 100,
        width: 100,
        offset: {
            x: 0.5,
            y: 0.5
        }
    }
}
}
}

```

MAIN.TS

```

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

```

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

Adding symbol description for symbols in the palette

The diagram provides support to add symbol description below each symbol of a palette. This descriptive representation of each symbol will enhance the details of the symbol visually. The height and width of the symbol description can also be set individually.

- The property `getSymbolInfo`, can be used to add the symbol description at runtime.

The following code is an example to set a symbol description for symbols in the palette.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule, SymbolPaletteModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation } from '@angular/core';
import { NodeModel, PaletteModel, SymbolPreviewModel, NodeConstraints }
from '@syncfusion/ej2-angular-diagrams';
import { ExpandMode } from '@syncfusion/ej2-navigations';
@Component({
  imports: [
    DiagramModule, SymbolPaletteModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  template: `<ejs-symbolpalette id="symbolpalette"width="100%"
height="100%" [expandMode]="expandMode" [enableSearch]=true
[palettes]='palettes'
[symbolHeight]=80 [symbolWidth]=80 [symbolPreview]='symbolPreview'
[getSymbolInfo]='getSymbolInfo'>
</ejs-symbolpalette>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  public expandMode?: ExpandMode;
  public palettes?: PaletteModel[];
  public symbolPreview?: SymbolPreviewModel[];
  public getSymbolInfo(symbol: any) {
    //Defines the symbol description
    return { width: 75, height: 40, description: { text:
symbol.shape['shape'] } }
  };
  public getBasicShapes(): NodeModel[] {
    let basicShapes: NodeModel[] = [{
      id: 'Rectangle',
      shape: {
        type: 'Basic',
        shape: 'Rectangle'
      }
    }
  ],
```

```

        {
            id: 'Ellipse',
            shape: {
                type: 'Basic',
                shape: 'Ellipse'
            },
            tooltip: {
                content: 'Customized Tooltip',
            },
            //Enable customized tooltip to display on the symbol
            constraints: NodeConstraints.Default |
NodeConstraints.Tooltip
        },
        {
            id: 'Hexagon',
            shape: {
                type: 'Basic',
                shape: 'Hexagon'
            }
        }
    ];
    return basicShapes;
};

public getFlowShapes(): NodeModel[] {
    let flowShapes: NodeModel[] = [{
        id: 'process',
        shape: {
            type: 'Flow',
            shape: 'Process'
        }
    },
    {
        id: 'document',
        shape: {
            type: 'Flow',
            shape: 'Document'
        }
    }
    ];
    return flowShapes;
};

ngOnInit(): void {
    this.expandMode = 'Multiple'
    this.palettes = [{
        id: 'flow',
        expanded: true,
        symbols: this.getFlowShapes(),
        title: 'Flow Shapes',
        iconCss: 'e-ddb-icons e-flow'
    },
    {
        id: 'basic',
        expanded: true,
        symbols: this.getBasicShapes(),
        title: 'Basic Shapes',
        iconCss: 'e-ddb-icons e-basic'
    }
    ]
}

```

```
    ],  
    this.symbolPreview = [{  
      height: 100,  
      width: 100  
    }]  
  }  
}
```

MAIN.TS

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

Tooltip for symbols in symbol palette

The Symbol palette supports displaying tooltips when mouse hovers over the symbols. You can customize the tooltip for each symbol in the symbol palette.

Default tooltip for symbols

When hovering over symbols in the symbol palette, the default tooltip displays the symbol's ID.

Refer to the image below for an illustration of the tooltip behavior in the symbol palette.



Custom tooltip for symbols

To customize the tooltips for symbols in the symbol palette, assign a custom tooltip to the 'Tooltip' content property of each symbol. Once you define the custom tooltip, enable the Tooltip constraints for each symbol, ensuring that the tooltips are displayed when users hover over them.

Here, the code provided below demonstrates how to define tooltip content to symbols within a symbol palette.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule, SymbolPaletteModule } from '@syncfusion/ej2-angular-diagrams'
import { Component, ViewEncapsulation } from '@angular/core';
import { NodeModel, PaletteModel, SymbolPreviewModel, NodeConstraints }
from '@syncfusion/ej2-angular-diagrams';
import { ExpandMode } from '@syncfusion/ej2-navigations';
@Component({
  imports: [
    DiagramModule, SymbolPaletteModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  template: `<ejs-symbolpalette id="symbolpalette"width="100%"
height="100%" [expandMode]="expandMode" [enableSearch]=true
[palettes]='palettes'
[symbolHeight]=80 [symbolWidth]=80 [symbolPreview]='symbolPreview'
[getSymbolInfo]='getSymbolInfo'>
</ejs-symbolpalette>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  public expandMode?: ExpandMode;
  public palettes?: PaletteModel[];
  public symbolPreview?: SymbolPreviewModel[];
  public getSymbolInfo(symbol: any) {
    //Defines the symbol description
    return { width: 75, height: 40, description: { text:
symbol.shape['shape'] } }
  };
  public getBasicShapes(): NodeModel[] {
    let basicShapes: NodeModel[] = [{
      id: 'Rectangle',
      shape: {
        type: 'Basic',
        shape: 'Rectangle'
      }
    },
    {
      id: 'Ellipse',
      shape: {
        type: 'Basic',
        shape: 'Ellipse'
      },
      tooltip:{
        content: 'Customized Tooltip',
      },
      //Enable customized tooltip to display on the symbol
      constraints: NodeConstraints.Default |
NodeConstraints.Tooltip
    },
    {
      id: 'Hexagon',

```

```

        shape: {
            type: 'Basic',
            shape: 'Hexagon'
        }
    ];
    return basicShapes;
};

public getFlowShapes(): NodeModel[] {
    let flowShapes: NodeModel[] = [{
        id: 'process',
        shape: {
            type: 'Flow',
            shape: 'Process'
        }
    },
    {
        id: 'document',
        shape: {
            type: 'Flow',
            shape: 'Document'
        }
    }
    ];
    return flowShapes;
};

ngOnInit(): void {
    this.expandMode = 'Multiple'
    this.palettes = [{
        id: 'flow',
        expanded: true,
        symbols: this.getFlowShapes(),
        title: 'Flow Shapes',
        iconCss: 'e-ddb-icons e-flow'
    },
    {
        id: 'basic',
        expanded: true,
        symbols: this.getBasicShapes(),
        title: 'Basic Shapes',
        iconCss: 'e-ddb-icons e-basic'
    }
    ],
    this.symbolPreview = [{
        height: 100,
        width: 100
    }]
}
}

```

MAIN.TS

```

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

```

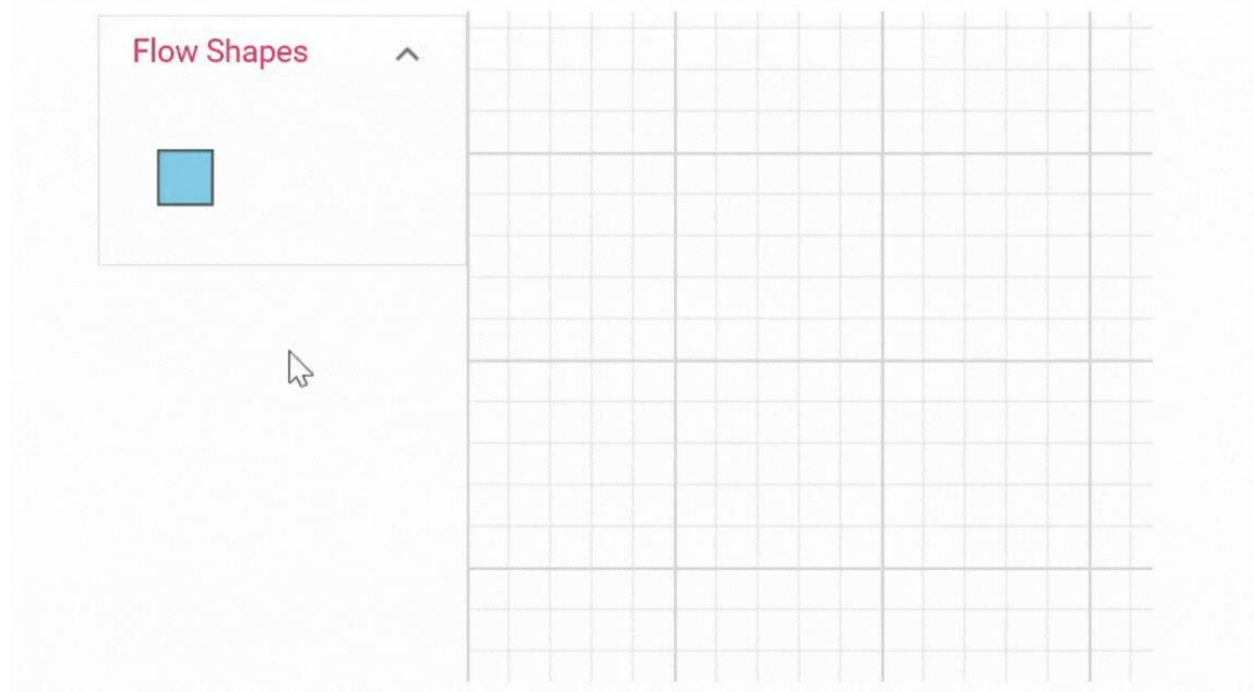
How to provide different tooltip for Symbol palette and diagram elements.

Differentiate the tooltips between symbols in the symbol palette and dropped nodes by utilizing the `dragEnter` event. When a custom tooltip is defined for a symbol, it will be displayed for both the symbol and the dropped node in the diagram canvas.

However, to provide distinct tooltips for symbols in the palette and dropped nodes, capture the `dragEnter` event and assign specific tooltips dynamically.

When a symbol is dragged from the symbol palette and enters the diagram canvas, the `[DragEnter]` [IDragEnterEventArgs](#) event is triggered. Within this event, you can define a new tooltip for the dropped node. By assigning custom tooltip content to the `Tooltip` property of the node, you can provide a distinct tooltip that is specific to the dropped node.

The following image illustrates the differentiation of tooltips displayed in the Symbol Palette and the Diagram.



The following code snippet will demonstrate how to define two different tooltip for symbol in the symbol palette and dropped node in the diagram canvas.

```
`ts
//Initialize the Diagram
let diagram: Diagram = new Diagram({
width: '100%', height: '500px',
connectors: connectors, nodes: nodes,
//event to change tooltip content while dragging symbols into Diagram
dragEnter: dragEnter,
```

```
});  
diagram.appendTo('#diagram');  
function dragEnter(args:IDragEnterEventArgs)  
{  
    //enable tooltip constraints for the dragged symbol  
    args.dragItem.constraints = NodeConstraints.Default | NodeConstraints.Tooltip;  
    //change the tooltip content of the dragged symbol  
    args.dragItem.tooltip.content='This is Diagram Tooltip';  
}  
`
```

Palette interaction

Palette interaction notifies the element enter, leave, and dragging of the symbols into the diagram.

DragEnter

[DragEnter] [IDragEnterEventArgs](#) notifies, when the element enter into the diagram from symbol palette.

DragLeave

[DragLeave] [IDragLeaveEventArgs](#) notifies, when the element leaves from the diagram.

DragOver

[DragOver] [IDragOverEventArgs](#) notifies, when an element drag over another diagram element.

Note: The diagram provides support to cancel the drag and drop operation from the symbol palette to the diagram when the ESC key is pressed.

See Also

- [How to add the symbol to the diagram](#)

Overview in Angular Diagram component

Overview control allows you to see a preview or an overall view of the entire content of a Diagram. This helps you to look at the overall picture of a large Diagram and also to navigate, pan, or zoom, on a particular position of the page.

When you work on a very large Diagram, you may not know the part you are actually working on, or navigation from one part to another might be difficult. One solution for navigation is to zoom out the entire Diagram and find where you are. Then, you can zoom in a particular area you want to. This solution is not suitable when you need some frequent navigation.

Overview control solves these problems by showing you a preview, that is, an overall view of the entire Diagram. A rectangle indicates viewport of the Diagram. Navigation becomes easy by dragging this rectangle.

[Create overview](#)

The `sourceID` property of overview should be set with the corresponding Diagram ID for you need the overall view.

The `width` and `height` property of the overview allows you to define the size of the overview.

The following code illustrates how to create overview.

[Zoom and Pan](#)

In overview, the view port of the Diagram is highlighted with a red colored rectangle. Diagram can be zoomed/panned by interacting with that. You can interact with overview as follows.

- Resize the rectangle - Zooms in/out the Diagram
- Drag the rectangle - Pans the Diagram
- Click at a position - Navigates to the clicked region
- Choose a particular region by clicking and dragging - Navigates to the specified region

The following image shows how the Diagram is zoomed/panned with overview.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DiagramModule, OverviewModule, DataBindingService,
HierarchicalTreeService } from '@syncfusion/ej2-angular-diagrams'
import { Component, OnInit, ViewEncapsulation, ViewChild } from
'@angular/core';
import { DiagramComponent, OverviewComponent, Diagram, NodeModel,
ConnectorModel, OverviewModel, SnapSettingsModel, LayoutModel,
DataSourceModel, TextModel, DecoratorModel, ShapeStyleModel, } from
'@syncfusion/ej2-angular-diagrams';
import { DataManager, Query } from '@syncfusion/ej2-data';
@Component({
  imports: [
    DiagramModule, OverviewModule
  ],
  providers: [DataBindingService, HierarchicalTreeService],
  standalone: true,
  selector: "app-container",
  template: `<div><ejs-diagram #diagram id="diagram" width="100%"
height="580px" [getNodeDefaults]="getNodeDefaults"
[getConnectorDefaults]="getConnectorDefaults" [snapSettings]="snapSettings"
[layout]="layout" [dataSourceSettings]="dataSourceSettings">
</ejs-diagram></div>
<div style=" width:50%; height: 200px
padding:0px;right:5px;bottom:5px;background:#f7f7f7;position:absolute">
<ejs-overview id="overview" width="100%" sourceID="diagram">
</ejs-overview>
</div>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild("diagram")
  public diagram?: DiagramComponent;
  public snapSettings?: SnapSettingsModel;
  public items?: DataManager;
```

```
public layout?: LayoutModel;
public dataSourceSettings?: DataSourceModel;
//Initializes data source
public data: object[] = [{
    Id: "parent",
    Role: "Project Management"
},
{
    Id: 1,
    Role: "R&D Team",
    Team: "parent"
},
{
    Id: 3,
    Role: "Philosophy",
    Team: "1"
},
{
    Id: 4,
    Role: "Organization",
    Team: "1"
},
{
    Id: 5,
    Role: "Technology",
    Team: "1"
},
{
    Id: 7,
    Role: "Funding",
    Team: "1"
},
{
    Id: 8,
    Role: "Resource Allocation",
    Team: "1"
},
{
    Id: 9,
    Role: "Targeting",
    Team: "1"
},
{
    Id: 11,
    Role: "Evaluation",
    Team: "1"
},
{
    Id: 156,
    Role: "HR Team",
    Team: "parent"
},
{
    Id: 13,
    Role: "Recruitment",
    Team: "156"
},
},
```

```
{
  Id: 113,
  Role: "Training",
  Team: "12"
},
{
  Id: 112,
  Role: "Employee Relation",
  Team: "156"
},
{
  Id: 14,
  Role: "Record Keeping",
  Team: "12"
},
{
  Id: 15,
  Role: "Compensations & Benefits",
  Team: "12"
},
{
  Id: 16,
  Role: "Compliances",
  Team: "12"
},
{
  Id: 17,
  Role: "Production & Sales Team",
  Team: "parent"
},
{
  Id: 119,
  Role: "Design",
  Team: "17"
},
{
  Id: 19,
  Role: "Operation",
  Team: "17"
},
{
  Id: 20,
  Role: "Support",
  Team: "17"
},
{
  Id: 21,
  Role: "Quality Assurance",
  Team: "17"
},
{
  Id: 23,
  Role: "Customer Interaction",
  Team: "17"
},
{
  Id: 24,
```

```

        Role: "Support and Maintenance",
        Team: "17"
    },
    {
        Id: 25,
        Role: "Task Coordination",
        Team: "17"
    }
];
//Sets the default properties for all the Nodes
public getNodeDefaults(obj: NodeModel, diagram: Diagram): NodeModel {
    obj.shape = {
        type: 'Text',
        content: (obj.data as {
            Role: 'string'
        }).Role
    };
    obj.style = {
        fill: 'None',
        strokeColor: 'none',
        strokeWidth: 2,
        bold: true,
        color: 'white'
    };
    obj.borderColor = 'white';
    obj.backgroundColor = '#6BA5D7';
    obj.borderWidth = 1;
    obj.width = 75;
    obj.height = 40;
    (obj.shape as TextModel).margin = {
        left: 5,
        right: 5,
        top: 5,
        bottom: 5
    };
    return obj;
}
//Sets the default properties for all the connectors
public getConnectorDefaults(connector: ConnectorModel, diagram:
Diagram): ConnectorModel {
    connector.style = {
        strokeColor: '#6BA5D7',
        strokeWidth: 2
    };
    (((connector as ConnectorModel).targetDecorator as
DecoratorModel).style as ShapeStyleModel).fill = '#6BA5D7';
    (((connector as ConnectorModel).targetDecorator as
DecoratorModel).style as ShapeStyleModel).strokeColor = '#6BA5D7';
    connector.type = 'Orthogonal';
    return connector;
}
ngOnInit(): void {
    this.snapSettings = {
        constraints: 0
    }
    this.items = new DataManager(this.data as JSON[], new
Query().take(5));

```



```

//Uses layout to auto-arrange nodes on the Diagram page
this.layout = {
  //set the type as Organizational Chart
  type: 'OrganizationalChart'
}
//Configures data source for Diagram
this.dataSourceSettings = {
  id: 'Id',
  parentId: 'Team',
  dataSource: this.items
}
}
}

```

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 Diagram component

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

Background

<!-- markdownlint-disable MD033 -->

Behavior	API in Essential JS 1	API in Essential JS 2
Defines the background color of diagram elements	Property: <code>backgroundColorHTML</code>	Property: <code>backgroundColorHTML</code>
Defines how to align the background image over the diagram area	Property: <code>backgroundImage.alignmentHTML</code> Script <pre> public backgroundImage; constructor() { this.backgroundImage = { alignment: "XMidYMid", } }; </pre>	Property: <code>background.alignHTML</code> Script <pre> public pageSettings: PageSettingsModel; ngOnInit(): void { this.pageSettings = { background: { align: 'XMinYMin' }, } } </pre>
Defines how the background image should be	Property: <code>backgroundImage.scaleHTML</code> Script <pre> public backgroundImage; constructor() { this.backgroundImage = { scale: "Meet", }; } </pre>	Property: <code>background.scaleHTML</code> Script <pre> public pageSettings: PageSettingsModel; ngOnInit(): void { this.pageSettings = { background: { scale: 'Meet', }, } } </pre>

scaled/str etched		
Sets the source path of the backgroun d image	Property: backgroundImage.sourceHTML Script public backgroundImage; constructor() { this.backgroundImage = { source: "https://www.w3schools.com/images/ w3schools_green.jpg" }; }	Property: background.sourceHTML Script public pageSettings: PageSettingsModel; ngOnInit(): void { this.pageSettings = { background: { source: "https://www.w3schools.com/images/ w3schools_green.jpg", }, } }

Bridging

<!-- markdownlint-disable MD033 -->

Behavior	API in Essential JS 1	API in Essential JS 2
Sets the direction of line bridges	Property: bridgeDirectionHTML	Property: bridgeDirectionHTML

CommandManager

<!-- markdownlint-disable MD033 -->

Behavior	API in Essential JS 1	API in Essential JS 2
comm ands	Property: commandManager.commandsHT ML Script commandManager: Object; constructor() { this.commandManager = { commands: { "clone": { gesture: { key: Keys.C, keyModifiers: KeyModifiers.Shift }, canExecute: function(args) { let diagram = \$("#diagramContent").ejDiagram("ins tance"); return diagram.model.selectedItems.childre n.length; }, execute: function(args) { let diagram = \$("#diagramContent").ejDiagram("ins tance"); diagram.copy(); diagram.paste(); } } } } });	Property: commandManager.commandsHT ML Script public commandManager: CommandManager; ngOnInit(): void { this.commandManager = { commands: [{ name: 'customCopy', parameter: 'node', canExecute: function() { if (diagram.selectedItems.nodes.length > 0 diagram.selectedItems.connectors.le ngth > 0) { return true; } return false; }, execute: function() { for (let i: number = 0; i < diagram.selectedItems.nodes.length; i++) { diagram.selectedItems.nodes[i].styl e.fill = 'red'; } diagram.dataBind(); }, gesture: { key: Keys.G, keyModifiers: KeyModifiers.Shift } }]} }
The comm and is executa ble at the	Property: commandManager.commands.can ExecuteHTML Script commandManager: Object; constructor() { this.commandManager = { commands: { "clone": { gesture: { key: Keys.C, keyModifiers: KeyModifiers.Shift }, canExecute: function(args) { let diagram =	Property: commandManager.commands.can ExecuteHTML Script public commandManager: CommandManager; ngOnInit(): void { this.commandManager = { commands: [{ name: 'customCopy', parameter: 'node', canExecute: function() { if (diagram.selectedItems.nodes.length

moment or not.	<pre>\$("#diagramContent").ejDiagram("instance"); return diagram.model.selectedItems.children.length; }, execute: function(args) { let diagram = \$("#diagramContent").ejDiagram("instance"); diagram.copy(); diagram.paste(); } } } } }));</pre>	<pre>> 0 diagram.selectedItems.connectors.length > 0) { return true; } return false; }, execute: function() { for (let i: number = 0; i < diagram.selectedItems.nodes.length; i++) { diagram.selectedItems.nodes[i].style.fill = 'red'; } diagram.dataBind(); }, gesture: { key: Keys.G, keyModifiers: KeyModifiers.Shift } }] } }</pre>
Defines what to be executed when the key combination is recognized	<pre>Property:commandManager.commands.executeHTML Script commandManager: Object; constructor() { this.commandManager = { commands: { "clone": { gesture: { key: Keys.C, keyModifiers: KeyModifiers.Shift }, canExecute: function(args) { let diagram = \$("#diagramContent").ejDiagram("instance"); return diagram.model.selectedItems.children.length; }, execute: function(args) { let diagram = \$("#diagramContent").ejDiagram("instance"); diagram.copy(); diagram.paste(); } } } } } }));</pre>	<pre>Property:commandManager.commands.executeHTML Script public commandManager: CommandManager; ngOnInit(): void { this.commandManager = { commands: [{ name: 'customCopy', parameter: 'node', canExecute: function() { if (diagram.selectedItems.nodes.length > 0 diagram.selectedItems.connectors.length > 0) { return true; } return false; }, execute: function() { for (let i: number = 0; i < diagram.selectedItems.nodes.length; i++) { diagram.selectedItems.nodes[i].style.fill = 'red'; } diagram.dataBind(); }, gesture: { key: Keys.G, keyModifiers: KeyModifiers.Shift } }] } }</pre>
Defines a combination of keys and key modifiers, on recognition of which the command will be executed	<pre>Property:commandManager.commands.gestureHTML Script commandManager: Object; constructor() { this.commandManager = { commands: { "clone": { gesture: { key: Keys.C, keyModifiers: KeyModifiers.Shift }, canExecute: function(args) { let diagram = \$("#diagramContent").ejDiagram("instance"); return diagram.model.selectedItems.children.length; }, execute: function(args) { let diagram = \$("#diagramContent").ejDiagram("instance"); diagram.copy(); diagram.paste(); } } } } } }));</pre>	<pre>Property:commandManager.commands.gestureHTML Script public commandManager: CommandManager; ngOnInit(): void { this.commandManager = { commands: [{ name: 'customCopy', parameter: 'node', canExecute: function() { if (diagram.selectedItems.nodes.length > 0 diagram.selectedItems.connectors.length > 0) { return true; } return false; }, execute: function() { for (let i: number = 0; i < diagram.selectedItems.nodes.length; i++) { diagram.selectedItems.nodes[i].style.fill = 'red'; } diagram.dataBind(); }, gesture: { key: Keys.G, keyModifiers: KeyModifiers.Shift } }] } }</pre>

Sets the key value, on recognition of which the command will be executed	<pre>Property:commandManager.commands.gesture.keyHTML Script commandManager: Object; constructor() { this.commandManager = { commands: { "clone": { gesture: { key: Keys.C, keyModifiers: KeyModifiers.Shift }, canExecute: function(args) { let diagram = \$("#diagramContent").ejDiagram("instance"); return diagram.model.selectedItems.children.length; }, execute: function(args) { let diagram = \$("#diagramContent").ejDiagram("instance"); diagram.copy(); diagram.paste(); } } } } });</pre>	<pre>Property:commandManager.commands.gesture.keyHTML Script commandManager: Object; constructor() { this.commandManager = { commands: [{ name: 'customCopy', parameter: 'node', canExecute: function() { if (diagram.selectedItems.nodes.length > 0 diagram.selectedItems.connectors.length > 0) { return true; } return false; }, execute: function() { for (let i: number = 0; i < diagram.selectedItems.nodes.length; i++) { diagram.selectedItems.nodes[i].style.fill = 'red'; } diagram.dataBind(); }, gesture: { key: Keys.G, keyModifiers: KeyModifiers.Shift } }]} }</pre>
Sets a combination of key modifiers, on recognition of which the command will be executed.	<pre>Property:commandManager.commands.gesture.keyModifiersHTML Script commandManager: Object; constructor() { this.commandManager = { commands: { "clone": { gesture: { key: Keys.C, keyModifiers: KeyModifiers.Shift }, canExecute: function(args) { let diagram = \$("#diagramContent").ejDiagram("instance"); return diagram.model.selectedItems.children.length; }, execute: function(args) { let diagram = \$("#diagramContent").ejDiagram("instance"); diagram.copy(); diagram.paste(); } } } } });</pre>	<pre>Property:commandManager.commands.gesture.keyModifiersHTML Script public commandManager: CommandManager; ngOnInit(): void { this.commandManager = { commands: [{ name: 'customCopy', parameter: 'node', canExecute: function() { if (diagram.selectedItems.nodes.length > 0 diagram.selectedItems.connectors.length > 0) { return true; } return false; }, execute: function() { for (let i: number = 0; i < diagram.selectedItems.nodes.length; i++) { diagram.selectedItems.nodes[i].style.fill = 'red'; } diagram.dataBind(); }, gesture: { key: Keys.G, keyModifiers: KeyModifiers.Shift } }]} }</pre>
Defines any additional parameters that are required at runtime	<pre>Property:commandManager.commands.parameterHTML Script commandManager: Object; constructor() { this.commandManager = { commands: { "clone": { parameter : "node", gesture: { key: Keys.C, keyModifiers: KeyModifiers.Shift }, canExecute: function(args) { let diagram = \$("#diagramContent").ejDiagram("instance"); return diagram.model.selectedItems.children.length; }, execute: function(args) { let diagram =</pre>	<pre>Property:commandManager.commands.parameterHTML Script public commandManager: CommandManager; ngOnInit(): void { this.commandManager = { commands: [{ name: 'customCopy', parameter: 'node', canExecute: function() { if (diagram.selectedItems.nodes.length > 0 diagram.selectedItems.connectors.length > 0) { return true; } return false; }, execute: function() { for (let i: number = 0; i < diagram.selectedItems.nodes.length;</pre>

	<pre>\$("#diagramContent").ejDiagram("instance"); diagram.copy(); diagram.paste(); } } } } });</pre>	<pre>i++) { diagram.selectedItems.nodes[i].style.fill = 'red'; } diagram.dataBind(); }, gesture: { key: Keys.G, keyModifiers: KeyModifiers.Shift } }] } }</pre>
--	------------------------------------------------------------------------------------------------------	------------------------------------------------------------------------------------------------------------------------------------------------------------------

Connectors

<!-- markdownlint-disable MD033 -->

Behavior	API in Essential JS 1	API in Essential JS 2
addInfo	Property:connectors.addInfoHTML Script <pre>public addInfo; constructor() { var addInfo = { Description: "Bidirectional Flow" }; }</pre>	Property:connectors.addInfoHTML Script <pre>public addInfo: object[]; ngOnInit(): void { this.addInfo = { Description: "Bidirectional Flow" }; }</pre>
Defines the bridgeSpace of connector	Property:connectors.bridgeSpaceHTML Script <pre>public connectors; constructor() { var connector = { name: "connector", sourcePoint: { x: 100, y: 100 }, targetPoint: { x: 200, y: 200 }, bridgeSpace: 15, }; this.connectors = [connector]; }</pre>	Property:connectors.bridgeSpaceHTML Script <pre>public connectors: ConnectorModel[]; ngOnInit(): void { let connectors: ConnectorModel[] = [{ id: 'connector', type: 'Straight', sourcePoint: { x: 500, y: 100 }, bridgeSpace: 15, targetPoint: { x: 600, y: 200 } }]; }</pre>
Enables or disables the behaviors of connectors	Property:connectors.constraintsHTML Script <pre>public connectors; constructor() { var ConnectorConstraints = ConnectorConstraints; var connector = { name: "connector", sourcePoint: { x: 100, y: 100 }, targetPoint: { x: 200, y: 200 }, constraints: ConnectorConstraints.Default & ~ConnectorConstraints.Select }; this.connectors = [connector]; }</pre>	Property:connectors.constraintsHTML Script <pre>public connectors: ConnectorModel[]; ngOnInit(): void { let connectors: ConnectorModel[] = [{ id: 'connector', type: 'Straight', sourcePoint: { x: 500, y: 100 }, targetPoint: { x: 600, y: 200 }, constraints: ConnectorConstraints.Default ConnectorConstraints.Drag }]; }</pre>
Defines the radius of the rounded corner	Property:connectors.cornerRadiusHTML Script <pre>public connectors; constructor() { var connector = { name: "connector", sourcePoint: { x: 100, y: 100 }, targetPoint: { x: 200, y: 200 }, cornerRadius: 10, segments:[{ type: "orthogonal"}] }; this.connectors = [connector]; }</pre>	Property:connectors.cornerRadiusHTML Script <pre>public connectors: ConnectorModel[]; ngOnInit(): void { let connectors: ConnectorModel[] = [{ id: 'connector', type: 'Straight', sourcePoint: { x: 500, y: 100 }, targetPoint: { x: 600, y: 200 }, cornerRadius: 10, type: 'Orthogonal', }]; }</pre>
cssClass	Property:connectors.cssClassHTML Script <pre>public connectors; constructor() { var connector = { name:</pre>	Not applicable

	<code>"connector", sourcePoint: { x: 100, y: 100 }, targetPoint: { x: 200, y: 200 }, cssClass: "hoverConnector" };</code> <code>this.connectors = [connector]; }</code>	
Alignment	Property: <code>connectors.horizontalAlign</code> HTML L Script <code>public connectors;</code> <code>constructor() { var connector = { name: "connector", sourcePoint: { x: 100, y: 100 }, targetPoint: { x: 200, y: 200 }, horizontalAlign: HorizontalAlignment.Right }; this.connectors = [connector]; }</code>	Not applicable
A collection of JSON objects where each object represents a label	Property: <code>connectors.labels</code> HTML Script <code>public connectors;</code> <code>constructor() { var connector = { name: "connector", sourcePoint: { x: 100, y: 100 }, targetPoint: { x: 200, y: 200 }, labels: [{ text: "connector" }] }; this.connectors = [connector]; }</code>	Property: <code>connectors.annotations</code> HTML Script <code>public connectors;</code> <code>ConnectorModel[]; ngOnInit(): void { let connectors: ConnectorModel[] = [{ id: 'connector', type: 'Straight', sourcePoint: { x: 500, y: 100 }, targetPoint: { x: 600, y: 200 }, annotations: [{ id: 'label', content: 'Text', offset: 0.5 }] }]; }</code>
stroke color of the connector	Property: <code>connectors.lineColor</code> HTML Script <code>public connectors;</code> <code>constructor() { let connector = { name: "connector", sourcePoint: { x: 100, y: 100 }, targetPoint: { x: 200, y: 200 }, lineColor: "blue" }; this.connectors = [connector]; }</code>	Property: <code>connectors.style.strokeColor</code> HTML Script <code>public connectors;</code> <code>ConnectorModel[]; ngOnInit(): void { let connectors: ConnectorModel[] = [{ id: 'connector', type: 'Straight', sourcePoint: { x: 500, y: 100 }, targetPoint: { x: 600, y: 200 }, style: { strokeColor: 'blue' }, }]; }</code>
Sets the pattern of dashes and gaps used to stroke the path of the connector	Property: <code>connectors.lineDashArray</code> HTML Script <code>public connectors;</code> <code>constructor() { let connector = { name: "connector", sourcePoint: { x: 100, y: 100 }, targetPoint: { x: 200, y: 200 }, lineColor: "blue", lineDashArray: "2,2" }; this.connectors = [connector]; }</code>	Property: <code>connectors.style.strokeDashArray</code> HTML Script <code>public connectors;</code> <code>ConnectorModel[]; ngOnInit(): void { let connectors: ConnectorModel[] = [{ id: 'connector', type: 'Straight', sourcePoint: { x: 500, y: 100 }, targetPoint: { x: 600, y: 200 }, style: { strokeColor: 'blue', strokeWidth: 3, strokeDashArray: '2,2' }, }]; }</code>
Sets the width of the line	Property: <code>connectors.lineWidth</code> HTML Script <code>public connectors;</code> <code>constructor() { let connector = { name: "connector", sourcePoint: { x: 100, y: 100 }, targetPoint: { x: 200, y: 200 },</code>	Property: <code>connectors.style.strokeWidth</code> HTML L Script <code>public connectors;</code> <code>ConnectorModel[]; ngOnInit(): void { let connectors: ConnectorModel[] = [{ id: 'connector', type: 'Straight', sourcePoint: { x: 500, y: 100 }, targetPoint: { x: 600, y:</code>

	<code>lineWidth: 10 }; this.connectors = [connector]; }</code>	<code>200 }, style: { strokeColor: 'blue', strokeWidth: 3, strokeDashArray: '2,2' }, }); }</code>
Defines the padding value to ease the interaction with connectors	Property: <code>connectors.lineHitPaddingHTML</code> Script <code>public connectors; constructor() { let connector = { name: "connector", sourcePoint: { x: 100, y: 100 }, targetPoint: { x: 200, y: 200 }, lineHitPadding: 15 }; this.connectors = [connector]; }</code>	Property: <code>connectors.hitPaddingHTML</code> Script <code>public connectors; ConnectorModel[]; ngOnInit(): void { let connectors: ConnectorModel[] = [{ id: 'connector', type: 'Straight', sourcePoint: { x: 500, y: 100 }, targetPoint: { x: 600, y: 200 }, hitPadding: 10 }]; }</code>
Defines the minimum space to be left between the bottom of parent bounds and the connector	Property: <code>connectors.marginBottomHTML</code> Script <code>public connectors; constructor() { let connector = { name: "connector", sourcePoint: { x: 100, y: 100 }, targetPoint: { x: 200, y: 200 }, marginBottom: 15 }; this.connectors = [connector]; }</code>	Property: <code>connectors.margin.bottomHTML</code> Script <code>public connectors; ConnectorModel[]; ngOnInit(): void { let connectors: ConnectorModel[] = [{ id: 'connector', type: 'Straight', sourcePoint: { x: 500, y: 100 }, targetPoint: { x: 600, y: 200 }, margin: { bottom: 3 } }]; }</code>
Defines the minimum space to be left between the top of parent bounds and the connector	Property: <code>connectors.marginTopHTML</code> Script <code>public connectors; constructor() { let connector = { name: "connector", sourcePoint: { x: 100, y: 100 }, targetPoint: { x: 200, y: 200 }, marginTop: 15 }; this.connectors = [connector]; }</code>	Property: <code>connectors.margin.topHTML</code> Script <code>public connectors; ConnectorModel[]; ngOnInit(): void { let connectors: ConnectorModel[] = [{ id: 'connector', type: 'Straight', sourcePoint: { x: 500, y: 100 }, targetPoint: { x: 600, y: 200 }, margin: { top: 3 } }]; }</code>
Defines the minimum space to be left between the left of parent bounds and the connector	Property: <code>connectors.marginLeftHTML</code> Script <code>public connectors; constructor() { let connector = { name: "connector", sourcePoint: { x: 100, y: 100 }, targetPoint: { x: 200, y: 200 }, marginLeft: 15 }; this.connectors = [connector]; }</code>	Property: <code>connectors.margin.leftHTML</code> Script <code>public connectors; ConnectorModel[]; ngOnInit(): void { let connectors: ConnectorModel[] = [{ id: 'connector', type: 'Straight', sourcePoint: { x: 500, y: 100 }, targetPoint: { x: 600, y: 200 }, margin: { left: 3 } }]; }</code>
Defines the minimum space to be left between	Property: <code>connectors.marginRightHTML</code> Script <code>public connectors; constructor() { let connector = { name: "connector", sourcePoint: { x: 100, y: 100 },</code>	Property: <code>connectors.margin.rightHTML</code> Script <code>public connectors; ConnectorModel[]; ngOnInit(): void { let connectors: ConnectorModel[] = [{ id: 'connector', type:</code>

the right of parent bounds and the connector	<pre>targetPoint: { x: 200, y: 200 }, marginRight: 15 }; this.connectors = [connector]; }</pre>	<pre>'Straight', sourcePoint: { x: 500, y: 100 }, targetPoint: { x: 600, y: 200 }, margin: { right: 3 } }]; }</pre>
Sets a unique name for the connector	Property:connectors.nameHTML Script <pre>public connectors; constructor() { let connector = { name: "connector", sourcePoint: { x: 100, y: 100 }, targetPoint: { x: 200, y: 200 }, }; this.connectors = [connector]; }</pre>	Property:connectors.idHTML Script <pre>public connectors: ConnectorModel[]; ngOnInit(): void { let connectors: ConnectorModel[] = [{ id: 'connector', type: 'Straight', sourcePoint: { x: 500, y: 100 }, targetPoint: { x: 600, y: 200 } }]; }</pre>
Defines the transparency of the connector	Property:connectors.opacityHTML Script <pre>public connectors; constructor() { let connector = { name: "connector", sourcePoint: { x: 100, y: 100 }, targetPoint: { x: 200, y: 200 }, opacity: 0.5 }; this.connectors = [connector]; }</pre>	Property:connectors.style.opacityHTML Script <pre>public connectors: ConnectorModel[]; ngOnInit(): void { let connectors: ConnectorModel[] = [{ id: 'connector', type: 'Straight', sourcePoint: { x: 500, y: 100 }, targetPoint: { x: 600, y: 200 }, style: { strokeColor: 'blue', strokeWidth: 3, strokeDashArray: '2,2', opacity: 0.5 }, }]; }</pre>
Sets the parent name of the connector.	Property:connectors.parentHTML Script <pre>public connectors; constructor() { let connector = { name: "connector", sourcePoint: { x: 100, y: 100 }, targetPoint: { x: 200, y: 200 }, parent:"group" }; let group = { name : "group", children:["connector"] }; this.connectors = [connector]; }</pre>	Not applicable
An array of JSON objects where each object represents a segment	Property:connectors.segmentsHTML Script <pre>public connectors; constructor() { let connector = { name: "connector", sourcePoint: { x: 100, y: 100 }, targetPoint: { x: 200, y: 200 }, segments: [{type:"straight", point: { x:75, y:150 } }]; this.connectors = [connector]; }</pre>	Property:connectors.segmentsHTML Script <pre>public connectors: ConnectorModel[]; ngOnInit(): void { let connectors: ConnectorModel[] = [{ id: 'connector', type: 'Straight', sourcePoint: { x: 500, y: 100 }, targetPoint: { x: 600, y: 200 }, segments: [{ type: 'Orthogonal', length: 30, direction: 'Bottom' }, { type: 'Orthogonal', length: 80, direction: 'Right' }] }]; }</pre>
Sets the direction of orthogonal segment	Property:connectors.segments.directionHTML Script <pre>public connectors; constructor() { let connector = { name: "connector", sourcePoint: { x: 100, y: 100 },</pre>	Property:connectors.segments.directionHTML Script <pre>public connectors: ConnectorModel[]; ngOnInit(): void { let connectors: ConnectorModel[] = [{ id: 'connector', type:</pre>

	<pre>targetPoint: { x: 200, y: 200 }, segments: [{type:"straight", point: { x:75, y:150 }, direction:"bottom"}] }; this.connectors = [connector]; }</pre>	<pre>'Straight', sourcePoint: { x: 500, y: 100 }, targetPoint: { x: 600, y: 200 }, segments: [{ type: 'Orthogonal', length: 30, direction: 'Bottom' }, { type: 'Orthogonal', length: 80, direction: 'Right' }] }]; }</pre>
Describes the length of orthogonal segment	Property:connectors.segments.lengthHTML Script public connectors; constructor() { let connector = { name: "connector", sourcePoint: { x: 100, y: 100 }, targetPoint: { x: 200, y: 200 }, segments: [{type:"straight", point: { x:75, y:150 }, length:50 }] }; this.connectors = [connector]; }	Property:connectors.segments.lengthHTML Script public connectors; ConnectorModel[]; ngOnInit(): void { let connectors: ConnectorModel[] = [{ id: 'connector', type: 'Straight', sourcePoint: { x: 500, y: 100 }, targetPoint: { x: 600, y: 200 }, segments: [{ type: 'Orthogonal', length: 30, direction: 'Bottom' }, { type: 'Orthogonal', length: 80, direction: 'Right' }] }]; }
Describes the end point of bezier/straight segment	Property:connectors.segments.pointHTML Script public connectors; constructor() { let connector = { name: "connector", sourcePoint: { x: 100, y: 100 }, targetPoint: { x: 200, y: 200 }, segments: [{type:"straight", point: { x:75, y:150 } }] }; this.connectors = [connector]; }	Property:connectors.segments.pointHTML Script public connectors; ConnectorModel[]; ngOnInit(): void { let connectors: ConnectorModel[] = [{ id: 'connector', type: 'Straight', sourcePoint: { x: 500, y: 100 }, targetPoint: { x: 600, y: 200 }, segments: [{ type: 'Straight', point: { x: 800, y: 50 } }] }]; }
Defines the first control point of the bezier segment	Property:connectors.segments.point1HTML Script public connectors; constructor() { let connector = { name: "connector", sourcePoint: { x: 100, y: 100 }, targetPoint: { x: 200, y: 200 }, segments: [{ type:"bezier", point1: { x:150, y:50} }] }; this.connectors = [connector]; }	Property:connectors.segments.point1HTML Script public connectors; ConnectorModel[]; ngOnInit(): void { let connectors: ConnectorModel[] = [{ id: 'connector', type: 'Straight', sourcePoint: { x: 500, y: 100 }, targetPoint: { x: 600, y: 200 }, segments: [{ type: 'Bezier', point: { x: 600, y: 300 }, point1: { x: 525, y: 475 } }] }]; }
Defines the second control point of bezier segment	Property:connectors.segments.point2HTML Script public connectors; constructor() { let connector = { name: "connector", sourcePoint: { x: 100, y: 100 }, targetPoint: { x: 200, y: 200 }, segments: [{ type:"bezier", point1: { x:150, y:50}, point2:{ x: 150, y: 150 } }] }; this.connectors = [connector]; }	Property:connectors.segments.point2HTML Script public connectors; ConnectorModel[]; ngOnInit(): void { let connectors: ConnectorModel[] = [{ id: 'connector', type: 'Straight', sourcePoint: { x: 500, y: 100 }, targetPoint: { x: 600, y: 200 }, segments: [{ type: 'Bezier', point: { x: 600, y: 300 }, point1: { x: 525, y: 475 }, point2: { x: 575, y: 475 } }] }]; }

Sets the type of the segment	Property:connectors.segments.typeHTML L Script public connectors; constructor() { let connector = { name: "connector", sourcePoint: { x: 100, y: 100 }, targetPoint: { x: 200, y: 200 }, segments: [{ type: Segments.Bezier }] }; this.connectors = [connector]; }	Property:connectors.segments.typeHTML Script public connectors: ConnectorModel[]; ngOnInit(): void { let connectors: ConnectorModel[] = [{ id: 'connector', type: 'Straight', sourcePoint: { x: 500, y: 100 }, targetPoint: { x: 600, y: 200 }, segments: [{ type: 'Bezier', point: { x: 600, y: 300 }, point1: { x: 525, y: 475 }, point2: { x: 575, y: 475 } }] }]; }
Describes the length and angle between the first control point and the start point of bezier segment	Property:connectors.segments.vector1HTML Script public connectors; constructor() { let connector = { name: "connector", sourcePoint: { x: 100, y: 100 }, targetPoint: { x: 200, y: 200 }, segments: [{ type: "bezier", vector1: { distance:75, angle: 0}, vector2: { distance:75, angle: 180} }] }; this.connectors = [connector]; }	Property:connectors.segments.vector1HTML L Script public connectors: ConnectorModel[]; ngOnInit(): void { let connectors: ConnectorModel[] = [{ id: 'connector', type: 'Straight', sourcePoint: { x: 500, y: 100 }, targetPoint: { x: 600, y: 200 }, segments: [{ type: 'Bezier', point: { x: 900, y: 160 }, vector1: { angle: 20, distance: 75 }, vector2: { angle: 20, distance: 75 } }], }]; }
Describes the length and angle between the second control point and end point of bezier segment	Property:connectors.segments.vector2HTML Script public connectors; constructor() { let connector = { name: "connector", sourcePoint: { x: 100, y: 100 }, targetPoint: { x: 200, y: 200 }, segments: [{ type:"bezier", vector1: { distance:75, angle: 0}, vector2: { distance:75, angle: 180} }] }; this.connectors = [connector]; }	Property:connectors.segments.vector2HTML L Script public connectors: ConnectorModel[]; ngOnInit(): void { let connectors: ConnectorModel[] = [{ id: 'connector', type: 'Straight', sourcePoint: { x: 500, y: 100 }, targetPoint: { x: 600, y: 200 }, segments: [{ type: 'Bezier', point: { x: 900, y: 160 }, vector1: { angle: 20, distance: 75 }, vector2: { angle: 20, distance: 75 } }] }]; }
Sets the type of the connector	Property:connectors.shape.typeHTML Script public connectors; constructor() { let connector = { name: "connector", sourcePoint: { x: 100, y: 100 }, targetPoint: { x: 200, y: 200 }, shape: { type: "bpmn"}, segments: [{ type:"straight" }] }; this.connectors = [connector]; }	Property:connectors.shape.typeHTML Script public connectors: ConnectorModel[]; ngOnInit(): void { let connectors: ConnectorModel[] = [{ id: 'connector', type: 'Straight', sourcePoint: { x: 500, y: 100 }, targetPoint: { x: 600, y: 200 }, segments: [{ type: 'Bezier', point: { x: 600, y: 300 }, point1: { x: 525, y: 475 }, point2: { x: 575, y: 475 } }], shape: { type: 'Bpmn', flow: 'Message', message: 'InitiatingMessage' } }]; }
Defines the source decorator	Property:connectors.sourceDecoratorHTML Script public connectors; constructor() { let connector =	Property:connectors.sourceDecoratorHTML Script public connectors: ConnectorModel[]; ngOnInit(): void

of the connector	<pre>{ name: "connector", sourcePoint: { x: 100, y: 100 }, targetPoint: { x: 200, y: 200 }, sourceDecorator: { shape:"openarrow" } }; this.connectors = [connector]; }</pre>	<pre>{ let connectors: ConnectorModel[] = [{ id: 'connector', type: 'Straight', sourcePoint: { x: 500, y: 100 }, targetPoint: { x: 600, y: 200 }, sourceDecorator: { shape: 'Arrow', }, }]; }</pre>
Sets the border color of the source decorator	Property:connectors.sourceDecorator.borderColorHTML Script public connectors; constructor() { let connector = { name: "connector", sourcePoint: { x: 100, y: 100 }, targetPoint: { x: 200, y: 200 }, sourceDecorator: { shape:"openarrow", borderColor:"red" } }; this.connectors = [connector]; }	Property:connectors.sourceDecorator.style.strokeColorHTML Script public connectors: ConnectorModel[]; ngOnInit(): void { let connectors: ConnectorModel[] = [{ id: 'connector', type: 'Straight', sourcePoint: { x: 500, y: 100 }, targetPoint: { x: 600, y: 200 }, sourceDecorator: { shape: 'Arrow', style: { strokeColor: 'red' }, }, }]; }
Sets the border width of the decorator	Property:connectors.sourceDecorator.borderColorWidthHTML Script public connectors; constructor() { let connector = { name: "connector", sourcePoint: { x: 100, y: 100 }, targetPoint: { x: 200, y: 200 }, sourceDecorator: { shape:"openarrow", borderWidth: 5 } }; this.connectors = [connector]; }	Property:connectors.sourceDecorator.style.strokeWidth:5HTML Script public connectors: ConnectorModel[]; ngOnInit(): void { let connectors: ConnectorModel[] = [{ id: 'connector', type: 'Straight', sourcePoint: { x: 500, y: 100 }, targetPoint: { x: 600, y: 200 }, sourceDecorator: { shape: 'Arrow', strokeWidth: 5 }, }]; }
Defines to customize sourceDecorator appearance using user-defined CSS	Property:connectors.sourceDecorator.cssClassHTML Script public connectors; constructor() { let connector = { name: "connector", sourcePoint: { x: 100, y: 100 }, targetPoint: { x: 200, y: 200 }, sourceDecorator: { shape:"openarrow", cssClass:"hoverDecorator" } }; this.connectors = [connector]; }	Not applicable
Sets the fill color of the source decorator	Property:connectors.sourceDecorator.fillColorHTML Script public connectors; constructor() { let connector = { name: "connector", sourcePoint: { x: 100, y: 100 }, targetPoint: { x: 200, y: 200 }, sourceDecorator: { shape:"openarrow", fillColor:"red" } }; this.connectors = [connector]; }	Property:connectors.sourceDecorator.style.fillHTML Script public connectors: ConnectorModel[]; ngOnInit(): void { let connectors: ConnectorModel[] = [{ id: 'connector', type: 'Straight', sourcePoint: { x: 500, y: 100 }, targetPoint: { x: 600, y: 200 }, sourceDecorator: { shape: 'Arrow', fill: 'black' }, }]; }
Sets the height of	Property:connectors.sourceDecorator.heightHTML Script public connectors; constructor() { let connector =	Property:connectors.sourceDecorator.heightHTML Script public connectors: ConnectorModel[]; ngOnInit(): void

the source decorator	<pre>{ name: "connector", sourcePoint: { x: 100, y: 100 }, targetPoint: { x: 200, y: 200 }, sourceDecorator: { width: 10, height:10 } }; this.connectors = [connector]; }</pre>	<pre>{ let connectors: ConnectorModel[] = [{ id: 'connector', type: 'Straight', sourcePoint: { x: 500, y: 100 }, targetPoint: { x: 600, y: 200 }, sourceDecorator: { shape: 'Arrow', height: 10, width: 10 }, }]; }</pre>
Defines the custom shape of the source decorator	Property:connectors.sourceDecorator.pathData HTML Script public connectors; constructor() { let connector = { name: "connector", sourcePoint: { x: 100, y: 100 }, targetPoint: { x: 200, y: 200 }, sourceDecorator: { shape:"path", pathData:"M 376.892, 225.284 L 371.279,211.95 L 376.892,198.617 L 350.225,211.95 L 376.892,225.284 Z" } }; this.connectors = [connector]; }	Property:connectors.sourceDecorator.path HTML Script public connectors; constructor() { let connector = { name: "connector", sourcePoint: { x: 100, y: 100 }, targetPoint: { x: 200, y: 200 }, sourceDecorator: { shape:"path", pathData:"M 376.892, 225.284 L 371.279,211.95 L 376.892,198.617 L 350.225,211.95 L 376.892,225.284 Z" } }; this.connectors = [connector]; }
Defines the shape of the source decorator.	Property:connectors.sourceDecorator.shape HTML Script public connectors; constructor() { let connector = { name: "connector", sourcePoint: { x: 100, y: 100 }, targetPoint: { x: 200, y: 200 }, sourceDecorator: { shape:DecoratorShapes.Circle } }; this.connectors = [connector]; }	Property:connectors.sourceDecorator.shape HTML Script public connectors; constructor() { let connector = { name: "connector", sourcePoint: { x: 100, y: 100 }, targetPoint: { x: 200, y: 200 }, sourceDecorator: { shape:'Arrow', }, } }; this.connectors = [connector]; }
Defines the width of the source decorator	Property:connectors.sourceDecorator.width HTML Script public connectors; constructor() { let connector = { name: "connector", sourcePoint: { x: 100, y: 100 }, targetPoint: { x: 200, y: 200 }, sourceDecorator: { shape:"openarrow", width: 10, height:10 } }; this.connectors = [connector]; }	Property:connectors.sourceDecorator.width HTML Script public connectors; constructor() { let connector = { name: "connector", sourcePoint: { x: 100, y: 100 }, targetPoint: { x: 200, y: 200 }, sourceDecorator: { shape:'Arrow', width: 10, height:10 }, } }; this.connectors = [connector]; }
Sets the source node of the connector	Property:connectors.sourceNode HTML Script public connectors; constructor() { let node1 = {name:"source", offsetX:100, offsetY:100, width: 50, height: 50 }; let node2 = {name:"target", offsetX:300, offsetY:300, width: 50, height: 50 }; let connector = { name: "connector", sourceNode:"source", targetNode:"target" }; this.connectors = [connector]; }	Property:connectors.sourceID HTML Script public connectors; constructor() { let node1 = {name:"source", offsetX:100, offsetY:100, width: 50, height: 50 }; let node2 = {name:"target", offsetX:300, offsetY:300, width: 50, height: 50 }; let connector = { name: "connector", sourceID:"source", targetID:"target" }; this.connectors = [connector]; }

Defines the space to be left between the source node and the source point of a connector	Property:connectors.sourcePaddingHTML Script public connectors; constructor() { let model = {name:"source", offsetX:100, offsetY:100, width: 50, height: 50 }; let node2 = {name:"target", offsetX:300, offsetY:300, width: 50, height: 50 }; let connector = { name: "connector", sourceNode:"source", targetNode:"target", sourcePadding: 2, targetPadding: 2 }; this.connectors = [connector]; }	Property:connectors.hitPaddingHTML Script public connectors: ConnectorModel[]; ngOnInit(): void { let nodes: NodeModel[] = [{ id: 'source', width: 60, height: 60, offsetX: 75, offsetY: 90 }, { id: 'target', width: 75, height: 70, offsetX: 210, offsetY: 90 }]; let connectors: ConnectorModel[] = [{ id: 'connector', type: 'Straight', hitPadding: 2 sourceID: 'source', targetID: 'target' }]; }
Describes the start point of the connector	Property:connectors.sourcePointHTML Script public connectors; constructor() { let connector = { name: "connector", sourcePoint:{x:100, y:100}, targetPoint:{x:200, y:200} }; this.connectors = [connector]; }	Property:connectors.sourcePointHTML Script public connectors: ConnectorModel[]; ngOnInit(): void { let connectors: ConnectorModel[] = [{ id: 'connector', type: 'Straight', sourcePoint: { x: 100, y: 100 }, targetPoint: { x: 200, y: 200 }, }]; }
Sets the source port of the connector	Property:connectors.sourcePortHTML Script public connectors; constructor() { let model = {name:"source", offsetX:100, offsetY:100, width: 50, height: 50, ports:[{ name:"port", offset: { x:1, y:0.5 } }]} }; let node2 = {name:"target", offsetX:200, offsetY:200, width: 50, height: 50, ports:[{ name:"port1", offset: { x:0, y:0.5 } }]} }; let connector = { name:"connector", sourceNode:"source", targetNode:"target", sourcePort: "port", targetPort:"port1" }; this.connectors = [connector]; }	Property:connectors.sourcePortIDHTML Script public connectors: ConnectorModel[]; ngOnInit(): void { let nodeport1: PointPortModel = { id: 'port', shape: 'Square', offset: { x: 1, y: 0.5 } }; let nodeport2: PointPortModel = { id: 'port1', shape: 'Square', offset: { x: 0, y: 0.5 } }; let nodes: NodeModel[] = [{ id: 'source', width: 60, height: 60, offsetX: 75, offsetY: 90, ports: [nodeport1] }, { id: 'target', width: 75, height: 70, offsetX: 210, offsetY: 90, ports: [nodeport2] }]; let connectors: ConnectorModel[] = [{ id: 'connector', type: 'Straight', sourceID: 'source', targetID: 'target', sourcePortID: 'port', targetPortID: 'port1', }]; }
Defines the target decorator of the connector	Property:connectors.targetDecoratorHTML Script public connectors; constructor() { let connector = { name: "connector", sourcePoint: { x: 100, y: 100 }, targetPoint: { x: 200, y: 200 }, targetDecorator: { shape:"openarrow" } }; this.connectors = [connector]; }	Property:connectors.targetDecoratorHTML Script public connectors: ConnectorModel[]; ngOnInit(): void { let connectors: ConnectorModel[] = [{ id: 'connector', type: 'Straight', sourcePoint: { x: 500, y: 100 }, targetPoint: { x: 600, y: 200 }, targetDecorator: { shape: 'Arrow', }, }]; }

Sets the border color of the target decorator	Property:connectors.targetDecorator.borderColorHTML Script public connectors; constructor() { let connector = { name: "connector", sourcePoint: { x: 100, y: 100 }, targetPoint: { x: 200, y: 200 }, targetDecorator: { shape:"openarrow", borderColor:"red" } }; this.connectors = [connector]; }	Property:connectors.targetDecorator.style.strokeColorHTML Script public connectors: ConnectorModel[]; ngOnInit(): void { let connectors: ConnectorModel[] = [{ id: 'connector', type: 'Straight', sourcePoint: { x: 500, y: 100 }, targetPoint: { x: 600, y: 200 }, targetDecorator: { shape: 'Arrow', style: { strokeColor: 'red' }, }, }]; }
Sets the border width of the decorator	Property:connectors.targetDecorator.borderWidthHTML Script public connectors; constructor() { let connector = { name: "connector", sourcePoint: { x: 100, y: 100 }, targetPoint: { x: 200, y: 200 }, targetDecorator: { shape:"openarrow", borderWidth: 5 } }; this.connectors = [connector]; }	Property:connectors.targetDecorator.style.strokeWidth:5HTML Script public connectors: ConnectorModel[]; ngOnInit(): void { let connectors: ConnectorModel[] = [{ id: 'connector', type: 'Straight', sourcePoint: { x: 500, y: 100 }, targetPoint: { x: 600, y: 200 }, targetDecorator: { shape: 'Arrow', strokeWidth: 5 }, }]; }
Defines to customize target Decorator appearance using user-defined CSS	Property:connectors.targetDecorator.cssClassHTML Script public connectors; constructor() { let connector = { name: "connector", sourcePoint: { x: 100, y: 100 }, targetPoint: { x: 200, y: 200 }, targetDecorator: { shape:"openarrow", cssClass:"hoverDecorator" } }; this.connectors = [connector]; }	Not applicable
Sets the fill color of the target decorator	Property:connectors.targetDecorator.fillColorHTML Script public connectors; constructor() { let connector = { name: "connector", sourcePoint: { x: 100, y: 100 }, targetPoint: { x: 200, y: 200 }, targetDecorator: { shape:"openarrow", fillColor:"red" } }; this.connectors = [connector]; }	Property:connectors.targetDecorator.style.fillHTML Script public connectors: ConnectorModel[]; ngOnInit(): void { let connectors: ConnectorModel[] = [{ id: 'connector', type: 'Straight', sourcePoint: { x: 500, y: 100 }, targetPoint: { x: 600, y: 200 }, targetDecorator: { shape: 'Arrow', fill: 'black' }, }]; }
Sets the height of the target decorator	Property:connectors.targetDecorator.heightHTML Script public connectors; constructor() { let connector = { name: "connector", sourcePoint: { x: 100, y: 100 }, targetPoint: { x: 200, y: 200 }, targetDecorator: { width: 10, height:10 } }; this.connectors = [connector]; }	Property:connectors.targetDecorator.heightHTML Script public connectors: ConnectorModel[]; ngOnInit(): void { let connectors: ConnectorModel[] = [{ id: 'connector', type: 'Straight', sourcePoint: { x: 500, y: 100 }, targetPoint: { x: 600, y: 200 }, targetDecorator: { shape:

		'Arrow', height: 10, width: 10 }, }); }
Defines the custom shape of the target decorator	Property:connectors.targetDecorator.pathDataHTML Script public connectors; constructor() { let connector = { name: "connector", sourcePoint: { x: 100, y: 100 }, targetPoint: { x: 200, y: 200 }, targetDecorator: { shape:"path", pathData:"M 376.892,225.284 L 371.279,211.95 L 376.892,198.617 L 350.225,211.95 L 376.892,225.284 Z" } }; this.connectors = [connector]; }	Property:connectors.targetDecorator.pathDataHTML Script public connectors: ConnectorModel[]; ngOnInit(): void { let connectors: ConnectorModel[] = [{ id: 'connector', type: 'Straight', sourcePoint: { x: 500, y: 100 }, targetPoint: { x: 600, y: 200 }, targetDecorator: { shape: 'Custom', pathData:"M 376.892,225.284 L 371.279,211.95 L 376.892,198.617 L 350.225,211.95 L 376.892,225.284 Z" }, }]; }
Defines the shape of the target decorator.	Property:connectors.targetDecorator.shapeHTML Script public connectors; constructor() { let connector = { name: "connector", sourcePoint: { x: 100, y: 100 }, targetPoint: { x: 200, y: 200 }, targetDecorator: { shape: DecoratorShapes.Circle } }; this.connectors = [connector]; }	Property:connectors.targetDecorator.shapeHTML Script public connectors: ConnectorModel[]; ngOnInit(): void { let connectors: ConnectorModel[] = [{ id: 'connector', type: 'Straight', sourcePoint: { x: 500, y: 100 }, targetPoint: { x: 600, y: 200 }, targetDecorator: { shape: 'Arrow', }, }]; }
Defines the width of the target decorator	Property:connectors.targetDecorator.widthHTML Script public connectors; constructor() { let connector = { name: "connector", sourcePoint: { x: 100, y: 100 }, targetPoint: { x: 200, y: 200 }, targetDecorator: { shape:"openarrow", width: 10, height:10 } }; this.connectors = [connector]; }	Property:connectors.targetDecorator.widthHTML Script public connectors: ConnectorModel[]; ngOnInit(): void { let connectors: ConnectorModel[] = [{ id: 'connector', type: 'Straight', sourcePoint: { x: 500, y: 100 }, targetPoint: { x: 600, y: 200 }, targetDecorator: { shape: 'Arrow', width: 10, height:10 }, }]; }
Sets the target node of the connector	Property:connectors.targetNodeHTML Script public connectors; constructor() { let node1 = {name:"source", offsetX:100, offsetY:100, width: 50, height: 50 }; let node2 = {name:"target", offsetX:300, offsetY:300, width: 50, height: 50 }; let connector = { name: "connector", sourceNode:"source", targetNode:"target" }; this.connectors = [connector]; }	Property:connectors.targetIDHTML Script public connectors: ConnectorModel[]; ngOnInit(): void { let nodes: NodeModel[] = [{ id: 'source', width: 60, height: 60, offsetX: 75, offsetY: 90 }, { id: 'target', width: 75, height: 70, offsetX: 210, offsetY: 90 }]; let connectors: ConnectorModel[] = [{ id: 'connector', type: 'Straight', sourceID: 'source', targetID: 'target' }]; }
Defines the space to be left between	Property:connectors.targetPaddingHTML Script public connectors; constructor() { let node1 = {name:"source", offsetX:100,	Property:connectors.hitPaddingHTML Script public connectors: ConnectorModel[]; ngOnInit(): void { let nodes: NodeModel[] = [{ id:

the target node and the target point of a connector	<pre>offsetY:100, width: 50, height: 50 }; let node2 = {name:"target", offsetX:300, offsetY:300, width: 50, height: 50 }; let connector = { name: "connector", sourceNode:"source", targetNode:"target", sourcePadding: 2, targetPadding: 2 }; this.connectors = [connector]; }</pre>	<pre>'source', width: 60, height: 60, offsetX: 75, offsetY: 90 }, { id: 'target', width: 75, height: 70, offsetX: 210, offsetY: 90 }]; let connectors: ConnectorModel[] = [{ id: 'connector', type: 'Straight', hitPadding: 2 sourceID: 'source', targetID: 'target' }]; }</pre>
Describes the start point of the connector	Property:connectors.targetPointHTML Script public connectors; constructor() { let connector = { name: "connector", sourcePoint:{x:100, y:100}, targetPoint:{x:200, y:200} }; this.connectors = [connector]; }	Property:connectors.targetPointHTML Script public connectors: ConnectorModel[]; ngOnInit(): void { let connectors: ConnectorModel[] = [{ id: 'connector', type: 'Straight', sourcePoint: { x: 100, y: 100 }, targetPoint: { x: 200, y: 200 }, }]; }
Sets the target port of the connector	Property:connectors.targetPortHTML Script public connectors; constructor() { let model = {name:"source", offsetX:100, offsetY:100, width: 50, height: 50, ports:[{ name:"port", offset: { x:1, y:0.5 } }]} }; let node2 = {name:"target", offsetX:200, offsetY:200, width: 50, height: 50, ports:[{ name:"port1", offset: { x:0, y:0.5 } }]} }; let connector = { name:"connector", sourceNode:"source", targetNode:"target", sourcePort:"port", targetPort:"port1" }; this.connectors = [connector]; }	Property:connectors.targetPortIDHTML Script public connectors: ConnectorModel[]; ngOnInit(): void { let nodeport1: PointPortModel = { id: 'port', shape: 'Square', offset: { x: 1, y: 0.5 } }; let nodeport2: PointPortModel = { id: 'port1', shape: 'Square', offset: { x: 0, y: 0.5 } }; let nodes: NodeModel[] = [{ id: 'source', width: 60, height: 60, offsetX: 75, offsetY: 90, ports: [nodeport1] }, { id: 'target', width: 75, height: 70, offsetX: 210, offsetY: 90, ports: [nodeport2] }]; let connectors: ConnectorModel[] = [{ id: 'connector', type: 'Straight', sourceID: 'source', targetID: 'target', sourcePortID: 'port', targetPortID: 'port1', }]; }
Defines the tooltip that should be shown when the mouse hovers over connector	Property:connectors.tooltipHTML Script public connectors; constructor() { let tooltip = { templateId: "mouseOverTooltip", alignment: { horizontal: "center", vertical: "bottom" } }; let ConnectorConstraints = ConnectorConstraints; let connector = { name: "flow", sourcePoint: { x:100, y: 100 }, targetPoint :{ x:200, y:200 }, constraints: ConnectorConstraints.Default & ~ConnectorConstraints.InheritToo	Property:connectors.tooltipHTML Script public connectors: ConnectorModel[]; ngOnInit(): void { let connectors: ConnectorModel[] = [{ id: 'connector', type: 'Straight', sourcePoint: { x:100, y: 100 }, targetPoint :{ x:200, y:200 }, constraints: ConnectorConstraints.Default ConnectorConstraints.Tooltip, tooltip: { content: 'Connector', position: 'TopCenter', showTipPointer: true, }, }]; }

	<code>ltip, tooltip:tooltip }; this.connectors = [connector]; }</code>	
Sets the vertical alignment of connector	Property:connectors.verticalAlignHTML Script <code>public connectors; constructor() { let connector = { name:"connector", sourcePoint:{x:100, y:100}, targetPoint:{x:150, y:150}, verticalAlign:VerticalAlignment.Bottom }; this.connectors = [connector]; }</code>	Not applicable
Enables or disables the visibility of connector	Property:connectors.visibleHTML Script <code>public connectors; constructor() { let connector = { name:"connector", sourcePoint:{x:100, y:100}, targetPoint:{x:200, y:200}, visible: true }; this.connectors = [connector]; }</code>	Property:connectors.visibleHTML Script <code>public connectors: ConnectorModel[]; ngOnInit(): void { let connectors: ConnectorModel[] = [{ id: 'connector', type: 'Straight', sourcePoint:{x:100, y:100}, targetPoint:{x:200, y:200}, visible: true }]; }</code>
Sets the z-index of the connector	Property:connectors.zOrderHTML Script <code>public connectors; constructor() { let connector = { name:"connector", sourcePoint:{x:100, y:100}, targetPoint:{x:200, y:200}, zOrder: 4 }; this.connectors = [connector]; }</code>	Property:connectors.zIndexHTML Script <code>public connectors: ConnectorModel[]; ngOnInit(): void { let connectors: ConnectorModel[] = [{ id: 'connector', type: 'Straight', sourcePoint:{x:100, y:100}, targetPoint:{x:200, y:200}, zIndex: 1 }]; }</code>
Binds the custom JSON data with connector properties	Property:connectors.connectorTemplateHTML Script <code>public dataSourceSettings; public connectorTemplate; constructor() { let data = [{ "Id": "E1", "Name": "Maria Anders", "Designation": "Managing Director" }, { "Id": "E2", "Name": "Ana Trujillo", "Designation": "Project Manager", "ReportingPerson": "E1" }]; dataSourceSettings = { id: "Id", parent: "ReportingPerson", dataSource: data }, connectorTemplate: "connectorTemplate" public connectorTemplate(diagram, connector) { if(connector.sourceNode && connector.targetNode){ connector.lineColor = "green"; } } }</code>	Not applicable
Enables/Disables the	Property:constraintsHTML Script <code>let DiagramConstraints =</code>	Property:constraintsHTML Script <code>public constraints: DiagramConstraints;</code>

default behaviors of the diagram	<pre>DiagramConstraints; public constraints; constructor() { let constraints = DiagramConstraints.Default DiagramConstraints.Bridging; this.connectors = [connector]; }</pre>	<pre>ngOnInit(): void { this.constraints =DiagramConstraints.Default DiagramConstraints.Bridging }</pre>
----------------------------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	------------------------------------------------------------------------------------------------------------

ContextMenu

<!-- markdownlint-disable MD033 -->

Behavior	API in Essential JS 1	API in Essential JS 2
Defines the collection of context menu items	Property:contextMenu.itemsHTML Script public contextMenu; public enableContextMenu; constructor() { let menuItems = [{ "name": "hyperLink" }]; this.contextMenu = {items: menuItems }; this.enableContextMenu = true; }	Property:contextMenuSettings.itemsHTML Script public constraints: ContextMenuSettingsModel; ngOnInit(): void { this.contextMenuSettings = { show: true, items: [{ text: 'delete', id: 'delete', target: '.e-diagramcontent', iconCss: 'e-copy' }], } }
Defines the text for the collection of context menu item	Property:contextMenu.items.textHTML Script public contextMenu; public enableContextMenu; constructor() { let menuItems = [{ "text": "ZoomIn" }]; this.contextMenu = {items: menuItems }; this.enableContextMenu = true; }	Property:contextMenuSettings.items.textHTML Script public constraints: ContextMenuSettingsModel; ngOnInit(): void { this.contextMenuSettings = { show: true, items: [{ text: 'ZoomIn' }], } }
Defines the name for the collection of context menu items	Property:contextMenu.items.nameHTML Script public contextMenu; public enableContextMenu; constructor() { let menuItems = [{ "name": "hyperLink" }]; this.contextMenu = {items: menuItems }; this.enableContextMenu = true; }	Property:contextMenuSettings.items.idHTML Script public constraints: ContextMenuSettingsModel; ngOnInit(): void { this.contextMenuSettings = { show: true, items: [{ text: 'delete', id: 'delete' }], } }
Defines the image url for the collection of context menu items	Property:contextMenu.items.imageUrlHTML Script public contextMenu; public enableContextMenu; constructor() { let menuItems = [{ "name": "zoomin", "text": "ZoomIn", "imageUrl": "Images/zoomin.png", "style": "" }]; this.contextMenu = {items: menuItems }; this.enableContextMenu = true; }	Property:contextMenuSettings.items.urlHTML Script public constraints: ContextMenuSettingsModel; ngOnInit(): void { this.contextMenuSettings = { show: true, items: [{ 'id': 'zoomin', 'text': 'ZoomIn', 'url': 'Images/zoomin.png', }], } }
Defines the cssClass for the collection	Property:contextMenu.items.cssClassHTML Script public contextMenu; public enableContextMenu; constructor() { let menuItems = [{ "name": "zoomin", "text":	Property:contextMenuSettings.items.iconCSSHTML Script public constraints: ContextMenuSettingsModel; ngOnInit(): void { this.contextMenuSettings = { show:

of context menu items	<pre>"ZoomIn", "imageUrl": "Images/zoomin.png", "cssClass": "menu", "style": "" }]; this.contextMenu = {items: menuItems }; this.enableContextMenu = true; }</pre>	<pre>true, items: [{ text: 'delete', id: 'delete', target: '.e- diagramcontent', iconCss: 'e-copy' }], } }</pre>
Defines the collection of sub items for the context menu items	<pre>Property:contextMenu.items.subItemsHTML Script public contextMenu; public enableContextMenu; constructor() { this.contextMenu = { // Defines the custom context menu items items: [{ name: "zoom", // Text to be displayed text: "Zoom", // Defines the sub items subItems: [{name: "zoomIn", text: "ZoomIn"}, {name: "zoomOut",text: "ZoomOut"}] }] }; this.enableContextMenu = true; }</pre>	<pre>Property:contextMenuSettings.itemsHTML Script public constraints: ContextMenuSettingsModel; ngOnInit(): void { this.contextMenuSettings = { show: true, items: [{ text: 'Zoom', id: 'zoom', items: [{name: "zoomIn", text: "ZoomIn"}, {name: "zoomOut",text: "ZoomOut"}] }], showCustomMenuOnly: false, } }</pre>
set whether to display the default context menu items or not	<pre>Property:contextMenu.showCustomMenuItemsOnlyHTML Script public contextMenu; public enableContextMenu; constructor() { this.contextMenu = { showCustomMenuItemsOnly: true }; this.enableContextMenu = true; }</pre>	<pre>Property:contextMenuSettings.showCustomMenuOnlyHTML Script public constraints: ContextMenuSettingsModel; ngOnInit(): void { this.contextMenuSettings = { showCustomMenuOnly: false, } }</pre>
separator	Not applicable	<pre>Property:contextMenuSettings.items.separatorHTML Script public constraints: ContextMenuSettingsModel; ngOnInit(): void { this.contextMenuSettings = { show: true, items: [{ text: 'Save', id: 'save', target: '.e- diagramcontent', iconCss: 'e- save', separator: true }, { text: 'Load', id: 'load', target: '.e- diagramcontent', iconCss: 'e-load' }], } }</pre>
Define the target to show the menu item.	Not applicable	<pre>Property:contextMenuSettings.items.targetHTML Script public constraints: ContextMenuSettingsModel; ngOnInit(): void { this.contextMenuSettings = { show: true, items: [{ text: 'delete', id: 'delete', target: '.e- diagramcontent', iconCss: 'e-copy' }], showCustomMenuOnly: false, } }</pre>
Enables/Disables the	Not applicable	<pre>Property:contextMenuSettings.showHTML Script public constraints: ContextMenuSettingsModel; ngOnInit(): void {</pre>

context menu items		<pre>this.contextMenuSettings = { show: true, items: [{ text: 'delete', id: 'delete', target: '.e- diagramcontent', iconCss: 'e-copy' }], showCustomMenuOnly: false, }</pre>
--------------------	--	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

DataSourceSettings

<!-- markdownlint-disable MD033 -->

Behavior	API in Essential JS 1	API in Essential JS 2
Defines the data source either as a collection of objects or as an instance of ej.DataManager	Property: dataSourceSettings.dataSourceHTML Script <pre>public dataSourceSettings; constructor() { let data = [{ "Id": "E1", "Name": "Maria Anders", "Designation": "Managing Director" }, { "Id": "E2", "Name": "Ana Trujillo", "Designation": "Project Manager", "ReportingPerson": "E1" }]; this.dataSourceSettings = { dataSource: data } }</pre>	Property: dataSourceSettings.dataSourceHTML Script <pre>public diagram: DiagramComponent; public dataSourceSettings: Object; ngOnInit(): void { let items: object[] = [{ "Id": "E1", "Name": "Maria Anders", "Designation": "Managing Director" }, { "Id": "E2" , "Name": "Ana Trujillo", "Designation": "Project Manager", "ReportingPerson": "E1" }] ; this.dataSourceSettings = { dataSource: items } }</pre>
Sets the unique id of the data source items	Property: dataSourceSettings.idHTML Script <pre>public dataSourceSettings; constructor() { let data = [{ "Id": "E1", "Name": "Maria Anders", "Designation": "Managing Director" }, { "Id": "E2", "Name": "Ana Trujillo", "Designation": "Project Manager", "ReportingPerson": "E1" }]; this.dataSourceSettings = { id: "Id", dataSource: data } }</pre>	Property: dataSourceSettings.idHTML Script <pre>public diagram: DiagramComponent; public dataSourceSettings: Object; ngOnInit(): void { let items: object[] = [{ "Id": "E1", "Name": "Maria Anders", "Designation": "Managing Director" }, { "Id": "E2" , "Name": "Ana Trujillo", "Designation": "Project Manager", "ReportingPerson": "E1" }] ; this.dataSourceSettings = { id: 'Id', dataSource: items } }</pre>

Defines the parent id of the data source item	<pre>Property:dataSourceSettings.parentHTML Script public dataSourceSettings; constructor() { let data = [{ "Id": "E1", "Name": "Maria Anders", "Designation": "Managing Director" }, { "Id": "E2", "Name": "Ana Trujillo", "Designation": "Project Manager", "ReportingPerson": "E1" }]; this.dataSourceSettings = { id: "Id", parent: "ReportingPerson", dataSource: data } }</pre>	<pre>Property:dataSourceSettings. parentIdHTML Script public diagram: DiagramComponent; public dataSourceSettings: Object; ngOnInit(): void { let items: object[] = [{ "Id": "E1", "Name": "Maria Anders", "Designation": "Managing Director" }, { "Id": "E2" , "Name": "Ana Trujillo", "Designation": "Project Manager", "ReportingPerson": "E1" }]; this.dataSourceSettings = { id: 'Id', parentId: 'ReportingPerson', dataManager: items } }</pre>
Describes query to retrieve a set of data from the specified datasource	<pre>Property:dataSourceSettings.queryHTML Script public dataSourceSettings; constructor() { this.dataSourceSettings = { dataSource: ej.DataManager({ url: "http://mvc.syncfusion.com/Services/Northwnd. svc/" }), query: ej.Query().from("Employees").select("Employee ID,ReportsTo,FirstName"), tableName: "Employees", id: "EmployeeID", parent: "ReportsTo" } }</pre>	Not applicable
Sets the unique id of the root data source item	<pre>Property:dataSourceSettings.rootHTML Script public dataSourceSettings; constructor() { let data = [{ "Id": "E1", "Name": "Maria Anders", "Designation": "Managing Director" }, { "Id": "E2", "Name": "Ana Trujillo", "Designation": "Project Manager", "ReportingPerson": "E1" }]; this.dataSourceSettings = { id: "Id", parent: "ReportingPerson", root: "E1", dataSource: data } }</pre>	<pre>Property:dataSourceSettings.r ootHTML Script public diagram: DiagramComponent; public dataSourceSettings: Object; ngOnInit(): void { let items: object[] = [{ "Id": "E1", "Name": "Maria Anders", "Designation": "Managing Director" }, { "Id": "E2" , "Name": "Ana Trujillo", "Designation": "Project Manager", "ReportingPerson": "E1" }]; this.dataSourceSettings = { id: 'Id', parentId: 'ReportingPerson', dataManager: items, root: 'E1' } }</pre>

Describes the name of the table on which the specified query has to be executed	Property: <code>dataSourceSettings.tableName</code>HTML Script <pre>public dataSourceSettings; constructor() { this.dataSourceSettings = { dataSource: ej.DataManager({ url: "http://mvc.syncfusion.com/Services/Northwnd. svc/" }), query: ej.Query().from("Employees").select("Employee ID,ReportsTo,FirstName"), //Table name tableName: "Employees", id: "EmployeeID", parent: "ReportsTo" } } }</pre>	Not applicable
Specifies the method name which is used to get the updated data from client side to the server side	Property: <code>dataSourceSettings.crudAction</code>HTML Script <pre>public dataSourceSettings; constructor() { this.dataSourceSettings = { id: "Name", crudAction: { read: "http://js.syncfusion.com/demos/ejservices/ap i/Diagram/GetNodes" } } }</pre>	Not applicable
Specifies the create method which is used to get the nodes to be added from client side to the server side	Property: <code>dataSourceSettings.crudAction.create</code>HTML Script <pre>public dataSourceSettings; constructor() { this.dataSourceSettings = { id: "Name", crudAction: { create: "http://js.syncfusion.com/demos/ejservices/ap i/Diagram/AddNodes", } } }</pre>	Not applicable
Specifies the update method which is used to get the updated	Property: <code>dataSourceSettings.crudAction.update</code>HTML Script <pre>public dataSourceSettings; constructor() { this.dataSourceSettings = { id: "Name", crudAction: { update: "http://js.syncfusion.com/demos/ejservices/ap i/Diagram/UpdateNodes", } } }</pre>	Not applicable

data from client side to the server side		
Specifies the destroy method which is used to get the deleted items data from client side to the server side	Property: dataSourceSettings.crudAction.destroyHTML Script public dataSourceSettings; constructor() { this.dataSourceSettings = { id: "Name", crudAction: { destroy: "http://js.syncfusion.com/demos/ejservices/ap i/Diagram/DeleteNodes" } } }	Not applicable
Specifies the read method to get the created nodes from client side to the server side	Property: dataSourceSettings.crudAction.readHTML Script public dataSourceSettings; constructor() { let data = [this.dataSourceSettings = { id: "Name", crudAction: { read: "http://js.syncfusion.com/demos/ejservices/ap i/Diagram/GetNodes"; } } }	Not applicable
Defines the data source either as a collection of objects or as an instance of ej.DataManager	Property: dataSourceSettings.customFieldsHTML Script public dataSourceSettings; constructor() { this.dataSourceSettings = { id: 'Name', customFields: ["Description", "Color"] } }	Property: dataSourceSettings. dataHTML Script public diagram: DiagramComponent; public dataSourceSettings: Object; ngOnInit(): void { this.dataSourceSettings = { id: 'Name', customFields: ["Description", "Color"] } }

Defines the data source either as a collection of objects or as an instance of ej.DataManager	Property: <code>dataSourceSettings.connectionDataSourceHTML</code> <pre>HTML Script public dataSourceSettings; constructor() { this.dataSourceSettings = { id: "Name", connectionDataSource: { id: "Name" } } }</pre>	Not applicable
Sets the datasource for the connection datasource settings items	Property: <code>dataSourceSettings.connectionDataSource.dataSourceHTML</code> <pre>HTML Script public dataSourceSettings; constructor() { let data = [{ "Id": "E1", "Name": "Maria Anders", "Designation": "Managing Director" }, { "Id": "E2", "Name": "Ana Trujillo", "Designation": "Project Manager", "ReportingPerson": "E1" }]; this.dataSourceSettings = { id: "Name", connectionDataSource: { id: "Name", dataSource: data } }</pre>	Not applicable
Sets the unique id of the connection data source item	Property: <code>dataSourceSettings.connectionDataSource.idHTML</code> <pre>HTML Script public dataSourceSettings; constructor() { this.dataSourceSettings = { id: "Name", connectionDataSource: { id: "Name" } } }</pre>	Not applicable
Sets the source node of the connection data source item	Property: <code>dataSourceSettings.connectionDataSource.sourceNodeHTML</code> <pre>HTML Script public dataSourceSettings; constructor() { this.dataSourceSettings = { id: "Name", connectionDataSource: { id: "Name", sourceNode: "sourceNode", } } }</pre>	Not applicable
Sets the target node of the connection data source item	Property: <code>dataSourceSettings.connectionDataSource.targetNodeHTML</code> <pre>HTML Script public dataSourceSettings; constructor() { this.dataSourceSettings = { id: "Name", connectionDataSource: { id: "Name", targetNode: "targetNode" } } }</pre>	Not applicable

Sets the sourcePointX value of the connection data source item	Property: <code>dataSourceSettings.connectionDataSource.sourcePointX</code> HTML Script <pre>public dataSourceSettings; constructor() { this.dataSourceSettings = { id: "Name", connectionDataSource: { id: "Name", sourcePointX:200 } } }</pre>	Not applicable
Sets the sourcePointY value of the connection data source item	Property: <code>dataSourceSettings.connectionDataSource.sourcePointY</code> HTML Script <pre>public dataSourceSettings; constructor() { this.dataSourceSettings = { id: "Name", connectionDataSource: { id: "Name", sourcePointY:200 } } }</pre>	Not applicable
Sets the targetPointX value of the connection data source item	Property: <code>dataSourceSettings.connectionDataSource.targetPointX</code> HTML Script <pre>public dataSourceSettings; constructor() { this.dataSourceSettings = { id: "Name", connectionDataSource: { id: "Name", targetPointX:200 } } }</pre>	Not applicable
Sets the targetPointY value of the connection data source item	Property: <code>dataSourceSettings.connectionDataSource.targetPointY</code> HTML Script <pre>public dataSourceSettings; constructor() { this.dataSourceSettings = { id: "Name", connectionDataSource: { id: "Name", targetPointY:200 } } }</pre>	Not applicable
Specifies the method name which is used to get updated connectors from client side to the	Property: <code>dataSourceSettings.connectionDataSource.crudAction</code> HTML Script <pre>public dataSourceSettings; constructor() { this.dataSourceSettings = { id: "Name", connectionDataSource: { id: "Name", sourceNode: "sourceNode", targetNode: "targetNode", crudAction: { read: "http://js.syncfusion.com/demos/ejservices/api/Diagram/GetConnectors" } } } }</pre>	Not applicable

server side		
Specifies the create method which is used to get the connectors to be added from client side to the server side	Property: <code>dataSourceSettings.connectionDataSource.crudAction.createHTML</code> <pre>Script public dataSourceSettings; constructor() { this.dataSourceSettings = { id: "Name", connectionDataSource: { id: "Name", sourceNode: "sourceNode", targetNode: "targetNode", crudAction: { create: "http://js.syncfusion.com/demos/ejservices/ap i/Diagram/AddConnectors", } } } }</pre>	Not applicable
Specifies the update method which is used to get the updated connectors from client side to the server side	Property: <code>dataSourceSettings.connectionDataSource.crudAction.updateHTML</code> <pre>Script public dataSourceSettings; constructor() { this.dataSourceSettings = { id: "Name", connectionDataSource: { id: "Name", crudAction: { update: "http://js.syncfusion.com/demos/ejservices/ap i/Diagram/UpdateConnectors", } } } }</pre>	Not applicable
Specifies the destroy method which is used to get the deleted items data from client side to the	Property: <code>dataSourceSettings.connectionDataSource.crudAction.destroyHTML</code> <pre>Script public dataSourceSettings; constructor() { this.dataSourceSettings = { id: "Name", connectionDataSource: { id: "Name", crudAction: { destroy: "http://js.syncfusion.com/demos/ejservices/ap i/Diagram/DeleteConnectors" } } } }</pre>	Not applicable

server side		
Specifies the read method which is used to get the data from client side to the server side	Property: <code>dataSourceSettings.connectionDataSource.crudAction.readHTML</code> Script <pre>public dataSourceSettings; constructor() { this.dataSourceSettings = { id: "Name", connectionDataSource: { id: "Name", crudAction: { read: "http://js.syncfusion.com/demos/ejservices/api/Diagram/GetConnectors" } } } }</pre>	Not applicable
Specifies the custom fields to get the updated data from client side to the server side	Property: <code>dataSourceSettings.connectionDataSource.customFieldsHTML</code> Script <pre>public dataSourceSettings; constructor() { this.dataSourceSettings = { id: "Name", connectionDataSource: { id: "Name", customFields: ["Description", "Color"] } } }</pre>	Not applicable
Binds the custom data with node model	Property: <code>dataSourceSettings.doBindingHTML</code> Script <pre>public dataSourceSettings; public layout; constructor() { this.layout = { type: 'HierarchicalTree', verticalSpacing: 40 } this.dataSourceSettings = { id: 'Name', parentId: 'ReportingPerson', dataManager: items, doBinding: (nodeModel: NodeModel, data: object, diagram: Diagram) => { nodeModel.annotations = [{ content: data['Name'], margin: { top: 10 } }]; } }</pre>	Not applicable

DefaultSettings

<!-- markdownlint-disable MD033 -->

Behavior	API in Essential JS 1	API in Essential JS 2
Initializes the default values for nodes and connectors	Property: <code>defaultSettings.nodeHTML</code> Script <pre>public defaultSettings; constructor() { this.defaultSettings = { node: { fillColor:"red" } }; }</pre>	Property: <code>getNodeDefaultsHTML</code> Script <pre>public diagram: DiagramComponent; public getNodeDefaults(node: NodeModel, diagram: Diagram):</pre>

		<pre>NodeModel { node.style = { fill: 'blue', strokeColor: 'none', strokeWidth: 2 }; return node;}</pre>
Initializes the default connector properties	Property:defaultSettings.connectorHTML <pre>Script public defaultSettings; constructor() { this.defaultSettings = { connector: { lineColor:"red", lineWidth:4, lineDashArray:"2,2" } }; }</pre>	Property:getConnectorDefaultsHTML <pre>Script public diagram: DiagramComponent; public getConnectorDefaults(connector: ConnectorModel, diagram: Diagram): ConnectorModel { connector= { targetDecorator: { shape: 'None' }, type: 'Orthogonal' }; return connector;}</pre>
Initializes the default properties of groups	Property:defaultSettings.groupHTML <pre>Script public defaultSettings; constructor() { this.defaultSettings = { group: {constraints: NodeConstraints.Default & ~NodeConstraints.Drag } }; }</pre>	Not applicable

DrawType

<!-- markdownlint-disable MD033 -->

Behavior	API in Essential JS 1	API in Essential JS 2
Sets the type of JSON object to be drawn through drawing tool	Property:drawTypeHTML Script <pre>public drawType; constructor() { this.drawType ={"type":"node"}; }</pre>	Property:drawingObjectHTML <pre>Script public diagram: DiagramComponent; public diagramCreate(args: Object): void { this.diagram.drawingObject = {id: 'connector', type: 'Straight'}; }</pre>

EnableAutoScroll

<!-- markdownlint-disable MD033 -->

Behavior	API in Essential JS 1	API in Essential JS 2
Enables or disables auto scroll in diagram	Property:enableAutoScrollHTML <pre>Script public enableAutoScroll; constructor() { this.enableAutoScroll = false; }</pre>	Property:canAutoScrollHTML <pre>Script public canAutoScroll:boolean; ngOnInit(): void { this.canAutoScroll= true; }</pre>

EnableContextMenu

<!-- markdownlint-disable MD033 -->

Behavior	API in Essential JS 1	API in Essential JS 2
Enables or disables diagram context menu	Property:enableContextMenuHTML Script public enableContextMenu; constructor() { this.enableContextMenu = false; }	Property:contextMenuSettings.showHTML Script public contextMenuSettings; ngOnInit(): void { this. contextMenuSettings: { show: true } }

EnablePersistence

<!-- markdownlint-disable MD033 -->

Behavior	API in Essential JS 1	API in Essential JS 2
Enable or disable persisting component's state between page reloads/b>	Not applicable	Property:enablePersistenceHTML

EnableRtl

<!-- markdownlint-disable MD033 -->

Behavior	API in Essential JS 1	API in Essential JS 2
Enable or disable rendering component in right to left direction	Not applicable	Property:enableRtlHTML

GetCustomTool

<!-- markdownlint-disable MD033 -->

Behavior	API in Essential JS 1	API in Essential JS 2
Allows to get the custom tool	Not applicable	Property:getCustomToolHTML Script public getTool(action: string): ToolBase { let tool: ToolBase; if (action === 'userHandle') { tool = new CloneTool(diagram.commandHandler, true); } return tool; } public nodes: NodeModel[] = [{ id: 'node1', width: 100, height: 100, offsetX: 100, offsetY: 100, }, { id: 'node2', width: 100, height: 100, offsetX: 300, offsetY: 100, shape: { type: 'Basic', shape: 'Ellipse' }, }]; } class CloneTool extends ToolBase { public mouseDown(args: MouseEventArgs): void { super.mouseDown(args); diagram.copy(); diagram.paste(); } }

Height

<!-- markdownlint-disable MD033 -->

Behavior	API in Essential JS 1	API in Essential JS 2
Specifies the height of the diagram	Property:heightHTML	Property:heightHTML

HistoryManager

<!-- markdownlint-disable MD033 -->

Behavior	API in Essential JS 1	API in Essential JS 2
A method that takes a history entry as argument and returns whether the specific entry can be popped or not	Property:historyManager.canPopHTML Script let diagram = \$("#diagramCore").ejDiagram("instance"); let entry = { object: diagram.node, prevState: diagram.node.EmployeeInfo }; diagram.widget.model.historyManager.push(entry); let newValues = { role: "New role" }; node.EmployeeInfo = newValues; //Pop if the change doesn't need to be tracked if (diagram.widget.model.historyManager.canPop(entry)) diagram.widget.model.historyManager.pop();	Not applicable
A method that ends grouping the changes	Property:historyManager.closeGroupActionHTML Script let group = this.diagram.widget.model.selectedItems; // Start to group the changes this.diagram.widget.model.historyManager.startGroupAction(); //Makes the changes for (let i = 0; i	Property:historyList.endGroupActionHTML Script public nodes= [{ id: 'node1', offsetX: 100, offsetY: 100, width: 100, height: 100, }, { offsetX: 200, offsetY: 200, width: 100, height: 100, id: 'node2' }], public connectors = [{ id: 'connector1', sourcePoint: { x: 100, y: 200 }, targetPoint: { x: 200, y: 300 }, type: 'Orthogonal' }] }); public created() { let objects: (NodeModel ConnectorModel)[] = []; objects.push(this.diagram.nodes[0], this.diagram.nodes[1], this.diagram.connectors[0]); this.diagram.historyList.startGroupAction(); this.diagram.distribute('Top', objects); this.diagram.distribute('Bott

		<pre>om', objects); this.diagram.distribute('BottomToTop', objects); this.diagram.historyList.endGroupAction(); }</pre>
A method that removes the history of a recent change made in diagram	Property:historyManager.popHTML Script <pre>public created() { let diagram = \$("#diagramContent").ejDiagram("instance"); diagram.model.historyManager.pop(); }</pre>	Not applicable
A method that allows to track the custom changes made in diagram	Property:historyManager.pushHTML Script <pre>public created() { let entry = { object: node, prevState: node.employeeInfo }; this.diagram.model.historyManager.push(entry); let value = { role: "New role" }; node.employeeInfo = value; }</pre>	Property:historyList.pushHTML Script <pre>public nodes= [{ id: 'node1', offsetX: 100, offsetY: 100, width: 100, height: 100, }, { offsetX: 200, offsetY: 200, width: 100, height: 100, id: 'node2' }], public created() { let object = diagram.nodes[0]; object['description'] = (document.getElementById('custom') as HTMLSelectElement).value; let entry: HistoryEntry = { undoObject: object }; this.diagram.historyList.push(entry); this.diagram.dataBind(); }</pre>
Defines what should be happened while trying to restore a custom change	Property:historyManager.redoHTML Script <pre>public historyManager; this.historyManager: { undo: customUndoRedo, redo: customUndoRedo } public created() { let diagram = \$("#diagramContent").ejDiagram("instance"); let node: Node = args.object; let currentState = node.employeeInfo; node.employeeInfo = args.prevState; args.prevState = currentState; }</pre>	Property:historyList.redoHTML Script <pre>public nodes= [{ id: 'node1', offsetX: 100, offsetY: 100, width: 100, height: 100, }, { offsetX: 200, offsetY: 200, width: 100, height: 100, id: 'node2' }], public created() { let node1: NodeModel = this.diagram.nodes[0]; node1['customName'] = 'customNode'; let entry = { undoObject: node1 }; this.diagram.historyList.push(entry); this.diagram.historyList.undo = function(args: HistoryEntry) { args.redoObject = cloneObject(args.undoObject) }</pre>

		<pre> as NodeModel; args.undoObject['customName'] = 'customNodeChange'; } this.diagram.historyList.redo = function(args: HistoryEntry) { let current: NodeModel = cloneObject(args.undoObject) as NodeModel; args.undoObject['customName'] = args.redoObject['customName'] ; args.redoObject = current; } } </pre>
Gets the number of redo actions to be stored on the history manager. Its an read-only property and the collection should not be modified	Property: historyManager.redoStack let diagram = \$("#diagramContent").ejDiagram("instance"); diagram.model.historyManager.redoStack();	Property: historyList.redoStackHTML Script public nodes= [{ offsetX: 100, offsetY: 100, width: 100, height: 100, }] public created() { let diagram = \$("#diagramContent").ejDiagram("instance"); diagram.historyList.redoStack(); }
Restricts the undo and redo actions to a certain limit	Property: historyManager.stackLimitHTML Script public nodes= [{ offsetX: 100, offsetY: 100, width: 100, height: 100, }] public created() { this.diagram.model.historyManager.stackLimit(); }	Not applicable
A method that starts to group the changes to revert/re store	Property: historyManager.startGroupActionHTML Script public nodes= [{ offsetX: 100, offsetY: 100, width: 100, height: 100, }] public created() { let group = this.diagram.widget.model.selectedItems; // Start to group the changes this.diagram.widget.model.historyManager.startGroupAction(); //Makes the changes for(let i =0;i	Property: historyList.startGroupActionHTML Script public nodes= [{ offsetX: 100, offsetY: 100, width: 100, height: 100, }] public created() { let objects: (NodeModel ConnectorModel)[] = []; objects.push(this.diagram.nodes[0], this.diagram.nodes[1], this.diagram.connectors[0]); this.diagram.historyList.star

them in a single undo or redo		<pre> tGroupAction(); this.diagram.distribute('Top', objects); this.diagram.distribute('Bottom', objects); this.diagram.distribute('BottomToTop', objects); this.diagram.historyList.endGroupAction(); } }</pre>
Defines what should be happened while trying to revert a custom change	Property:historyManager.undoHTML Script <pre> public historyManager; this.historyManager: { undo: customUndoRedo, redo: customUndoRedo } public created() { let diagram = \$("#diagramContent").ejDiagram("instance"); let node = args.object; let currentState = node.employeeInfo; node.employeeInfo = args.prevState; args.prevState = currentState; }</pre>	Property:historyList.undoHTML Script <pre> public nodes = [{ offsetX: 100, offsetY: 100, width: 100, height: 100, }] this.historyManager: { undo: customUndoRedo, redo: customUndoRedo } public created() { let node: NodeModel = this.diagram.nodes[0]; node['customName'] = 'customNode'; let entry = { undoObject: node }; this.diagram.historyList.push(entry); this.diagram.historyList.undo = function(args: HistoryEntry) { args.redoObject = cloneObject(args.undoObject) as NodeModel; args.undoObject['customName'] = 'customNodeChange'; } this.diagram.historyList.redo = function(args: HistoryEntry) { let current: NodeModel = cloneObject(args.undoObject) as NodeModel; args.undoObject['customName'] = args.redoObject['customName']; args.redoObject = current; } } }</pre>
Gets the number of undo actions to be stored on the history manager. Its an	Property:historyManager.undoStackHTML Script <pre> public nodes = [{ offsetX: 100, offsetY: 100, width: 100, height: 100, }] this.historyManager: { undo: customUndoRedo, redo: customUndoRedo } public created() { let diagram = \$("#diagramContent").ejDiagram("instance"); diagram.model.historyManager.undoStack(); } }</pre>	Property:historyList.undoStackHTML Script <pre> public nodes = [{ offsetX: 100, offsetY: 100, width: 100, height: 100, }] this.historyManager: { undo: customUndoRedo, redo: customUndoRedo } public created() { this.diagram.historyList.undoStack(); }</pre>

read-only property and the collection should not be modified		
Set the current entry object	Not applicable	Property:historyList.currentEntryHTML Script public nodes = [{ offsetX: 100, offsetY: 100, width: 100, height: 100, }] this.historyManager: { undo: customUndoRedo, redo: customUndoRedo } public created() { diagram.historyList.currentEntry(); }
set the history entry can be undo	Not applicable	Property:historyList.canUndoHTML Script public nodes = [{ offsetX: 100, offsetY: 100, width: 100, height: 100, }] this.historyManager: { undo: customUndoRedo, redo: customUndoRedo } public created() { diagram.historyList.canUndo = true; }
Set the history entry can be redo	Not applicable	Property:historyList.canRedoHTML Script public nodes = [{ offsetX: 100, offsetY: 100, width: 100, height: 100, }] this.historyManager: { undo: customUndoRedo, redo: customUndoRedo } public created() { diagram.historyList.canUndo = true; }
Used to decide to stored the changes to history	Property:historyManager.canLogHTML Script public nodes = [{ offsetX: 100, offsetY: 100, width: 100, height: 100, }] public created() { let diagram = \$("#diagramContent").ejDiagram("instance"); diagram.model.historyManager.canLog(); }	Property:historyList.canLogHTML Script public nodes = [{ offsetX: 100, offsetY: 100, width: 100, height: 100, }] public created() { diagram.historyList.canLog = function (entry: HistoryEntry) { entry.cancel = true; return entry; } }

LabelRenderingMode

<!-- markdownlint-disable MD033 -->

Behavior	API in Essential JS 1	API in Essential JS 2
Defines the type of the rendering mode of label	Property: <code>labelRenderingModeHTML</code>	Not applicable

Layout

<!-- markdownlint-disable MD033 -->

Behavior	API in Essential JS 1	API in Essential JS 2
Specifies the custom bounds to arrange/align the layout	Property: <code>layout.boundsHTML</code> Script public layout; this.layout = { bounds: { x: 0, y: 0, width: 1000, height: 1000 } };	Property: <code>layout.boundsHTML</code> Script public layout:object = { bounds: new Rect(0, 0, 500, 500) };
Defines the fixed node with reference to which, the layout will be arranged and fixed node will not be repositioned	Property: <code>layout.fixedNodeHTML</code> Script public layout; this.layout = { fixedNode: "node" };	Property: <code>layout.fixedNodeHTML</code> Script public layout:object = { fixedNode: 'node' };
Customizes the orientation of trees/sub trees	Property: <code>layout.getLayoutInfoHTML</code> Script public layout; constructor() { function getLayoutInfo(diagram, node, options) { options.orientation = "vertical"; options.type = "left"; }; this.layout = { getLayoutInfo: getLayoutInfo }; }	Property: <code>layout.getLayoutInfoHTML</code> Script public layout:object = { getLayoutInfo: (node: Node, tree: TreeInfo) => { if (!tree.hasSubTree) { tree.orientation = 'vertical'; } };
Defines a method to customize the segments based on source and target nodes	Property: <code>layout.getConnectorSegmentsHTML</code> Script public layout; constructor() { function getLayoutInfo(diagram, node, options) { options.orientation = "vertical"; options.type = "left"; }; this.layout = { getConnectorSegments: getConnectorSegment }; }	Property: <code>layout.connectorSegmentsHTML</code> Script public layout:object = { connectorSegments: 'Default' };
Sets the space to be horizontally left between nodes	Property: <code>layout.horizontalSpacingHTML</code> Script public layout; constructor() { this.layout = { layout: { horizontalSpacing: 50 } }; }	Property: <code>layout.horizontalSpacingHTML</code> Script public layout:object = { horizontalSpacing: 30 };
Defines the space to be left between layout	Property: <code>layout.marginHTML</code> Script public layout; constructor() { this.layout =	Property: <code>layout.marginHTML</code> Script public layout:object = { margin: { left: 50,

bounds and layout	<pre>{ margin:{ left: 10, right: 10, top: 10, bottom: 10} }; }</pre>	<pre>top: 50, right: 0, bottom: 0 } };</pre>
Defines how to horizontally align the layout within the layout bounds	Property: <code>layout.horizontalAlignmentHTML</code> HTML Script public layout; constructor() { this.layout = { horizontalAlignment: HorizontalAlignment.Center }; }	Property: <code>layout.horizontalAlignm</code> HTML Script public layout:object = { horizontalAlignment: 'Center' };
Defines how to vertically align the layout within the layout bounds	Property: <code>layout.verticalAlignmentHTML</code> HTML Script public layout; constructor() { this.layout = { verticalAlignment: VerticalAlignment.Center }; }	Property: <code>layout.verticalAlignment</code> HTML Script public layout:object = { verticalAlignment: 'Center' };
Sets the orientation/direction to arrange the diagram elements	Property: <code>layout.orientationHTML</code> HTML Script public layout; constructor() { this.layout = { orientation: LayoutOrientations.LeftToRight }; }	Property: <code>layout.orientationHTML</code> HTML Script public layout:object = { orientation: 'TopToBottom', } };
Sets the type of the layout based on which the elements will be arranged	Property: <code>layout.typeHTML</code> HTML Script public layout; constructor() { this.layout = { type: LayoutTypes.HierarchicalTree } };	Property: <code>layout.typeHTML</code> HTML Script public layout:object = { type: 'OrganizationalChart' } };
Sets the space to be vertically left between nodes	Property: <code>layout.verticalSpacingHTML</code> HTML Script public layout; constructor() { this.layout = { verticalSpacing: 50 } };	Property: <code>layout.verticalSpacingHT</code> ML Script public layout:object = { verticalSpacing: 30 } };
Sets the value is used to define the root node of the layout	Property: <code>layout.rootHTML</code> HTML Script public layout; constructor() { this.layout = { root: 'rootNode' } };	Property: <code>layout.rootHTML</code> HTML Script public layout:object = { root: 'rootNode' } };
Defines how long edges should be, ideally. This will be the resting length for the springs	Property: <code>layout.springFactorHTML</code> HTML Script public layout; constructor() { this.layout = { springFactor: 0.442 } };	Property: <code>layout.springFactorHTML</code> HTML Script public layout:object = { type: 'SymmetricalLayout', springLength: 80, springFactor: 0.8, maxIteration: 500, } };
Defines how long edges should be, ideally. This will be the resting	Property: <code>layout.maxIterationHTML</code> HTML Script public layout; constructor() { this.layout = { maxIteration: 442 } };	Property: <code>layout.maxIterationHTM</code> L Script public layout:object = { type: 'SymmetricalLayout', springLength: 80,

length for the springs		springFactor: 0.8, maxIteration: 500, }; };
Defines how long edges should be, ideally. This will be the resting length for the springs	Property:layout.springLengthHTML Script public layout; constructor() { this.layout = { springLength: 80 } };	Property:layout.springLengthHTML Script public layout:object = { type: 'SymmetricalLayout', springLength: 80, springFactor: 0.8, maxIteration: 500, }; };
Sets how to define the connection direction (first segment direction & last segment direction)	Not applicable	Property:layout.connectionDirectionHTML Script public layout:object = { connectionDirection: 'Auto', type: 'SymmetricalLayout', springLength: 80, springFactor: 0.8, maxIteration: 500, }; };
Enables/Disables animation option when a node is expanded/collapsed	Not applicable	Property:layout.enableAnimationHTML Script public layout:object = { enableAnimation: true, orientation: 'TopToBottom', type: 'OrganizationalChart', margin: { top: 20 }, horizontalSpacing: 30, verticalSpacing: 30, }; };
Defines whether an object should be at the left/right of the mind map. Applicable only for the direct children of the root node	Not applicable	Property:layout.getBranchHTML Script public layout:object = { type: 'MindMap', }; };

Locale

<!-- markdownlint-disable MD033 -->

Behavior	API in Essential JS 1	API in Essential JS 2
Defines the current culture of diagram	Property:localeHTML js-diagram>	Property:localeHTML

Nodes

<!-- markdownlint-disable MD033 -->

Behavior	API in Essential JS 1	API in Essential JS 2
Array of JSON objects where each object represents a node	Property: <code>nodes</code> HTML Script <pre>public nodes; constructor() { this.nodes = [{ name: "node1", width: 175, height: 60, offsetX:100, offsetY:100}]; }</pre>	Property: <code>annotations.content</code> HTML Script <pre>public nodes: NodeModel[] = [{ offsetX: 250, offsetY: 250, width: 100, height: 100, }];</pre>
Defines the type of BPMN Activity. Applicable, if the node is a BPMN activity	Property: <code>nodes.activity</code> HTML Script <pre>public nodes; constructor() { this.nodes = [{ type: "bpmn", shape: BPMNShapes.Activity, activity: BPMNActivity.SubProcess, width:50, height:50 }]; }</pre>	Property: <code>nodes.shape.activity</code> HTML Script <pre>public nodes: NodeModel[] = [{ offsetX: 250, offsetY: 250, width: 100, height: 100, shape: { type: 'Bpmn', shape: 'Activity', activity: { activity: 'Task' } }, },];</pre>
To maintain additional information about nodes	Property: <code>nodes.addInfo</code> HTML Script <pre>public nodes; constructor() { let addInfo = { TooltipData: "Shares the information with the customer" }; let node1 = { name: "node1", addInfo: addInfo, offsetX:100, offsetY:100, width:50, height:50 }; let node2 = { type: "swimlane", name: "swimlane", addInfo: addInfo }; this.nodes = [node1, node2]; }</pre>	Property: <code>nodes.addInfo</code> HTML Script <pre>public nodes: NodeModel[] = [{ offsetX: 250, offsetY: 250, width: 100, height: 100, addInfo: { "borderColor": "black", "borderWidth": '1px' }, },];</pre>
Defines the additional information of a process. It is not directly related to the message flows or sequence flows of the process	Property: <code>nodes.annotation</code> HTML Script <pre>public nodes; constructor() { this.nodes = [{ name: "node1", width: 100, height:100, offsetX:50, offsetY:50, type:"bpmn", shape: "activity", annotation: { text: "This is a BPMN Activity shape", width: 100, height: 50, angle: -45, length: 150, direction: "top" } }]; }</pre>	Property: <code>nodes.shape.annotations</code> HTML Script <pre>public nodes: NodeModel[] = [{ offsetX: 250, offsetY: 250, width: 100, height: 100, shape: { type: 'Bpmn', shape: 'DataObject', dataObject: { collection: true, type: 'Input' }, annotations: [{ id: 'left', angle: 45, length: 150, text: 'Left', }] } }];</pre>
Sets the angle between the BPMN shape and the annotation	Property: <code>nodes.annotation.angle</code> HTML Script <pre>public nodes; constructor() { this.nodes = [{ name: "node1", width: 100, height:100, offsetX:50, offsetY:50, type:"bpmn", shape: "activity", annotation: { text: "This is a BPMN Activity shape", width: 100, height: 50, angle: -45 } }]; }</pre>	Property: <code>nodes.shape.annotations.angle</code> HTML Script <pre>public nodes: NodeModel[] = [{ offsetX: 250, offsetY: 250, width: 100, height: 100, shape: { type: 'Bpmn', shape: 'DataObject', dataObject: { collection: true, type: 'Input' }, annotations: [{ id: 'left', angle: 45, }] } }];</pre>

Sets the direction of the text annotation	Property: nodes.annotation.directionHTML L Script public nodes; constructor() { this.nodes = [{ name: "node1", width: 100, height:100, offsetX:50, offsetY:50, type:"bpmn", shape: "activity", annotation: { text: "This is a BPMN Activity shape", width: 100, height: 50, angle: -45, length: 150, direction: "top" } }]; }	Not applicable
Sets the height of the text annotation	Property: nodes.annotation.heightHTML Script public nodes; constructor() { this.nodes = [{ name: "node1", width: 100, height:100, offsetX:50, offsetY:50, type:"bpmn", shape: "activity", annotation: { text: "This is a BPMN Activity shape", width: 100, height: 50, } }]; }	Property: nodes.shape.annotations.heightHTML Script public nodes; NodeModel[]=[{ offsetX: 250, offsetY: 250, width: 100, height: 100, shape: { type: 'Bpmn', shape: 'DataObject', dataObject: { collection: true, type: 'Input' }, annotations: [{ id: 'left', text: 'Left', height: 50 }] } }];
Sets the distance between the BPMN shape and the annotation	Property: nodes.annotation.lengthHTML Script public nodes; constructor() { this.nodes = [{ name: "node1", width: 100, height:100, offsetX:50, offsetY:50, type:"bpmn", shape: "activity", annotation: { text: "This is a BPMN Activity shape", width: 100, height: 50, length: 150 } }]; }	Property: nodes.shape.annotations.lengthHTML Script public nodes; NodeModel[]=[{ offsetX: 250, offsetY: 250, width: 100, height: 100, shape: { type: 'Bpmn', shape: 'DataObject', dataObject: { collection: true, type: 'Input' }, annotations: [{ id: 'left', length: 150, text: 'Left', }] } }];
Defines the additional information about the flow object in a BPMN Process	Property: nodes.annotation.textHTML Script public nodes; constructor() { this.nodes = [{ name: "node1", width: 100, height:100, offsetX:50, offsetY:50, type:"bpmn", shape: "activity", annotation: { text: "This is a BPMN Activity shape" } }]; }	Property: nodes.shape.annotations.textHTML Script public nodes; NodeModel[]=[{ offsetX: 250, offsetY: 250, width: 100, height: 100, shape: { type: 'Bpmn', shape: 'DataObject', dataObject: { collection: true, type: 'Input' }, annotations: [{ text: 'Left', }] } }];
Sets the width of the text annotation	Property: nodes.annotation.widthHTML Script public nodes; constructor() { this.nodes = [{ name: "node1", width: 100, height:100, offsetX:50, offsetY:50, type:"bpmn", shape: "activity", annotation: { text: "This is a BPMN Activity shape", width: 100, height: 50 } }]; }	Property: nodes.shape.annotations.widthHTML Script public nodes; NodeModel[]=[{ offsetX: 250, offsetY: 250, width: 100, height: 100, shape: { type: 'Bpmn', shape: 'DataObject', dataObject: { collection: true, type: 'Input' }, annotations: [{ id: 'left', width: 45, text: 'Left', }] } }];

Sets the id for the annotation	Not applicable	Property: nodes.shape.annotations.id HTML Script <pre>public nodes: NodeModel[] = [{ offsetX: 250, offsetY: 250, width: 100, height: 100, shape: { type: 'Bpmn', shape: 'DataObject', dataObject: { collection: true, type: 'Input' }, annotations: [{ id: 'left', text: 'Left', }] } }];</pre>
Defines whether the group can be ungrouped or not	Property: nodes.canUngroup HTML Script <pre>public nodes; constructor() { let node1 = { name: "node1", width: 50, height:50, offsetX:50, offsetY:50, borderColor: "red" , borderDashArray: "4,2"}; let node2 = { name: "node2", width: 50, height:50, offsetX:150, offsetY:150, borderColor: "red" , borderDashArray: "4,2"}; let group = { name : "group", children:[node1, node2], canUngroup: false }; this.nodes = [group]; }</pre>	Not applicable
Array of JSON objects where each object represents a child node/connector	Property: nodes.children HTML Script <pre>public nodes; constructor() { let node1 = { name: "node1", width: 50, height:50, offsetX:50, offsetY:50, borderColor: "red" , borderDashArray: "4,2"}; let node2 = { name: "node2", width: 50, height:50, offsetX:150, offsetY:150, borderColor: "red" , borderDashArray: "4,2"}; let group = { name : "group", children:[node1, node2]}; this.nodes =[group]; }</pre>	Property: nodes.children HTML Script <pre>public nodes: NodeModel[] = [group]; public node1: NodeModel = { id: 'node1', offsetX: 250, offsetY: 250, width: 100, height: 100, }; public node2: NodeModel = { id: 'node2', offsetX: 450, offsetY: 450, width: 100, height: 100, }; public group: NodeModel = { id: 'group', children : ['node1', 'node2'] };</pre>
Sets the type of UML classifier	Property: nodes.classifier HTML Script <pre>public nodes; constructor() { this.nodes = [{ name: "Patient", offsetX: 100, offsetY: 100, borderWidth: 2, borderColor: "black", type: "UmlClassifier", classifier: ClassifierShapes.Class }]; }</pre>	Not applicable
Defines the name, attributes and methods of a Class. Applicable, if	Property: nodes.class HTML Script <pre>public nodes; constructor() { this.nodes = [{ name: "Patient", offsetX: 100, offsetY: 100, borderWidth: 2, borderColor: "black", type: "UmlClassifier", classifier:</pre>	Not applicable

the node is a Class	<code>ClassifierShapes.Class, "class": { name: "Patient", } }]; }</code>	
Sets the name of class	Property: <code>nodes.class.name</code> HTML Script <pre>public nodes; constructor() { this.nodes = [{ name: "Patient", offsetX: 100, offsetY: 100, borderWidth: 2, borderColor: "black", type: "UmlClassifier", classifier: ClassifierShapes.Class, "class": { name: "Patient", } }]; }</pre>	Not applicable
Defines the collection of attributes	Property: <code>nodes.class.attributes</code> HTML Script <pre>public nodes; constructor() { this.nodes = [{ name: "Patient", offsetX: 100, offsetY: 100, borderWidth: 2, borderColor: "black", type: "UmlClassifier", classifier: ClassifierShapes.Class, "class": { name: "Patient", attributes: [{ name: "accepted"}] } }]; }</pre>	Not applicable
Sets the name of the attribute	Property: <code>nodes.class.attributes.name</code> HTML Script <pre>public nodes; constructor() { this.nodes = [{ name: "Patient", offsetX: 100, offsetY: 100, borderWidth: 2, borderColor: "black", type: "UmlClassifier", classifier: ClassifierShapes.Class, "class": { name: "Patient", attributes: [{ name: "accepted" }]} }]; }</pre>	Not applicable
Sets the data type of attribute	Property: <code>nodes.class.attributes.type</code> HTML Script <pre>public nodes; constructor() { this.nodes = [{ name: "Patient", offsetX: 100, offsetY: 100, borderWidth: 2, borderColor: "black", type: "UmlClassifier", classifier: ClassifierShapes.Class, "class": { name: "Patient", attributes: [{ name: "accepted", type: "Date" }]} }]]; }</pre>	Not applicable
Defines the visibility of the attribute	Property: <code>nodes.class.attributes.scope</code> HTML Script <pre>public nodes; constructor() { this.nodes = [{ name: "Patient", offsetX: 100, offsetY: 100, borderWidth: 2, borderColor: "black", type: "UmlClassifier", classifier: ClassifierShapes.Class, "class":</pre>	Not applicable

	<pre>{ name: "Patient", attributes: [{ name: "accepted", type: "Date", scope:"protected" }] } }]; }</pre>	
Defines the collection of methods of a Class	Property:nodes.class.methodsHTML Script public nodes; constructor() { this.nodes =[{ name: "Patient", offsetX: 100, offsetY: 100, borderWidth: 2, borderColor: "black", type: "UmlClassifier", classifier: ClassifierShapes.Class, "class": { name: "Patient", methods: [{ name: "getHistory" }] } }]]; }	Not applicable
Sets the name of the method	Property:nodes.class.methods.nameHTML Script public nodes; constructor() { this.nodes = [{ name: "Patient", offsetX: 100, offsetY: 100, borderWidth: 2, borderColor: "black", type: "UmlClassifier", classifier: ClassifierShapes.Class, "class": { name: "Patient", methods: [{ name: "getHistory", arguments: [{name: "Date" }] }] } }]]; }	Not applicable
Defines the arguments of the method	Property:nodes.class.methods.arguments HTML Script public nodes; constructor() { this.nodes = [{ name: "Patient", offsetX: 100, offsetY: 100, borderWidth: 2, borderColor: "black", type: "UmlClassifier", classifier: ClassifierShapes.Class, "class": { name: "Patient", methods: [{ name: "getHistory", arguments: [{ name: "Date", type:"String" }] }] } }]]; }	Not applicable
Defines the name, attributes and methods of a Class. Applicable, if the node is a Class	Property:nodes.class.methods.arguments .nameHTML Script public nodes; constructor() { this.nodes = [{ name: "Patient", offsetX: 100, offsetY: 100, borderWidth: 2, borderColor: "black", type: "UmlClassifier", classifier: ClassifierShapes.Class, "class": { name: "Patient", methods: [{ name: "getHistory", arguments: [{ name: "Date" }], type: "History" }] } }]]; }	Not applicable
Sets the type of the argument	Property:nodes.class.methods.arguments .typeHTML Script public nodes;	Not applicable

	<pre> constructor() { this.nodes = [{ name: "Patient", offsetX: 100, offsetY: 100, borderWidth: 2, borderColor: "black", type: "UmlClassifier", classifier: ClassifierShapes.Class, "class": { name: "Patient", methods: [{ name: "getHistory", arguments: [{ name: "Date" }], type: "History" }] } }]}; </pre>	
Sets the return type of the method	Property: nodes.class.methods.typeHTML <pre> Script public nodes; constructor() { this.nodes = [{ name: "Patient", offsetX: 100, offsetY: 100, borderWidth: 2, borderColor: "black", type: "UmlClassifier", classifier: ClassifierShapes.Class, "class": { name: "Patient", methods: [{ name: "getHistory", arguments: [{name: "Date" }], type: "History" }] } }]}; </pre>	Not applicable
Sets the visibility of the method	Property: nodes.class.methods.scopeHTML <pre> L Script public nodes; constructor() { this.nodes = [{ name: "Patient", offsetX: 100, offsetY: 100, borderWidth: 2, borderColor: "black", type: "UmlClassifier", classifier: ClassifierShapes.Class, "class": { name: "Patient", methods: [{ name: "getHistory", arguments: [{name: "Date" }], type: "History", scope: "protected" }] } }]}; </pre>	Not applicable
Defines the state of the node is collapsed	Property: nodes.collapseIconHTML <pre> Script public nodes: NodeModel[] = [{ name: "node", width: 100, height: 100, offsetX: 100, offsetY: 100, collapseIcon: { shape: "ArrowUp", width: 10, height: 10 }, expandIcon: { height: 10, width: 10, shape: "ArrowDown" } }] </pre>	Property: nodes.collapseIconHTML <pre> Script public nodes: NodeModel[] = [{ offsetX: 100, offsetY: 100, width: 100, height: 100, expandIcon: { height: 20, width: 20, shape: "ArrowDown", fill: 'red' }, collapseIcon: { height: 20, width: 20, shape: "ArrowUp" } }]}; </pre>
Sets the border color for collapse icon of node	Property: nodes.collapseIcon.borderColorHTML <pre> Script public nodes: NodeModel[] = [{ name: "node", width: 100, height: 100, offsetX: 100, offsetY: 100, collapseIcon: { shape: "ArrowUp", width: 10, height: 10, borderColor: "red" }, expandIcon: { height: 10, width: </pre>	Property: nodes.collapseIcon.borderColorHTML <pre> Script public nodes: NodeModel[] = [{ offsetX: 100, offsetY: 100, width: 100, height: 100, expandIcon: { height: 20, width: 20, shape: "ArrowDown", borderColor: 'red' </pre>

	<pre>10, shape: "ArrowDown", borderColor: "red" } }]</pre>	<pre>}, collapseIcon: { height: 20, width: 20, shape: "ArrowUp", borderColor: 'red' } }];</pre>
Sets the border width for collapse icon of node	Property: <code>nodes.collapseIcon.borderWidth</code> HTML Script public nodes: <pre>NodeModel[] = [{ name: "node", width: 100, height: 100, offsetX: 100, offsetY: 100, collapseIcon: { shape:"ArrowUp", width:10, height:10, borderWidth: "2" }, expandIcon: { height: 10, width: 10, shape: "ArrowDown", borderWidth: "2" } }]]</pre>	Property: <code>nodes.collapseIcon.borderWidth</code> HTML Script public nodes: <pre>NodeModel [] = [{ offsetX: 100, offsetY: 100, width: 100, height: 100, expandIcon: { height: 20, width: 20, shape: "ArrowDown", borderWidth: '2' }, collapseIcon: { height: 20, width: 20, shape: "ArrowUp", borderWidth: '2' } }]];</pre>
Sets the fill color for collapse icon of node	Property: <code>nodes.collapseIcon.fillColor</code> HTML Script public nodes: <pre>NodeModel[] = [{ name: "node", width: 100, height: 100, offsetX: 100, offsetY: 100, collapseIcon: { shape:"ArrowUp", width:10, height:10, fillColor: "green" }, expandIcon: { height: 10, width: 10, shape: "ArrowDown", fillColor: "green" } }]]</pre>	Property: <code>nodes.collapseIcon.fill</code> HTML Script public nodes: <pre>NodeModel[] = [{ offsetX: 100, offsetY: 100, width: 100, height: 100, expandIcon: { height: 20, width: 20, shape: "ArrowDown", fill: 'red' }, collapseIcon: { height: 20, width: 20, shape: "ArrowUp", fill: 'red' } }]];</pre>
Defines the height for collapse icon of node	Property: <code>nodes.collapseIcon.height</code> HTML Script public nodes: <pre>NodeModel[] = [{ name: "node", width: 100, height: 100, offsetX: 100, offsetY: 100, collapseIcon: { shape:"ArrowUp", width:10, height:10 }, expandIcon: { height: 10, width: 10, shape: "ArrowDown" } }]]</pre>	Property: <code>nodes.collapseIcon.height</code> HTML Script public nodes: <pre>NodeModel[] = [{ offsetX: 100, offsetY: 100, width: 100, height: 100, expandIcon: { height: 20, width: 20, shape: "ArrowDown", fill: 'red' }, collapseIcon: { height: 20, width: 20, shape: "ArrowUp" } }]];</pre>
Sets the horizontal alignment of the icon	Property: <code>nodes.collapseIcon.horizontalAlignment</code> HTML Script public nodes: <pre>NodeModel[] = [{ name: "node", width: 100, height: 100, offsetX: 100, offsetY: 100, collapseIcon: { shape:"ArrowUp", width:10, height:10, horizontalAlignment:HorizontalAlignment.Left }, expandIcon: { height: 10, width: 10, shape: "ArrowDown", horizontalAlignment:HorizontalAlignment.Left } }]]</pre>	Property: <code>nodes.collapseIcon.horizontalAlignment</code> HTML Script public nodes: <pre>NodeModel[] = [{ offsetX: 100, offsetY: 100, width: 100, height: 100, expandIcon: { height: 20, width: 20, shape: "ArrowDown", horizontalAlignment:'Center' }, collapseIcon: { height: 20, width: 20, shape: "ArrowUp", horizontalAlignment:'Center' } }]]</pre>
To set the margin for the	Property: <code>nodes.collapseIcon.margin</code> HTML Script public nodes: <pre>NodeModel[] = [{ name: "node", width: 100, height: 100, offsetX: 100,</pre>	Property: <code>nodes.collapseIcon.margin</code> HTML Script public nodes: <pre>NodeModel[] = [{ offsetX: 100, offsetY: 100, width: 100,</pre>

collapse icon of node	offsetY: 100, collapseIcon: { shape:"ArrowUp", width:10, height:10, margin:{ left: 5 } }, expandIcon: { height: 10, width: 10, shape: "ArrowDown", margin:{ left: 5 } } } }	height: 100, , expandIcon: { height: 20, width: 20, shape: "ArrowDown", fill: 'red', margin:{ left: 5 } }, collapseIcon: { height: 20, width: 20, shape: "ArrowUp", margin:{ left: 5 } } } }
Sets the fraction/ratio(relative to node) that defines the position of the icon	Property:nodes.collapseIcon.offsetHTML Script public nodes: NodeModel[] = [{ name: "node", width: 100, height: 100, offsetX: 100, offsetY: 100, collapseIcon: { shape:"ArrowUp", width:10, height:10, offset:Point(0,0.5) }, expandIcon: { height: 10, width: 10, shape: "ArrowDown", offset:Point(0,0.5) } } }	Property:nodes.collapseIcon.offsetHTML Script public nodes: NodeModel[] = [{ offsetX: 100, offsetY: 100, width: 100, height: 100, expandIcon: { height: 20, width: 20, shape: "ArrowDown", offset: { x: 0, y: 0.5 } }, collapseIcon: { height: 20, width: 20, shape: "ArrowUp", offset: { x: 0, y: 0.5 } } } }];
Defines the shape of the collapsed state of the node	Property:nodes.collapseIcon.shapeHTML Script public nodes: NodeModel[] = [{ name: "node", width: 100, height: 100, offsetX: 100, offsetY: 100, collapseIcon: { shape:"ArrowUp", width:10, height:10 }, expandIcon: { height: 10, width: 10, shape: "ArrowDown" } } }	Property:nodes.collapseIcon.shapeHTML Script public nodes: NodeModel[] = [{ offsetX: 100, offsetY: 100, width: 100, height: 100, , expandIcon: { height: 20, width: 20, shape: "ArrowDown", fill: 'red' }, collapseIcon: { height: 20, width: 20, shape: "ArrowUp" } } }];
Sets the vertical alignment of the icon	Property:nodes.collapseIcon.verticalAlignment HTML Script public nodes: NodeModel[] = [{ name: "node", width: 100, height: 100, offsetX: 100, offsetY: 100, collapseIcon: { shape:"ArrowUp", width:10, height:10, verticalAlignment:VerticalAlignment.Top }, expandIcon: { height: 10, width: 10, shape: "ArrowDown", verticalAlignment:VerticalAlignment.Top } } }	Property:nodes.collapseIcon.verticalAlignment HTML Script public nodes: NodeModel[] = [{ offsetX: 100, offsetY: 100, width: 100, height: 100, , expandIcon: { height: 20, width: 20, shape: "ArrowDown", verticalAlignment: 'Center' }, collapseIcon: { height: 20, width: 20, shape: "ArrowUp", verticalAlignment: 'Center' } } }
Defines the custom content of the icon	Not applicable	Property:nodes.collapseIcon.contentHTML Script public nodes: NodeModel[] = [{ offsetX: 100, offsetY: 100, width: 100, height: 100, expandIcon: { height: 20, width: 20, shape: "Template", content: '' + '' }, collapseIcon: { height: 20, width: 20, shape: "ArrowUp" } } }

Defines the geometry of a path	Not applicable	Property: <code>nodes.collapseIcon.pathData</code> HTML Script public nodes: NodeModel[] = [{ offsetX: 100, offsetY: 100, width: 100, height: 100, expandIcon: { height: 20, width: 20, shape: "Path", pathData: "M0,0 L0,100" }, collapseIcon: { height: 20, width: 20, shape: "Path", pathData: "M0,0 L0,100" } }] }
Defines the corner radius of the icon border	Not applicable	Property: <code>nodes.collapseIcon.cornerRadius</code> HTML Script public nodes: NodeModel[] = [{ offsetX: 100, offsetY: 100, width: 100, height: 100, , expandIcon: { height: 20, width: 20, shape: "ArrowDown", cornerRadius: 3 }, collapseIcon: { height: 20, width: 20, shape: "ArrowUp", cornerRadius: 3 } }] }
Defines the space that the icon has to be moved from the icon border	Not applicable	Property: <code>nodes.collapseIcon.padding</code> HTML Script public nodes: NodeModel[] = [{ offsetX: 100, offsetY: 100, width: 100, height: 100, expandIcon: { height: 20, width: 20, shape: "ArrowDown", padding: { left: 50 } }, collapseIcon: { height: 20, width: 20, shape: "ArrowUp", padding: { left: 50 } } }] }
Defines the distance to be left between a node and its connections(In coming and outgoing connections)	Property: <code>nodes.connectorPadding</code> HTML Script public nodes: NodeModel[] = [{ name: "node", width: 100, height: 100, offsetX: 100, offsetY: 100, connectorPadding: 5 }] }	Not applicable
Enables or disables the default behaviors of the node	Property: <code>nodes.constraints</code> HTML Script public NodeConstraints = NodeConstraints; public nodes: NodeModel[] = [{ name: "node", width: 100, height: 100, offsetX: 100, offsetY: 100, constraints: NodeConstraints.Default & ~NodeConstraints.Select }] }	Property: <code>nodes.constraints</code> HTML Script public nodes: NodeModel[] = [{ offsetX: 100, offsetY: 100, width: 100, height: 100, constraints: NodeConstraints.Default NodeConstraints.Select }] }
Defines how the child	Property: <code>nodes.container</code> HTML Script public nodes; constructor() { let	Not applicable

objects need to be arranged(Either in any predefined manner or automatically). Applicable, if the node is a group	<pre> node1 = { name: "node1", width: 50, height: 50, borderColor: "red", borderDashArray: "4,2" }; let node2 = { name: "node2", width: 50, height: 50, borderColor: "red", borderDashArray: "4,2" }; let group = { name: "group", children: [node1, node2], container: { type: "stack" }, offsetX: 200, offsetY: 100 }; this.nodes = [group]; </pre>	
Defines the orientation of the container. Applicable, if the group is a container	Property: <code>nodes.container.orientation</code> HTML Script <pre> public nodes; constructor() { let node1 = { name: "node1", width: 50, height: 50, borderColor: "red", borderDashArray: "4,2" }; let node2 = { name: "node2", width: 50, height: 50, borderColor: "red", borderDashArray: "4,2" }; let group = { name: "group", children: [node1, node2], container: { type: "stack", orientation: "horizontal" }, offsetX: 200, offsetY: 100 }; this.nodes = [group]; } </pre>	Not applicable
Sets the type of the container. Applicable if the group is a container.	Property: <code>nodes.container.type</code> HTML Script <pre> public nodes; constructor() { let node1 = { name: "node1", width: 50, height: 50, borderColor: "red", borderDashArray: "4,2" }; let node2 = { name: "node2", width: 50, height: 50, borderColor: "red", borderDashArray: "4,2" }; let group = { name: "group", children: [node1, node2], container: { type: ContainerType.Stack }, offsetX: 200, offsetY: 100 }; this.nodes = [group]; } </pre>	Not applicable
Defines the corner radius of rectangular shapes	Property: <code>nodes.cornerRadius</code> HTML Script <pre> public nodes: NodeModel[] = [{ name: "node", width: 100, height: 100, offsetX: 100, offsetY: 100, type: "basic", shape: "rectangle", cornerRadius: 5 }] </pre>	Not applicable
This property allows you to customize nodes	Property: <code>nodes.cssClass</code> HTML Script <pre> public nodes: NodeModel[] = [{ name: "node", width: 100, height: </pre>	Not applicable

appearance using user-defined CSS	100, offsetX: 100, offsetY: 100, cssClass: "hoverNode" }}	
Defines the BPMN data object	Property:nodes.data.typeHTML Script public nodes: NodeModel[] = [{ name: "dataObject", type: "bpmn", shape: BPMNShapes.DataObject, data: { type: BPMNDataObjects.Input }, width: 50, height: 50, offsetX: 100, offsetY: 100 }]	Property:nodes.shape.dataObject.type HTML Script public nodes: NodeModel[] = [{ id: 'node', width: 100, height: 100, offsetX: 100, offsetY: 100, shape: { type: 'Bpmn', shape: 'DataObject', dataObject: { collection: false, type: 'Input' } } }]
Defines whether the BPMN data object is a collection or not	Property:nodes.data.collectionHTML Script public nodes: NodeModel[] = [{ name: "dataObject", type: "bpmn", shape: BPMNShapes.DataObject, data: { type: BPMNDataObjects.Input, collection: false }, width: 50, height: 50, offsetX: 100, offsetY: 100 }]	Property:nodes.shape.dataObject.colle ctionHTML Script public nodes: NodeModel[] = [{ id: 'node', width: 100, height: 100, offsetX: 100, offsetY: 100, shape: { type: 'Bpmn', shape: 'DataObject', dataObject: { collection: false, type: 'Input' } } }]
Defines an Enumeration in a UML Class Diagram	Property:nodes.enumerationHTML Script public nodes: NodeModel[] = [{ name: "Enums", offsetX: 100, offsetY: 100, borderWidth: 2, borderColor: "black", type: "UmlClassifier", classifier: ClassifierShapes Enumeration, enumeration: { name: "AccountType", } }]	Not applicable
Sets the name of the Enumeration	Property:nodes.enumeration.nameHTML Script public nodes: NodeModel[] = [{ name: "Enums", offsetX: 100, offsetY: 100, borderWidth: 2, borderColor: "black", type: "UmlClassifier", classifier: ClassifierShapes Enumeration, enumeration: { name: "AccountType", } }]	Not applicable
Defines the collection of enumeration members	Property:nodes.enumeration.membersHT ML Script public nodes: NodeModel[] = [{ name: "Enums", offsetX: 100, offsetY: 100, borderWidth: 2, borderColor: "black", type: "UmlClassifier", classifier: ClassifierShapes Enumeration, enumeration: { name:	Not applicable

	"AccountType", members: [{ name: "CheckingAccount"}] } }]	
Sets the name of the enumeration member	Property:nodes.enumeration.members.nameHTML Script public nodes: NodeModel[] = [{ name: "Enums", offsetX: 100, offsetY: 100, borderWidth: 2, borderColor: "black", type: "UmlClassifier", classifier: ClassifierShapes Enumeration, enumeration: { name: "AccountType", members: [{ name: "CheckingAccount"}] } }]	Not applicable
Sets the type of the BPMN Events. Applicable, if the node is a BPMN event	Property:nodes.eventHTML Script public nodes: NodeModel[] = [{ name: "nodeEvent", type: "bpmn", shape: "event", event: BPMNEvents.Intermediate, width: 50, height: 50 }]	Property:nodes.shape.eventHTML Script public nodes: NodeModel = [{ offsetX: 100, offsetY: 100, width: 100, height: 100, shape: { type: 'Bpmn', shape: 'Event', event: { event: 'Start', trigger: 'None' } } }]
Defines the type of the trigger	Property:nodes.event.triggerHTML Script public nodes: NodeModel[] = [{ name: "nodeEvent", type: "bpmn", shape: BPMNShapes.Event, trigger: BPMNTriggers.None, width: 50, height: 50 }]	Property:nodes.shape.event.triggerHTML Script public nodes: NodeModel[] = [{ offsetX: 100, offsetY: 100, width: 100, height: 100, shape: { type: 'Bpmn', shape: 'Event', event: { event: 'Start', trigger: 'None' } } }]
Defines whether the node can be automatically arranged using layout or not	Property:nodes.excludeFromLayoutHTML Script public nodes; public layout; constructor() { let node1 = { name: "node1", width: 50, height: 50, offsetX: 50, offsetY: 50, excludeFromLayout: true }; let node2 = { name: "node2", width: 50, height: 50 }; let node3 = { name: "node3", width: 50, height: 50 }; this.nodes = [node1, node2, node3]; this.layout= { type: "HierarchicalTree" } }	Property:nodes.excludeFromLayoutHTML Script public nodes: NodeModel[] = [{ id: 'node', offsetX: 100, offsetY: 100, width: 100, height: 100, excludeFromLayout: true, }, { id: 'node1', width: 70, height: 70, annotations: [{ content: 'node1' }] }, { id: 'node2', width: 70, height: 70, annotations: [{ content: 'node2' }] };] public layout:object = { type: 'RadialTree', }
Defines the fill color of the node	Property:nodes.fillColorHTML Script public nodes; constructor() { let node1 = { name: "node1", width: 50, height: 50, offsetX: 50, offsetY: 50, fillColor:"red"}; this.nodes = [node1]; }	Property:nodes.style.fillHTML Script public nodes: NodeModel[] = [{ id: 'node', offsetX: 100, offsetY: 100, width: 100, height: 100, style: { fill: 'red' } },]

Sets the type of the BPMN Gateway. Applicable, if the node is a BPMN gateway	Property: nodes.gateway HTML Script <pre>public nodes; constructor() { this.nodes = [{ name: "node1", width: 50, height: 50, offsetX: 50, offsetY: 50, type: "bpmn", shape: "gateway" , gateway: BPMNGateways.Exclusive }]; }</pre>	Property: nodes.shape.gateway HTML Script <pre>public nodes: NodeModel[] = [{ id: 'node', width: 100, height: 100, offsetX: 100, offsetY: 100, shape: { type: 'Bpmn', shape: 'Gateway', gateway: { type: 'Exclusive' } } }]</pre>
Paints the node with linear color transitions	Property: nodes.gradient.type HTML Script <pre>public nodes; constructor() { let gradient = { LinearGradient: { type: "linear", x1: 0, x2: 50, y1: 0, y2: 50, stops: [{ color: "white", offset: 0 }, { color: "red", offset: 50 }] } }; let node1 = { name: "node1", width: 50, height: 50, offsetX: 50, offsetY: 50, gradient: gradient }; this.nodes = [node1]; }</pre>	Property: nodes.style.gradient.type HTML Script <pre>public nodes: NodeModel[] = [{ id: 'node', width: 100, height: 100, offsetX: 100, offsetY: 100, style: { gradient: gradient1 } }]; public stopsCol: StopModel[] = []; public stops1: StopModel = { color: 'white', offset: 0 }; stopsCol.push(stops1); public stops2: StopModel = { color: 'red', offset: 50 }; stopsCol.push(stops2); public gradient1: LinearGradientModel = { x1: 0, x2: 50, y1: 0, y2: 50, stops: stopsCol, type: 'Linear' };</pre>
Defines the x1 value of linear gradient	Property: nodes.gradient.LinearGradient.x1 HTML Script <pre>public nodes; constructor() { let gradient = { type: "linear", x1: 0, x2: 50, y1: 0, y2: 50, stops: [{ color: "white", offset: 0 }, { color: "red", offset: 50 }] }; let node1 = { name: "node1", width: 50, height: 50, offsetX: 50, offsetY: 50, gradient : gradient }; this.nodes = [node1]; }</pre>	Property: nodes.style.gradient.LinearGradient.x1 HTML Script <pre>public nodes: NodeModel[] = [{ id: 'node', width: 100, height: 100, offsetX: 100, offsetY: 100, style: { gradient: gradient1 } }]; public stopsCol: StopModel[] = []; public stops1: StopModel = { color: 'white', offset: 0 }; stopsCol.push(stops1); public stops2: StopModel = { color: 'red', offset: 50 }; stopsCol.push(stops2); public gradient1: LinearGradientModel = { x1: 0, x2: 50, y1: 0, y2: 50, stops: stopsCol, type: 'Linear' };</pre>
Defines the x2 value of linear gradient	Property: nodes.gradient.LinearGradient.x2 HTML Script <pre>public nodes; constructor() { let gradient = { LinearGradient: { type: "linear", x1: 0, x2: 50, y1: 0, y2: 50, stops: [{ color: "white", offset: 0 }, { color: "red", offset: 50 }] } }; let node1 = { name: "node1", width: 50, height:</pre>	Property: nodes.style.gradient.LinearGradient.x2 HTML Script <pre>public nodes: NodeModel[] = [{ id: 'node', width: 100, height: 100, offsetX: 100, offsetY: 100, style: { gradient: gradient1 } }]; public stopsCol: StopModel[] = []; public stops1: StopModel = { color: 'white', offset: 0 };</pre>

	50, offsetX: 50, offsetY: 50, gradient : gradient }; this.nodes = [node1]; }	}; stopsCol.push(stops1); public stops2: StopModel = { color: 'red', offset: 50 }; stopsCol.push(stops2); public gradient1: LinearGradientModel = { x1: 0, x2: 50, y1: 0, y2: 50, stops: stopsCol, type: 'Linear' };
Defines the y1 value of linear gradient	Property:nodes.gradient.LinearGradient.y1HTML Script public nodes; constructor() { let gradient = { LinearGradient:{ type: "linear", x1: 0, x2: 50, y1: 0, y2: 50, stops: [{ color: "white", offset: 0}, { color: "red", offset: 50}] } }; let node1 = { name: "node1", width: 50, height: 50, offsetX: 50, offsetY: 50, gradient : gradient }; this.nodes = [node1]; }	Property:nodes.style.gradient.LinearGradient.y1HTML Script public nodes: NodeModel[] = [{ id: 'node', width: 100, height: 100, offsetX: 100, offsetY: 100, style: { gradient: gradient1 } }] public stopsCol: StopModel[] = []; public stops1: StopModel = { color: 'white', offset: 0 }; stopsCol.push(stops1); public stops2: StopModel = { color: 'red', offset: 50 }; stopsCol.push(stops2); public gradient1: LinearGradientModel = { x1: 0, x2: 50, y1: 0, y2: 50, stops: stopsCol, type: 'Linear' };
Defines the y2 value of linear gradient	Property:nodes.gradient.LinearGradient.y2HTML Script public nodes; constructor() { let gradient = { LinearGradient:{ type: "linear", x1: 0, x2: 50, y1: 0, y2: 50, stops: [{ color: "white", offset: 0}, { color: "red", offset: 50}] } }; let node1 = { name: "node1", width: 50, height: 50, offsetX: 50, offsetY: 50, gradient : gradient }; this.nodes = [node1]; }	Property:nodes.style.gradient.LinearGradient.y2HTML Script public nodes: NodeModel[] = [{ id: 'node', width: 100, height: 100, offsetX: 100, offsetY: 100, style: { gradient: gradient1 } }] public stopsCol: StopModel[] = []; public stops1: StopModel = { color: 'white', offset: 0 }; stopsCol.push(stops1); public stops2: StopModel = { color: 'red', offset: 50 }; stopsCol.push(stops2); public gradient1: LinearGradientModel = { x1: 0, x2: 50, y1: 0, y2: 50, stops: stopsCol, type: 'Linear' };
Defines the type of gradient	Property:nodes.gradient.RadialGradient.typeHTML Script public nodes; constructor() { let node = { name: "node", width: 50, height: 50, offsetX: 100, offsetY: 100, gradient: { RadialGradient:{ type: "radial", fx: 50, fy: 50, cx: 50, cy: 50, stops: [{ color: "white", offset: 0 }, { color: "red", offset: 50 }] } }; let node = { name: "node", width: 100, height: 100, gradient: { type: "radial", fx: 50, fy: 50, cx: 50, cy: 50, stops: [{ color: "white", offset: 0 }, { color: "red", offset: 50 }] } }; this.nodes = [node]; }	Property:nodes.style.gradient.typeHTML Script public stops: StopModel[] = [{ color: 'white', offset: 0 }, { color: 'red', offset: 50 }]; public gradient: RadialGradientModel = { cx: 50, cy: 50, fx: 50, fy: 50, stops: stops, type: 'Radial' }; public nodes: NodeModel[] = [{ id: 'node', width: 100, height: 100,

	<pre>"red", offset: 100 }} } } };; this.nodes = [node1]; }</pre>	<pre>offsetX: 100, offsetY: 100, style: { gradient: gradient } }]</pre>
Defines the position of the outermost circle	<pre>Property:nodes.gradient.RadialGradient.cx HTML Script public nodes; constructor() { let node = { name: "node", width: 50, height: 50, offsetX: 100, offsetY: 100, gradient: { RadialGradient:{ type: "radial", fx: 50, fy: 50, cx: 50, cy: 50, stops: [{ color: "white", offset: 0 }, { color: "red", offset: 100 }} } } };; this.nodes = [node1]; }</pre>	<pre>Property:nodes.style.RadialGradient.cx HTML Script public stops: StopModel[] = [{ color: 'white', offset: 0 }, { color: 'red', offset: 50 }]; public gradient: RadialGradientModel = { cx: 50, cy: 50, fx: 50, fy: 50, stops: stops, type: 'Radial' }; public nodes: NodeModel[] = [{ id: 'node', width: 100, height: 100, offsetX: 100, offsetY: 100, style: { gradient: gradient } }]</pre>
Defines the outer most circle of the radial gradient	<pre>Property:nodes.gradient.RadialGradient.cy HTML Script public nodes; constructor() { let node = { name: "node", width: 50, height: 50, offsetX: 100, offsetY: 100, gradient: { RadialGradient:{ type: "radial", fx: 50, fy: 50, cx: 50, cy: 50, stops: [{ color: "white", offset: 0 }, { color: "red", offset: 100 }} } } };; this.nodes = [node1]; }</pre>	<pre>Property:nodes.style.RadialGradient.cy HTML Script public stops: StopModel[] = [{ color: 'white', offset: 0 }, { color: 'red', offset: 50 }]; public gradient: RadialGradientModel = { cx: 50, cy: 50, fx: 50, fy: 50, stops: stops, type: 'Radial' }; public nodes: NodeModel[] = [{ id: 'node', width: 100, height: 100, offsetX: 100, offsetY: 100, style: { gradient: gradient } }]</pre>
Defines the innermost circle of the radial gradient	<pre>Property:nodes.gradient.RadialGradient.fx HTML Script public nodes; constructor() { this.nodes = [{ name: "node", width: 50, height: 50, offsetX: 100, offsetY: 100, gradient: { RadialGradient:{ type: "radial", fx: 50, fy: 50, cx: 50, cy: 50, stops: [{ color: "white", offset: 0 }, { color: "red", offset: 100 }} } } }]; }</pre>	<pre>Property:nodes.style.RadialGradient.fx HTML Script public stops: StopModel[] = [{ color: 'white', offset: 0 }, { color: 'red', offset: 50 }]; public gradient: RadialGradientModel = { cx: 50, cy: 50, fx: 50, fy: 50, stops: stops, type: 'Radial' }; public nodes: NodeModel[] = [{ id: 'node', width: 100, height: 100, offsetX: 100, offsetY: 100, style: { gradient: gradient } }]</pre>
Defines the innermost circle of the radial gradient	<pre>Property:nodes.gradient.RadialGradient.fy HTML Script public nodes; constructor() { this.nodes = [{ name: "node", width: 50, height: 50, offsetX: 100, offsetY: 100, gradient: { RadialGradient:{</pre>	<pre>Property:nodes.style.RadialGradient.fy HTML Script public stops: StopModel[] = [{ color: 'white', offset: 0 }, { color: 'red', offset: 50 }]; public gradient: RadialGradientModel = { cx: 50, cy: 50, fx: 50, fy:</pre>

	<pre>type: "radial", fx: 50, fy: 50, cx: 50, cy: 50, stops: [{ color: "white", offset: 0 }, { color: "red", offset: 100 }] } }]]; }</pre>	<pre>50, stops: stops, type: 'Radial' }; public nodes: NodeModel[] = [{ id: 'node', width: 100, height: 100, offsetX: 100, offsetY: 100, style: { gradient: gradient } }]</pre>
Defines the different colors and the region of color transitions	<pre>Property:nodes.gradient.RadialGradient.stopsHTML Script public nodes; constructor() { this.nodes = [{ name: "node", width: 50, height: 50, offsetX: 100, offsetY: 100, gradient: { RadialGradient:{ type: "radial", fx: 50, fy: 50, cx: 50, cy: 50, stops: [{ color: "white", offset: 0 }, { color: "red", offset: 100 }] } } }]]; }</pre>	<pre>Property:nodes.style.RadialGradient.stopsHTML Script public stops: StopModel[] = [{ color: 'white', offset: 0 }, { color: 'red', offset: 50 }]; public gradient: RadialGradientModel = { cx: 50, cy: 50, fx: 50, fy: 50, stops: stops, type: 'Radial' }; public nodes: NodeModel[] = [{ id: 'node', width: 100, height: 100, offsetX: 100, offsetY: 100, style: { gradient: gradient } }]</pre>
Sets the color to be filled over the specified region	<pre>Property:nodes.gradient.stops.colorHTML Script public nodes; constructor() { this.nodes = [{ name: "node", width: 50, height: 50, offsetX: 100, offsetY: 100, gradient: { RadialGradient:{ type: "radial", fx: 50, fy: 50, cx: 50, cy: 50, stops: [{ color: "white", offset: 0 }, { color: "red", offset: 100 }] } } }]]; }</pre>	<pre>Property:nodes.style.gradient.stops.colorHTML Script public stops: StopModel[] = [{ color: 'white', offset: 0 }, { color: 'red', offset: 50 }]; public gradient: RadialGradientModel = { cx: 50, cy: 50, fx: 50, fy: 50, stops: stops, type: 'Radial' }; public nodes: NodeModel[] = [{ id: 'node', width: 100, height: 100, offsetX: 100, offsetY: 100, style: { gradient: gradient } }]</pre>
Sets the position where the previous color transition ends and a new color transition starts	<pre>Property:nodes.gradient.stops.offsetHTML Script public nodes; constructor() { this.nodes = [{ name: "node", width: 50, height: 50, offsetX: 100, offsetY: 100, gradient: { RadialGradient:{ type: "radial", fx: 50, fy: 50, cx: 50, cy: 50, stops: [{ color: "white", offset: 0 }, { color: "red", offset: 100 }] } } }]]; }</pre>	<pre>Property:nodes.style.gradient.stops.offsetHTML Script public stops: StopModel[] = [{ color: 'white', offset: 0 }, { color: 'red', offset: 50 }]; public gradient: RadialGradientModel = { cx: 50, cy: 50, fx: 50, fy: 50, stops: stops, type: 'Radial' }; public nodes: NodeModel[] = [{ id: 'node', width: 100, height: 100, offsetX: 100, offsetY: 100, style: { gradient: gradient } }]</pre>
Describes the transparency	<pre>Property:nodes.gradient.stops.opacityHTML Script public nodes;</pre>	<pre>Property:nodes.style.gradient.stops.opacityHTML Script public stops:</pre>

level of the region	<pre> constructor() { this.nodes = [{ name: "node", width: 50, height: 50, offsetX: 100, offsetY: 100, gradient: { RadialGradient:{ type: "radial", fx: 50, fy: 50, cx: 50, cy: 50, stops: [{ color: "white", offset: 0 }, { color: "red", offset: 100, opacity: 0.5 }] } } }]}; } </pre>	<pre> StopModel[] = [{ color: 'white', offset: 0 }, { color: 'red', offset: 50 , opacity: 0.5}]; public gradient: RadialGradientModel = { cx: 50, cy: 50, fx: 50, fy: 50, stops: stops, type: 'Radial' }; public nodes: NodeModel[] = [{ id: 'node', width: 100, height: 100, offsetX: 100, offsetY: 100, style: { gradient: gradient } }] </pre>
Defines the header of a swimlane/lane	Property: nodes.header HTML Script <pre> public nodes; constructor() { this.nodes = [{ type: "swimlane", name: "swimlane", header: { text: "Swimlane", fontSize: 12, bold: true } }]; } </pre>	Not applicable
Defines the height of the node	Property: nodes.height HTML Script <pre> public nodes: NodeModel[] = [{ name: "node", width: 100, height: 100, offsetX: 100, offsetY: 100, }] </pre>	Property: nodes.height HTML Script <pre> public nodes: NodeModel[] = [{ offsetX: 100, offsetY: 100, width: 100, height: 100, }] </pre>
Sets the horizontal alignment of the node. Applicable, if the parent of the node is a container	Property: nodes.horizontalAlign HTML Script <pre> public nodes; constructor() { let node1 = { name: "node1", width: 50, height: 50 }; let node2 = { name: "node2", width: 50, height: 50, horizontalAlign: HorizontalAlignment.Right }; let group = { name: "group", children: [node1, node2], container: { type: "canvas" }, offsetX: 200, offsetY: 100, minWidth: 200, minHeight: 200, fillColor: "red" }; this.nodes = [group]; } </pre>	Not applicable
A read only collection of the incoming connectors/edges of the node	Property: nodes.inEdges <pre> let node = diagram.selectionList[0]; for(let i = 0; i < node.inEdges.length; i++){ console.log(node.inEdges[i]); } </pre>	Property: nodes.height HTML Script <pre> public nodes: NodeModel[] = [{ offsetX: 100, offsetY: 100, width: 100, height: 100, }] </pre>
Defines an interface in a UML interface Diagram	Property: nodes.interface HTML Script <pre> public nodes; constructor() { this.nodes = [{ name: "Patient", offsetX: 100, offsetY: 100, borderWidth: 2, borderColor: "black", type: "UmlClassifier", classifier: ClassifierShapes.interface }]; } </pre>	Not applicable

Defines the name, attributes and methods of a Interface. Applicable, if the node is a Interface	Property: nodes.interface.nameHTML Script public nodes; constructor() { this.nodes = [{ name: "Patient", offsetX: 100, offsetY: 100, borderWidth: 2, borderColor: "black", type: "UmlClassifier", classifier: ClassifierShapes.interface, "interface": { name: "Patient", } }] }; }	Not applicable
Defines the collection of attributes	Property: nodes.interface.attributesHTML Script public nodes; constructor() { this.nodes = [{ name: "Patient", offsetX: 100, offsetY: 100, borderWidth: 2, borderColor: "black", type: "UmlClassifier", classifier: ClassifierShapes.interface, "interface": { name: "Patient", attributes: [{ name: "accepted" }] } }] }; }	Not applicable
Sets the name of the attribute	Property: nodes.interface.attributes.nameHTML Script public nodes; constructor() { this.nodes = [{ name: "Patient", offsetX: 100, offsetY: 100, borderWidth: 2, borderColor: "black", type: "UmlClassifier", classifier: ClassifierShapes.interface, "interface": { name: "Patient", attributes: [{ name: "accepted" }] } }] }; }	Not applicable
Sets the data type of attribute	Property: nodes.interface.attributes.typeHTML Script public nodes; constructor() { this.nodes = [{ name: "Patient", offsetX: 100, offsetY: 100, borderWidth: 2, borderColor: "black", type: "UmlClassifier", classifier: ClassifierShapes.interface, "interface": { name: "Patient", attributes: [{ name: "accepted", type: "Date" }] } }] }; }	Not applicable
Defines the visibility of the attribute	Property: nodes.interface.attributes.scopHTML Script public nodes; constructor() { this.nodes = [{ name: "Patient", offsetX: 100, offsetY: 100, borderWidth: 2, borderColor: "black", type: "UmlClassifier", classifier: ClassifierShapes.interface,	Not applicable

	<code>"interface": { name: "Patient", attributes: [{ name: "accepted", type: "Date", scope:"protected" }] } }]; }</code>	
Defines the collection of methods of a interface	<code>Property:nodes.interface.methodsHTML Script public nodes; constructor() { this.nodes = [{ name: "Patient", offsetX: 100, offsetY: 100, borderWidth: 2, borderColor: "black", type: "UmlClassifier", classifier: ClassifierShapes.interface, "interface": { name: "Patient", methods: [{ name: "getHistory" }] } } }]; }</code>	Not applicable
Sets the name of the method	<code>Property:nodes.interface.methods.name HTML Script public nodes; constructor() { this.nodes = [{ name: "Patient", offsetX: 100, offsetY: 100, borderWidth: 2, borderColor: "black", type: "UmlClassifier", classifier: ClassifierShapes.interface, "interface": { name: "Patient", methods: [{ name: "getHistory", arguments: [{name: "Date" }] }] } } }]; }</code>	Not applicable
Defines the arguments of the method	<code>Property:nodes.interface.methods.argum entsHTML Script public nodes; constructor() { this.nodes = [{ name: "Patient", offsetX: 100, offsetY: 100, borderWidth: 2, borderColor: "black", type: "UmlClassifier", classifier: ClassifierShapes.interface, "interface": { name: "Patient", methods: [{ name: "getHistory", arguments: [{ name: "Date", type:"String" }] }] } }]; }</code>	Not applicable
Defines the name, attributes and methods of a interface. Applicable, if the node is a interface	<code>Property:nodes.interface.methods.argum ents.nameHTML Script public nodes; constructor() { this.nodes = [{ name: "Patient", offsetX: 100, offsetY: 100, borderWidth: 2, borderColor: "black", type: "UmlClassifier", classifier: ClassifierShapes.interface, "interface": { name: "Patient", methods: [{ name: "getHistory", arguments: [{ name: "Date" }] , type: "History" }] } }]; }</code>	Not applicable

Sets the type of the argument	Property: nodes.interface.methods.arguments.typeHTML Script <pre> public nodes; constructor() { this.nodes = [{ name: "Patient", offsetX: 100, offsetY: 100, borderWidth: 2, borderColor: "black", type: "UmlClassifier", classifier: ClassifierShapes.interface, "interface": { name: "Patient", methods: [{ name: "getHistory", arguments: [{ name: "Date" }], type: "History" }] } }]}; </pre>	Not applicable
Sets the return type of the method	Property: nodes.interface.methods.typeHTML Script <pre> public nodes; constructor() { this.nodes = [{ name: "Patient", offsetX: 100, offsetY: 100, borderWidth: 2, borderColor: "black", type: "UmlClassifier", classifier: ClassifierShapes.interface, "interface": { name: "Patient", methods: [{ name: "getHistory", arguments: [{name: "Date" }]}, type: "History" }] } }]}; </pre>	Not applicable
Sets the visibility of the method	Property: nodes.interface.methods.scopeHTML Script <pre> public nodes; constructor() { this.nodes = [{ name: "Patient", offsetX: 100, offsetY: 100, borderWidth: 2, borderColor: "black", type: "UmlClassifier", classifier: ClassifierShapes.interface, "interface": { name: "Patient", methods: [{ name: "getHistory", arguments: [{name: "Date" }]}, type: "History", scope: "protected" }] } }]}; </pre>	Not applicable
Defines whether the sub tree of the node is expanded or collapsed	Property: nodes.isExpandedHTML Script <pre> public nodes; constructor() { let node1 = { name: "node1", width: 50, height: 50, offsetX: 50, offsetY: 50, isExpanded: false }; let node2 = { name: "node2", width: 50, height: 50 }; let connector = { sourceNode: "node1", targetNode: "node2", name: "connector" }; this.nodes = [node1, node2]; } </pre>	Property: nodes.isExpandedHTML Script <pre> public nodes: NodeModel[] = [{ offsetX: 100, offsetY: 100, width: 100, height: 100, id: 'node1', isExpanded: true, }, { id: 'node2', width: 50, height: 50 }] public connectors: ConnectorMode = [{ sourceNode: 'node1', targetNode: 'node2', id: 'connector' }] public layout: object = { type: "HierarchicalTree" } </pre>

Sets the node as a swimlane	Property: nodes.isSwimlaneHTML Script <pre>public nodes; constructor() { this.nodes = [{ type: "swimlane", name: "swimlane", isSwimlane: true, header: { text: "Swimlane", fontSize: 12, bold: true } }]; }</pre>	Not applicable
A collection of objects where each object represents a label	Property: nodes.labelsHTML Script <pre>public nodes; constructor() { this.nodes = [{ name: "node", width: 100, height: 100, offsetX: 100, offsetY: 100, labels: [{ text: "Label", fontColor: "Red" }] }]; }</pre>	Property: nodes.annotationsHTML Script <pre>public nodes: NodeModel[] = [{ offsetX: 100, offsetY: 100, width: 100, height: 100, annotations: [{ content: 'Annotation' }] }]</pre>
An array of objects where each object represents a lane. Applicable, if the node is a swimlane	Property: nodes.lanesHTML Script <pre>public nodes; constructor() { this.nodes = [{ type: "swimlane", name: "swimlane", offsetX: 300, offsetY: 200, lanes: [{ name: "lane1", width: 200 }, { name: "lane2", width: 100 }] }]; }</pre>	Not applicable
This property allows you to customize lanes appearance using user-defined CSS	Property: nodes.lanes.cssClassHTML Script <pre>public nodes; constructor() { this.nodes = [{ type: "swimlane", name: "swimlane", offsetX: 300, offsetY: 200, lanes: [{ name: "lane1", width: 200 }, { name: "lane2", width: 100, cssClass:"hoverLane", addInfo: addInfo, fillColor:"blue" }] }]; }</pre>	Not applicable
Defines the header of the lane	Property: nodes.lanes.headerHTML Script <pre>public nodes; constructor() { this.nodes = [{ type: "swimlane", name: "swimlane", offsetX: 300, offsetY: 200, lanes: [{ name: "lane1", width: 200 }, { name: "lane2", width: 100, header: { fillColor:"blue", fontColor:"white", text:"Function 1" } }] }]]; }</pre>	Not applicable
Defines the width of lane	Property: nodes.lanes.widthHTML Script <pre>public nodes; constructor() { this.nodes = [{ type: "swimlane", name: "swimlane", offsetX: 300, offsetY: 200, lanes: [{ name: "lane1", width: 200, height: 200, zOrder:10 }, { name: "lane2", width: 100 }] }]; }</pre>	Not applicable

An array of objects where each object represents a child node of the lane	Property: <code>nodes.lanes.children</code> HTML Script <pre>public nodes; constructor() { this.nodes = [{ type: "swimlane", name: "swimlane", offsetX: 300, offsetY: 200, lanes: [{ name: "lane1", width: 200 }, { name: "lane2", width: 100, children:[{name:"process", width: 50, height: 50 }] }]]]; }</pre>	Not applicable
Defines the object as a lane	Property: <code>nodes.lanes.isLane</code> HTML Script <pre>public nodes; constructor() { this.nodes = [{ type: "swimlane", name: "swimlane", offsetX: 300, offsetY: 200, lanes: [{ name: "lane1", width: 200, height: 200, isLane:true, orientation:"vertical" }, { name: "lane2", width: 100 }]]]; }</pre>	Not applicable
Defines the minimum space to be left between the bottom of parent bounds and the node	Property: <code>nodes.margin</code> HTML Script <pre>public nodes; constructor() { this.nodes = [{ type: "swimlane", name: "swimlane", offsetX: 300, offsetY: 200, lanes: [{ name: "lane1", width: 200, children: [{ name: "process", width: 50, height: 50, marginBottom: 50, marginLeft: 10, marginRight: 10, marginTop: 10 }] }]]]; }</pre>	Property: <code>nodes.margin</code> HTML Script <pre>public nodes: NodeModel[] = [{ offsetX: 100, offsetY: 100, width: 100, height: 100, margin : { left: 15, right: 15, top: 15, bottom: 15 } }]</pre>
Defines the maximum height limit of the node	Property: <code>nodes.maxHeight</code> HTML Script <pre>public nodes; constructor() { this.nodes = [{ name: "node1", width: 50, height: 50, offsetX: 50, offsetY: 50, maxHeight: 100, maxWidth: 100, minHeight: 10, minWidth: 10 }]]; }</pre>	Property: <code>nodes.maxHeight</code> HTML Script <pre>public nodes: NodeModel[] =[{ offsetX: 100, offsetY: 100, width: 100, height: 100, maxHeight: 100, maxWidth: 100, minHeight: 10, minWidth: 10 }]</pre>
Sets the unique identifier of the node	Property: <code>nodes.name</code> HTML Script <pre>public nodes; constructor() { this.nodes = [{ name: "node1", width: 50, height: 50, offsetX: 50, offsetY: 50, }]]; }</pre>	Property: <code>nodes.id</code> HTML Script <pre>public nodes: NodeModel[] = [{ id: 'node1' offsetX: 100, offsetY: 100, width: 100, height: 100, }]</pre>
Defines the opaque of the node	Property: <code>nodes.opacity</code> HTML Script <pre>public nodes; constructor() { this.nodes = [{ name: "node1", width: 50, height: 50, offsetX: 50, offsetY: 50, opacity: 0.5, rotateAngle: 70 }]]; }</pre>	Property: <code>nodes.style.opacity</code> HTML Script <pre>public nodes: NodeModel[] = [{ id: 'node1' offsetX: 100, offsetY: 100, width: 100, height: 100, rotateAngle: 70, style: { opacity: 0.5 } }]</pre>
Defines the minimum padding value	Property: <code>nodes.paddingBottom</code> HTML Script <pre>public nodes; constructor() { let node1 = { name: "node1", width: 50, height:50}; let node2</pre>	Not applicable Property: <code>pageSettings.background.color</code> HTML Script <pre>public pageSettings:</pre>

to be left between the bottom most position of a group and its children. Applicable, if the group is a container	<pre>= { name: "node2", width: 50, height:50, verticalAlign: "bottom"}; let group = { name : "group", children:[node1, node2], container: { type: "canvas" }, offsetX:200, offsetY:100, fillColor:"gray", minWidth:200, minHeight:200, paddingBottom:10, paddingLeft:10, paddingRight:10, paddingTop:10 }; this.nodes = [group]; }</pre>	<pre>PageSettingsModel = { background: { color: 'red', source: 'Clayton.png', scale: 'Meet', align: 'XMidYMid' } }</pre>
Defines the scrollable area of diagram. Applicable, if the scroll limit is limited	Property: <code>pageSettings.scrollableAreaHTML</code> Script public <code>pageSettings</code> ; constructor() { this.pageSettings = { scrollableArea: { x:0, y:0, width:1000, height:1000} }; }	Property: <code>scrollSettings.scrollableAreaHTML</code> Script public <code>scrollSettings</code> : ScrollSettingsModel = { scrollableArea: new Rect(0, 0, 300, 300), }
Defines the draggable region of diagram elements	Property: <code>pageSettings.boundaryConstraintsHTML</code> Script public <code>pageSettings</code> ; constructor() { this.pageSettings = { boundaryConstraints: BoundaryConstraints.Diagram }; }	Property: <code>pageSettings.boundaryConstraintsHTML</code> Script public <code>pageSettings</code> : PageSettingsModel = { width: 800, height: 600, boundaryConstraints: 'Diagram' }

SymbolPalette

<!-- markdownlint-disable MD033 -->

Behavior	API in Essential JS 1	API in Essential JS 2
Defines the size and preview size of the node to add that to symbol palette	Property: <code>nodes.paletteItemHTML</code> Script public <code>palettes</code> = [{ name: "Basic Shapes", expanded: true, items: [{ name: "Rectangle", height: 40, width: 80, paletteItem: { previewWidth: 100, previewHeight: 100 } }] }	Property: <code>palettesHTML</code> Script private <code>FlowShapes</code> : NodeModel[] = [{ id: 'Terminator', shape: { type: 'Flow', shape: 'Terminator' }, style: { strokeWidth: 2 } }, { id: 'Process', shape: { type: 'Flow', shape: 'Process' }, style: { strokeWidth: 2 } }, { id: 'Decision', shape: { type: 'Flow', shape: 'Decision' }, style: { strokeWidth: 2 } },] public <code>palettes</code> : PaletteModel[] = [{ id: 'flow', expanded: true, symbols: this.FlowShapes, title:

		<pre>'Flow Shapes' },] public symbolMargin= { left: 12, right: 12, top: 12, bottom: 12 } public getSymbolInfo: (symbol: NodeModel): SymbolInfo => { return { fit: true }; }</pre>
Defines whether the symbol should be drawn at its actual size regardless of precedence factors or not	Property: <code>nodes.palettItem.enableScaleHTML</code> Script <pre>public palettes= [{ name: "Basic Shapes", expanded: true, items: [{ name: "Rectangle", height: 40, width: 80, paletteItem: { previewWidth: 100, previewHeight: 100, enableScale:false } }] }]</pre>	Property: <code>palettes.fitHTML</code> Script <pre>private FlowShapes: NodeModel[] = [{ id: 'Terminator', shape: { type: 'Flow', shape: 'Terminator' }, style: { strokeWidth: 2 } }, { id: 'Process', shape: { type: 'Flow', shape: 'Process' }, style: { strokeWidth: 2 } }, { id: 'Decision', shape: { type: 'Flow', shape: 'Decision' }, style: { strokeWidth: 2 } },] public palettes:PaletteModel[] = [{ id: 'flow', expanded: true, symbols: this.FlowShapes, title: 'Flow Shapes' },] public symbolMargin= { left: 12, right: 12, top: 12, bottom: 12 }</pre>
To display a name for nodes in the symbol palette	Property: <code>nodes.palettItem.labelHTML</code> Script <pre>public palettes = [{ name: "Basic Shapes", expanded: true, items: [{ name: "Rectangle", height: 40, width: 80, paletteItem: { previewWidth: 100, previewHeight: 100, label: "label", margin: { left: 4, right: 4, top: 4, bottom: 4 } } }] }</pre>	Property: <code>palettes.titleHTML</code> Script <pre>private FlowShapes: NodeModel[] = [{ id: 'Terminator', shape: { type: 'Flow', shape: 'Terminator' }, style: { strokeWidth: 2 } }, { id: 'Process', shape: { type: 'Flow', shape: 'Process' }, style: { strokeWidth: 2 } }, { id: 'Decision', shape: { type: 'Flow', shape: 'Decision' }, style: { strokeWidth: 2 } },] public palettes:PaletteModel[] = [{ id: 'flow', expanded: true, symbols: this.FlowShapes, title:</pre>

		<pre>'Flow Shapes' },] public symbolMargin= { left: 12, right: 12, top: 12, bottom: 12 } public getSymbolInfo: (symbol: NodeModel): SymbolInfo => { return { fit: true }; }</pre>
--	--	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

SelectedItems

<!-- markdownlint-disable MD033 -->

Behavior	API in Essential JS 1	API in Essential JS 2
A read only collection of the selected items	Property: <code>selectedItems.children</code> <pre>@ViewChild('diagram') diagram: EJComponents; //Read the selected items for(let i =0; i< this.diagram.widget.model.selectedItems.children; i++){ //Do your actions here }</pre>	Not applicable
Controls the visibility of selector	Property: <code>selectedItems.constraintsHTML</code> Script <pre>public selectedItems; constructor() { this.selectedItems = { constraints: SelectorConstraints.UserHandles } }</pre>	Property: <code>selectedItems.constraintsHTML</code> Script <pre>public selectedItems = { constraints: SelectorConstraints.UserHandle }</pre>
Defines a method that dynamically enables/disables the interaction with multiple selection	Property: <code>selectedItems.getConstraintsHTML</code> Script <pre>public selectedItems; constructor() { this.selectedItems = { getConstraints: function() { return NodeConstraints.Drag NodeConstraints.Resize } }</pre>	Not applicable
Sets the height of the selected items	Property: <code>selectedItems.heightHTML</code> Script <pre>public selectedItems; constructor() { this.selectedItems = { height:100, width: 100, offsetX:100, offsetY: 100, rotateAngle: 90, } }</pre>	Property: <code>selectedItems.heightHTML</code> Script <pre>public selectedItems = { height:100, width: 100, offsetX:100, offsetY: 100, rotateAngle: 90 }, }</pre>

Sets the angle to rotate the selected items	Property: <code>selectedItems.tooltipHTML</code> Script <pre>public selectedItems; constructor() { this.selectedItems = { tooltip : { alignment: { vertical: "top" } } } }</pre>	Not applicable
A collection of frequently used commands that will be added around the selector	Property: <code>selectedItems.userHandlesHTML</code> Script <pre>public selectedItems; constructor() { this.selectedItems = { userHandles: userHandle } }</pre>	Property: <code>selectedItems.userHandlesHTML</code> Script <pre>public handle: UserHandleModel[] = [{ name: 'handle1', pathData: 'M 60.3,18 H 27.5 c -3,0-5.5,2.4-5.5,5.5 v 38.2 h 5.5 V 23.5 h 32.7 V 18 z M 68.5,28.9 h -30 c -3,0- 5.5,2.4-5.5,5.5 v 38.2 c 0,3,2.4,5.5,5.5,5.5 h 30 c 3,0,5.5-2.4,5.5-5.5 V 34.4 C 73.9,31.4,71.5,28.9,68.5,28.9 z M 68.5,72.5 h -30 V 34.4 h 30 V 72.5 z' , visible: true, backgroundColor: 'black', offset: 0, side: 'Bottom', margin: { top: 0, bottom: 0, left: 0, right: 0 }, pathColor: 'white' }]; public selectedItems = { constraints: SelectorConstraints.UserHandle, userHandles: this.handle }</pre>
Sets the horizontal alignment of the user handle	Property: <code>selectedItems.userHandles.horizontalAlignmentHTML</code> Script <pre>public selectedItems; constructor() { let userHandle = []; let cloneHandle = UserHandle(); cloneHandle = {name : "cloneHandle", pathData : "M 4.6350084,4.8909971 L 4.6350084,9.3649971 9.5480137,9.3649971 9.5480137,4.8909971 z M 3.0000062,2.8189973 L 11.184016,2.8189973 11.184016,10.999997 3.0000062,10.999997 z M 0,0 L 7.3649998,0 7.3649998,1.4020001 1.4029988,1.4020001 1.4029988,8.0660002 0,8.0660002 0,1.4020001 0,0.70300276 z", visible : "true", backgroundColor : "#4D4D4D", offset : point(0, 0), position : " middleLeft", margin : { left: 5 }, pathColor : "white", horizontalAlignment : HorizontalAlignment.Right, verticalAlignment : VerticalAlignment.Top, borderColor :</pre>	Property: <code>selectedItems.userHandlesHTML</code> Script <pre>public handle: UserHandleModel[] = [{ name: 'handle1', pathData: 'M 60.3,18 H 27.5 c -3,0-5.5,2.4-5.5,5.5 v 38.2 h 5.5 V 23.5 h 32.7 V 18 z M 68.5,28.9 h -30 c -3,0- 5.5,2.4-5.5,5.5 v 38.2 c 0,3,2.4,5.5,5.5,5.5 h 30 c 3,0,5.5-2.4,5.5-5.5 V 34.4 C 73.9,31.4,71.5,28.9,68.5,28.9 z M 68.5,72.5 h -30 V 34.4 h 30 V 72.5 z', visible: true, backgroundColor: 'black', offset: 0, side: 'Bottom', margin: { top: 0, bottom: 0, left: 0, right: 0 }, pathColor: 'white', horizontalAlignment: 'Center', verticalAlignment: 'Center', borderColor: 'red', borderWidth: 3, size: 30 }]; public selectedItems = { constraints: SelectorConstraints.UserHandle, userHandles: this.handle }</pre>

	<pre>"red", borderWidth : 3, size : 20}; userHandle.push(cloneHandle); this.selectedItems = { userHandles: userHandle } }</pre>	
Defines the interactive behavior of the user handle	<p>Property: <code>selectedItems.userHandles.toolHTML</code></p> <pre>Script public selectedItems; constructor() { let CloneTool = (function(base) { extend(CloneTool, base); function CloneTool(name) { base.call(this, name); this.singleAction = true; this.clonedNodes = []; this.cursor = "pointer"; } CloneTool.prototype.mouseup = function(event) { this.diagram.copy(); this.diagram.paste(); } } return CloneTool; })(ToolBase); let userHandle = []; let cloneHandle = UserHandle(); cloneHandle.name = "cloneHandle"; cloneHandle.pathData = "M 4.6350084,4.8909971 L 4.6350084,9.3649971 9.5480137,9.3649971 9.5480137,4.8909971 z M 3.0000062,2.8189973 L 11.184016,2.8189973 11.184016,10.999997 3.0000062,10.999997 z M 0,0 L 7.3649998,0 7.3649998,1.4020001 1.4029988,1.4020001 1.4029988,8.0660002 0,8.0660002 0,1.4020001 0,0.70300276 z"; cloneHandle.tool = new CloneTool(cloneHandle.name); userHandle.push(cloneHandle); this.selectedItems = { userHandles: userHandle } }</pre>	Not applicable
Defines whether the user handle should be added, when more than one element	<p>Property: <code>selectedItems.userHandles.enableMultiSelectionHTML</code></p> <pre>Script public selectedItems; constructor() { let userHandle = []; let cloneHandle = UserHandle(); cloneHandle.name = "cloneHandle"; cloneHandle.enableMultiSelection = true; userHandle.push(cloneHandle); this.selectedItems = { userHandles: userHandle } }</pre>	Not applicable

is selected		
Sets the horizontal alignment of the user handle	Not applicable	Property: selectedItems.userHandles.displacementHTML Script <pre> public handle: UserHandleModel[] = [{ name: 'handle1', pathData: 'M 60.3,18 H 27.5 c -3,0-5.5,2.4-5.5,5.5 v 38.2 h 5.5 V 23.5 h 32.7 V 18 z M 68.5,28.9 h -30 c -3,0- 5.5,2.4-5.5,5.5 v 38.2 c 0,3,2.4,5.5,5.5,5.5 h 30 c 3,0,5.5-2.4,5.5-5.5 V 34.4 C 73.9,31.4,71.5,28.9,68.5,28.9 z M 68.5,72.5 h -30 V 34.4 h 30 V 72.5 z', displacement: 30 }]]; public selectedItems = { constraints: SelectorConstraints.UserHandle, userHandles: this.handle } </pre>

SerializationSettings

<!-- markdownlint-disable MD033 -->

Behavior	API in Essential JS 1	API in Essential JS 2
Defines whether the default diagram properties can be serialized or not	Property: serializationSettings.preventDefaultValuesHTML Script <pre> public serializationSettings; constructor() { this.serializationSettings = { serializationSettings:{ preventDefaultValues: true } } }; } </pre>	Not applicable

How to load EJ1 diagram in EJ2 diagram

To load EJ1 JSON data in an EJ2 diagram, follow these steps.

1. Import and inject the EJ1SerializationModule as shown in the following code example.

```
`typescript
```

```
import { Diagram, DiagramComponent } from '@syncfusion/ej2-diagrams';
```

```
import { EJ1SerializationModule } from '@syncfusion/ej2-diagrams';
```

```
Diagram.Inject(EJ1SerializationModule);
```

```
,
```

2. Load the EJ1 JSON data using the diagram loadDiagram method and set the second parameter to true.

```
`typescript
```

```
@Component({
```

```

selector: "app-container",
template: `<ejs-diagram id="diagram"width="100%" height="700px">
</ejs-diagram>`,
encapsulation: ViewEncapsulation.None
})
export class AppComponent {
public diagram : DiagramComponent;
let ej1data = {"JSONData"}; // Replace JSONData with your EJ1 JSON data
//Load the EJ1 JSON and pass boolean value as true
this.diagram.loadDiagram(ej1data, true);
}
`

```

Tooltip

<!-- markdownlint-disable MD033 -->

Behavior	API in Essential JS 1	API in Essential JS 2
An object that defines the description, appearance and alignments of tooltip	Property: tooltipHTML Script <pre> public nodes=[{ name: "Elizabeth", width: 70, height: 40, offsetX: 100, offsetY: 100, Designation: "Managing Director" }] public tooltip= { templateId: "mouseOverTooltip", relativeMode: RelativeMode.Mouse, } </pre>	Property: tooltipHTML Script <pre> public constraints = DiagramConstraints.Default DiagramConstraints.Tooltip, public tooltip = { content: 'Diagram', position: 'TopLeft', relativeMode: 'Object', animation: { open: { effect: 'FadeZoomIn', delay: 0 }, close: { effect: 'FadeZoomOut', delay: 0 } } } </pre>
Defines the alignment of tooltip	Property: tooltip.alignmentHTML Script <pre> public nodes=[{ name: "Elizabeth", width: 70, height: 40, offsetX: 100, offsetY: 100, Designation: "Managing Director" }] public tooltip= { templateId: "mouseOverTooltip", alignment: { horizontal: "center", vertical: "bottom" }, relativeMode: RelativeMode.Mouse, } </pre>	Not applicable
Sets the margin of the tooltip	Property: tooltip.marginHTML Script <pre> public nodes=[{ name: "Elizabeth", width: 70, height: 40, offsetX: </pre>	Not applicable

	<pre>100, offsetY: 100, Designation: "Managing Director" }} public tooltip= { templateId: "mouseOverTooltip", alignment: { horizontal: "center", vertical: "bottom" }, relativeMode: RelativeMode.Mouse, margin : { left: 5, right: 5, top: 5, bottom: 5 } }</pre>	
Sets the svg/html template to be bound with tooltip	Property:tooltip.templateIdHTML Script public nodes=[{ name: "Elizabeth", width: 70, height: 40, offsetX: 100, offsetY: 100, Designation: "Managing Director" }} public tooltip= { templateId: "mouseOverTooltip", alignment: { horizontal: "center", vertical: "bottom" } }	Property:tooltip.contentHTML Script public constraints = DiagramConstraints.Default DiagramConstraints.Tooltip, public tooltip = { content: 'Diagram', relativeMode: 'Object', } }
Defines if the Tooltip has tip pointer or not	Not applicable	Property:tooltip.showTipPointerHTML Script public constraints = DiagramConstraints.Default DiagramConstraints.Tooltip, public tooltip = { content: 'Diagram', position: 'TopLeft', relativeMode: 'Object', showTipPointer: true, } }
Defines the position of the Tooltip	Not applicable	Property:tooltip.positionHTML Script public constraints = DiagramConstraints.Default DiagramConstraints.Tooltip, public tooltip = { content: 'Diagram', position: 'TopLeft', relativeMode: 'Object', } }
Allows to set the same or different animation option for the Tooltip, when it is opened or closed	Not applicable	Property:tooltip.animationHTML Script public constraints = DiagramConstraints.Default DiagramConstraints.Tooltip, public tooltip = { content: 'Diagram', position: 'TopLeft', relativeMode: 'Object', animation: { open: { effect: 'FadeZoomIn', delay: 0 }, close: { effect:

		'FadeZoomOut', delay: 0 } } }
Sets the width of the tooltip	Not applicable	Property: <code>tooltip.width</code> HTML Script <pre> public constraints = DiagramConstraints.Default DiagramConstraints.Tooltip, public tooltip = { width: 100, content: 'Diagram', position: 'TopLeft', relativeMode: 'Object', animation: { open: { effect: 'FadeZoomIn', delay: 0 }, close: { effect: 'FadeZoomOut', delay: 0 } } } </pre>
Sets the height of the Tooltip	Not applicable	Property: <code>tooltip.height</code> HTML Script <pre> public constraints = DiagramConstraints.Default DiagramConstraints.Tooltip, public tooltip = { height: 100, content: 'Diagram', position: 'TopLeft', relativeMode: 'Object', animation: { open: { effect: 'FadeZoomIn', delay: 0 }, close: { effect: 'FadeZoomOut', delay: 0 } } } </pre>

Dialog

Getting started with Angular Dialog component

This section explains the steps to create a simple **Dialog** component and configure its available functionalities in Angular.

Getting Started with Angular CLI

The following section explains the steps required to create a simple angular-cli application and how to configure **Dialog** component.

Prerequisites

To get started with Syncfusion Angular UI Components, make sure that you have compatible versions of Angular and TypeScript.

- Angular : 6+
- TypeScript : 2.6+

Setting up an Angular project

Angular provides an easiest way to setup project using Angular CLI [Angular CLI](#) tool.

Install the CLI application globally in your machine.

```
`javascript
```

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

```
,
```

Create a new application

```
`javascript
```

```
ng new syncfusion-angular-app
```

```
,
```

Once you have executed the above command you may ask for following options,

- Would you like to add Angular routing?
- Which stylesheet format would you like to use?

By default, it install the CSS style base application. To setup with SCSS, pass `--style=SCSS` argument on create project.

Example code snippet.

```
`javascript
```

```
ng new syncfusion-angular-app --style=SCSS
```

```
,
```

Use below command to Navigate into the created project folder.

```
`javascript
```

```
cd syncfusion-angular-app
```

```
,
```

Installing Syncfusion Popups package

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

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

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

Ivy library distribution package

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

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

```
`bash
```

```
npm install @syncfusion/ej2-angular-popups --save
```

```
,
```

Angular compatibility compiled package(ngcc)

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

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

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

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

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

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

Adding Dialog module

Once you have successfully installed the popups package, the component modules are ready to configure in your application from the installed location. Syncfusion Angular package provides two different types of ngModules.

Refer to [Ng-Module](#) to learn about ngModules.

Refer to the following snippet to import the `DialogModule` in `app.module.ts` from the `@syncfusion/ej2-angular-popups`.

```
`javascript
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { AppRoutingModuleModule } from './app-routing.module';
import { AppComponent } from './app.component';
// Imported syncfusion DialogModule from popups package
import { DialogModule } from '@syncfusion/ej2-angular-popups';
@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModuleModule,
```

```
// Registering EJ2 Dialog Module
```

```
DialogModule
```

```
],
```

```
providers: [],
```

```
bootstrap: [AppComponent]
```

```
})
```

```
export class AppModule { }
```

```
,
```

Adding Dialog component

Add the Dialog component snippet in `app.component.ts` as follows.

```
`javascript
```

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

```
import { DialogComponent } from '@syncfusion/ej2-angular-popups';
```

```
import { EmitType } from '@syncfusion/ej2-base';
```

```
@Component({
```

```
  selector: 'app-root',
```

```
  template: `
```

```
    <button class="e-control e-btn" style="position: absolute;" id="targetButton"
    (click)="onOpenDialog($event)">Open Dialog</button>
```

```
    <div #container class='root-container'></div>
```

```
    <ejs-dialog id='dialog' #ejDialog content='This is a Dialog with content' [target]='targetElement'
    width='250px'>
```

```
    </ejs-dialog>`
```

```
  })
```

```
  export class AppComponent implements OnInit {
```

```
    @ViewChild('ejDialog') ejDialog: DialogComponent;
```

```
    // Create element reference for dialog target element.
```

```
    @ViewChild('container', { read: ElementRef, static: true }) container: ElementRef;
```

```
    // The Dialog shows within the target element.
```

```
    public targetElement: HTMLElement;
```

```
    // To get all element of the dialog component after component get initialized.
```

```
    ngOnInit() {
```

```
      this.initilaizeTarget();
```

```
    }
```

```
// Initialize the Dialog component target element.
public initilaizeTarget: EmitType<object> = () => {
this.targetElement = this.container.nativeElement.parentElement;
}
// Sample level code to handle the button click action
public onOpenDialog = function(event: any): void {
// Call the show method to open the Dialog
this.ejDialog.show();
};
}
```

Add following styles in corresponding css file. The below mentioned styles are used in styles.css file,

```
`css
html,
body,
dialog-container {
display: block;
height: 100%;
margin: 0;
overflow: hidden;
width: 100%;
}
`
```

Note: Please do the necessary change in `index.html` file. In this demo we used id selector in `<app-root id='dialog-container'></app-root>`

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-icons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-angular-popups/styles/material.css';
`
```


The [Custom Resource Generator \(CRG\)](#) is an online web tool, which can be used to generate the custom script and styles for a set of specific components.

This web tool is useful to combine the required component scripts and styles in a single file.

Running the application

Run the `ng serve` command in command window, it will serve your application and you can open the browser window. Output will appear as follows.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DialogModule } from '@syncfusion/ej2-angular-popups'
import { Component, ViewChild, OnInit, ElementRef } from '@angular/core';
import { DialogComponent } from '@syncfusion/ej2-angular-popups';
import { EmitType } from '@syncfusion/ej2-base';
@Component({
  imports: [

    DialogModule

  ],
  standalone: true,
  selector: 'app-root',
  template: `
    <button class="e-control e-btn" style="position: absolute;"
    id="targetButton" (click)="onOpenDialog($event)">Open Dialog</button>
    <div #container class='root-container'></div>
    <ejs-dialog id='dialog' #ejDialog content='This is a Dialog with
    content' [target]='targetElement' width='250px'>
    </ejs-dialog>
  `
})
export class AppComponent implements OnInit {
  @ViewChild('ejDialog') ejDialog: DialogComponent | any;
  // Create element reference for dialog target element.
  @ViewChild('container', { read: ElementRef, static: true }) container:
  ElementRef | any;
  // The Dialog shows within the target element.
  public targetElement?: HTMLElement;
  // To get all element of the dialog component after component get
  initialized.
  ngOnInit() {
    this.initilaizeTarget();
  }
  // Initialize the Dialog component target element.
  public initilaizeTarget: EmitType<object> = () => {
    this.targetElement = this.container.nativeElement.parentElement;
  }
  // Sample level code to handle the button click action
  public onOpenDialog = (event: any): void => {
    // Call the show method to open the Dialog
    this.ejDialog.show();
  };
}
```

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 dialog control, max-height is calculated based on the dialog target element height. If the target property is not configured, the document.body is considered as a target. Therefore, to show a dialog in proper height, you need to add min-height to the target element.

If the dialog is rendered based on the body, then the dialog will get the height based on its body element height. If the height of the dialog is larger than the body height, then the dialog's height will not be set. For this scenario, we can set the CSS style for the html and body to get the dialog height.

```
`css
```

```
html, body {
```

```
height: 100%;
```

```
}
```

```
`
```

Modal Dialog

A [modal](#) shows an overlay behind the Dialog. So, the user should interact the Dialog compulsory before interacting with the remaining content in an application.

While the user clicks the overlay, the action can be handled through the [overlayClick](#) event. In the below sample the Dialog close action is performed while clicking on the overlay.

When the modal Dialog is opened, the Dialog's target scrolling will be disabled. The scrolling will be enabled again once close the Dialog.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DialogModule } from '@syncfusion/ej2-angular-popups'
import { Component, ViewChild, OnInit, ElementRef } from '@angular/core';
import { DialogComponent } from '@syncfusion/ej2-angular-popups';
import { EmitType } from '@syncfusion/ej2-base';
@Component({
  imports: [
    DialogModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `
    <button class="e-control e-btn" style="position: absolute;"
    id="targetButton" (click)="onOpenDialog($event)">Open Modal Dialog</button>
    <div #container class='root-container'></div>
    <ejs-dialog id='dialog' #ejDialog isModal='true'
    (overlayClick)="onOverlayClick()"`
})
```

```

        content='This is a modal dialog' [target]='targetElement' width='250px'>
</ejs-dialog>`
    })
    export class AppComponent implements OnInit {
        @ViewChild('ejDialog') ejDialog: DialogComponent | any;
        // The Dialog shows within the target element.
        @ViewChild('container', { read: ElementRef, static: true }) container:
        ElementRef | any;
        // The Dialog shows within the target element.
        public targetElement?: HTMLElement;
        // To get all element of the dialog component after component get
        initialized.
        ngOnInit() {
            this.initilaizeTarget();
        }
        // Initialize the Dialog component target element.
        public initilaizeTarget: EmitType<object> = () => {
            this.targetElement = this.container.nativeElement.parentElement;
        }
        // Sample level code to handle the button click action
        public onOpenDialog = (event: any): void => {
            // Call the show method to open the Dialog
            this.ejDialog.show();
        };
        // Sample level code to hide the Dialog when click the Dialog overlay
        public onOverlayClick: EmitType<object> = () => {
            this.ejDialog.hide();
        }
    }
}

```

MAIN.TS

```

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

```

Enable header

The Dialog header can be enabled by adding the header content as text or HTML content through the [header](#) property.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DialogModule } from '@syncfusion/ej2-angular-popups'
import { Component, ViewChild, OnInit, ElementRef } from '@angular/core';
import { DialogComponent } from '@syncfusion/ej2-angular-popups';
import { EmitType } from '@syncfusion/ej2-base';
@Component({
    imports: [

        DialogModule

    ],
    standalone: true,

```

```

    selector: 'app-root',
    template: `
      <button class="e-control e-btn" style="position: absolute;"
id="targetButton" (click)="onOpenDialog($event)">Open Dialog</button>
      <div #container class='root-container'></div>
      <ejs-dialog id='dialog' #ejDialog header='Dialog' showCloseIcon='true'
content='This is a dialog with header'
      [target]='targetElement' width='250px'> </ejs-dialog>`
  })
  export class AppComponent implements OnInit {
    @ViewChild('ejDialog') ejDialog: DialogComponent | any;
    // Create element reference for dialog target element.
    @ViewChild('container', { read: ElementRef, static: true }) container:
    ElementRef | any;
    // The Dialog shows within the target element.
    public targetElement?: HTMLElement;
    // To get all element of the dialog component after component get
    initialized.
    ngOnInit() {
      this.initilaizeTarget();
    }
    // Initialize the Dialog component target element.
    public initilaizeTarget: EmitType<object> = () => {
      this.targetElement = this.container.nativeElement.parentElement;
    }
    // Sample level code to handle the button click action
    public onOpenDialog = (event: any): void => {
      // Call the show method to open the Dialog
      this.ejDialog.show();
    };
  }
}

```

MAIN.TS

```

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

```

Enable footer

The Dialog provides built-in support to render the buttons on the footer (for ex: 'OK' or 'Cancel' buttons). Each Dialog button allows the user to perform any action while clicking on it.

The primary button will be focused automatically when open the Dialog and add the [click](#) event to handle the actions

When the Dialog initialize with more than one primary buttons, the first primary button gets focus on open the Dialog.

The below sample render with button and its click event.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DialogModule } from '@syncfusion/ej2-angular-popups'

```

```

import { Component, ViewChild, OnInit, ElementRef } from '@angular/core';
import { DialogComponent } from '@syncfusion/ej2-angular-popups';
import { EmitType } from '@syncfusion/ej2-base';
@Component({
  imports: [

    DialogModule

  ],
  standalone: true,
  selector: 'app-root',
  template: `
    <button class="e-control e-btn" style="position: absolute;"
    id="targetButton" (click)="onOpenDialog($event)">Open Dialog</button>
    <div #container class='root-container'></div>
    <ejs-dialog id='dialog' #ejDialog header='Dialog' content='This is a
    Dialog with button and primary button' [target]='targetElement'
    width='250px' [buttons]='buttons'>
    </ejs-dialog>
  `
})
export class AppComponent implements OnInit {
  @ViewChild('ejDialog') ejDialog: DialogComponent | any;
  // Create element reference for dialog target element.
  @ViewChild('container', { read: ElementRef, static: true }) container:
  ElementRef | any;
  // The Dialog shows within the target element.
  public targetElement?: HTMLElement;
  // To get all element of the dialog component after component get
  initialized.
  ngOnInit() {
    this.initilaizeTarget();
  }
  // Initialize the Dialog component's target element.
  public initilaizeTarget: EmitType<object> = () => {
    this.targetElement = this.container.nativeElement.parentElement;
  }
  // Hide the Dialog when click the footer button.
  public hideDialog: EmitType<object> = () => {
    this.ejDialog.hide();
  }
  // Enables the footer buttons
  public buttons: Object = [
    {
      'click': this.hideDialog.bind(this),
      // Accessing button component properties by buttonModel property
      buttonModel: {
        content: 'OK',
        // Enables the primary button
        isPrimary: true
      }
    },
    {
      'click': this.hideDialog.bind(this),
      buttonModel: {
        content: 'Cancel'
      }
    }
  ]
}

```

```

];
// Sample level code to handle the button click action
public onOpenDialog = (event: any): void => {
    // Call the show method to open the Dialog
    this.ejDialog.show();
};
}

```

MAIN.TS

```

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

```

Draggable

The Angular Dialog supports to [drag](#) within its target container by grabbing the Dialog header, which allows the user to reposition the Dialog dynamically.

The Dialog can be draggable only when the Dialog header is enabled.

From 16.2.x version, enabled draggable support for modal Dialog also.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DialogModule } from '@syncfusion/ej2-angular-popups'
import { Component, ViewChild, OnInit, ElementRef } from '@angular/core';
import { DialogComponent } from '@syncfusion/ej2-angular-popups';
import { EmitType } from '@syncfusion/ej2-base';
@Component({
  imports: [

    DialogModule

  ],
  standalone: true,
  selector: 'app-root',
  template: `
    <button class="e-control e-btn" style="position: absolute;"
    id="targetButton" (click)="onOpenDialog($event)">Open Dialog</button>
    <div #container class='root-container'></div>
    <ejs-dialog id='dialog' #ejDialog allowDragging='true' header='Dialog'
    content='This is a Dialog with drag enabled'
    [target]='targetElement' width='250px' [buttons]='buttons'> </ejs-
    dialog>`
})
export class AppComponent implements OnInit {
  @ViewChild('ejDialog') ejDialog: DialogComponent | any;
  // Create element reference for dialog target element.
  @ViewChild('container', { read: ElementRef, static: true }) container:
  ElementRef | any;
  // The Dialog shows within the target element.
  public targetElement?: HTMLElement;
  // To get all element of the dialog component after component get
  initialized.

```

```

ngOnInit() {
    this.initilaizeTarget();
}
// Initialize the Dialog component's target element.
public initilaizeTarget: EmitType<object> = () => {
    this.targetElement = this.container.nativeElement.parentElement;
}
// Hide the Dialog when click the footer button.
public hideDialog: EmitType<object> = () => {
    this.ejDialog.hide();
}
// Enables the footer buttons
public buttons: Object = [
    {
        'click': this.hideDialog.bind(this),
        // Accessing button component properties by buttonModel property
        buttonModel: {
            content: 'OK',
            isPrimary: true
        }
    },
    {
        'click': this.hideDialog.bind(this),
        buttonModel: {
            content: 'Cancel'
        }
    }
];
// Sample level code to handle the button click action
public onOpenDialog = (event: any): void => {
    // Call the show method to open the Dialog
    this.ejDialog.show();
};
}

```

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 Angular Dialog can be positioned using the [position](#) property by providing the X and Y co-ordinates. It can be positioned inside the target of the container or `<body>` of the element based on the given X and Y values.

- for X is: left, center, right (or) any offset value
- for Y is: top, center, bottom (or) any offset value

The below example demonstrates the different Dialog positions.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DialogModule } from '@syncfusion/ej2-angular-popups'
import { Component, ViewChild, OnInit, ElementRef } from '@angular/core';
import { DialogComponent } from '@syncfusion/ej2-angular-popups';
import { EmitType } from '@syncfusion/ej2-base';
@Component({
  imports: [

    DialogModule

  ],
  standalone: true,
  selector: 'app-root',
  template: `
    <div #container class='root-container'></div>
    <ejs-dialog id='dialog' #ejDialog [position]='position'
    [target]='targetElement' width='363px'
    [closeOnEscape]='closeOnEscape'>
      <ng-template #header>
        <span>Choose a Dialog Position</span>
      </ng-template>
      <ng-template #content>
        <table style="width:320px;padding:18px; padding-top:0px;">
          <tr>
            <td><input type="radio" name="xy"
            (click)="changePosition($event)" value="left top"
            checked="true">left top</td>
            <td><input type="radio" name="xy"
            (click)="changePosition($event)"
            value="center top">center top</td>
            <td><input type="radio" name="xy"
            (click)="changePosition($event)" value="right top">right top</td> </tr>
            <tr>
            <td><input type="radio" name="xy"
            (click)="changePosition($event)" value="left center">left center</td>
            <td><input type="radio" name="xy"
            (click)="changePosition($event)" value="center center">center center</td>
            <td><input type="radio" name="xy"
            (click)="changePosition($event)" value="right center">right center</td>
            </tr>
            <tr>
            <td><input type="radio" name="xy"
            (click)="changePosition($event)" value="left bottom">left bottom</td>
            <td><input type="radio" name="xy"
            (click)="changePosition($event)" value="center bottom">center bottom</td>
            <td><input type="radio" name="xy"
            (click)="changePosition($event)" value="right bottom">right bottom</td>
            </tr>
          </table>
          </ng-template>
          <ng-template #footerTemplate>
            <span id="posvalue" style="float:left; padding-
            left:10px;">Position: "Left", "Top"</span>
          </ng-template>
        </ejs-dialog>
      `
})

```



```

export class AppComponent implements OnInit {
    @ViewChild('ejDialog') ejDialog: DialogComponent | any;
    // Create element reference for dialog target element.
    @ViewChild('container', { read: ElementRef, static: true }) container:
    ElementRef | any;
    // The Dialog shows within the target element.
    public targetElement?: HTMLElement;
    //To get all element of the dialog component after component get
    initialized.
    ngOnInit() {
        this.initilaizeTarget();
    }
    // Initialize the Dialog component's target element.
    public initilaizeTarget: EmitType<object> = () => {
        this.targetElement = this.container.nativeElement.parentElement;
    }
    // Set Dialog position
    public position: object={ X: 'left', Y: 'top' };
    // Disable the Esc key option to hide the Dialog
    public closeOnEscape: boolean =false;
    // Sample level code to handle the button click action
    public onOpenDialog = (event: any): void => {
        // Call the show method to open the Dialog
        this.ejDialog.show();
    }
    public changePosition = (event: any): void =>{
        this.ejDialog.position = { X: event.currentTarget.value.split(" ")[0],
        Y: event.currentTarget.value.split(" ")[1] };
        document.getElementById("posvalue")!.innerHTML = 'Position: {X: "' +
        event.currentTarget.value.split(" ")[0] + '", Y: "' +
        event.currentTarget.value.split(" ")[1] + '"}';
    }
}

```

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 Dialog](#) feature tour page for its groundbreaking feature representations. You can also explore our [Angular Dialog example](#) that shows how to render the dialog.

See Also

- [Load dialog content using AJAX](#)
- [How to position the dialog on center of the page on scrolling](#)
- [Prevent closing of modal dialog](#)
- [Close dialog while click on outside of dialog](#)
- [How to make a reusable alert and confirm dialog](#)

Template in Angular Dialog component

In Dialog the template support is provided to the header, content and footer sections. So any text or HTML content can be appending in these sections.

Header

The Dialog header content can be provided through the [header](#) property, and it will allow both text and any HTML content as a string.

Also in header, close button is provided as built-in support, and this can be enabled through the [showCloseIcon](#) property.

Footer

The Dialog footer can be enabled by adding built-in [buttons](#) or providing any HTML string through the [footerTemplate](#).

The [buttons](#) and [footerTemplate](#) properties can't be used at the same time.

Content

The Dialog content can be update by providing any HTML string through the [content](#).

The below example demonstrates the usage of header, footer and content as template in the Dialog.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DialogModule } from '@syncfusion/ej2-angular-popups'
import { Component, ViewChild, OnInit, ElementRef } from '@angular/core';
import { DialogComponent } from '@syncfusion/ej2-angular-popups';
import { EmitType, isNullOrUndefined } from '@syncfusion/ej2-base';
@Component({
  imports: [
    DialogModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `
    <button class="e-control e-btn" style="position: absolute;"
    id="targetButton" (click)="onOpenDialog($event)">Open Dialog</button>
    <div #container class='root-container'></div>
    <ejs-dialog id='dialog' #template showCloseIcon='true'
    (open)="dialogOpen()" [height]='height' [target]='targetElement'
    width='435px'>
      <ng-template #footerTemplate>
        <input id="inVal" class="e-input" type="text" placeholder="Enter
        your message here!" />
        <button id="sendButton" class="e-control e-btn e-primary
        sendButton" data-ripple="true">Send</button>
      </ng-template>
      <ng-template #content>
        <div class="dialogContent">
          <span class="dialogText">Greetings Nancy! When will you
          share me the source files of the project?</span>
        </div>
      </ng-template>
      <ng-template #header>
```

```

        
        <div title="Nancy" class="e-icon-settings dlg-template"> Nancy
</div>
    </ng-template>
</ejs-dialog>
`
}))
export class AppComponent implements OnInit {
    @ViewChild('template') template: DialogComponent | any;
    // Create element reference for dialog target element.
    @ViewChild('container', { read: ElementRef }) container: ElementRef |
any;
    // The Dialog shows within the target element.
    public targetElement?: HTMLElement;
    public proxy: any = this;
    //To get all element of the dialog component after component get
initialized.
    ngOnInit() {
        this.initilaizeTarget();
    }
    // Initialize the Dialog component target element.
    public initilaizeTarget: EmitType<object> = () => {
        this.targetElement = this.container.nativeElement.parentElement;
    }
    public height: string = '250px';
    public dialogOpen: EmitType<object> = () => {
        (document.getElementById('sendButton') as any).keypress = (e: any)
=> {
            if (e.keyCode === 13) { this.updateTextValue(); }
        };
        (document.getElementById('inVal') as HTMLElement).onkeydown = (e:
any) => {
            if (e.keyCode === 13) { this.updateTextValue(); }
        };
        document.getElementById('sendButton')!.onclick = (): void => {
            this.updateTextValue();
        };
    }
    public updateTextValue: EmitType<object> = () => {
        let enteredVal: HTMLInputElement = document.getElementById('inVal')
as HTMLInputElement;
        let dialogTextElement: HTMLElement =
document.getElementsByClassName('dialogText')[0] as HTMLElement;
        let dialogTextWrap : HTMLElement =
document.getElementsByClassName('dialogContent')[0] as HTMLElement;
        if
(!isNullOrUndefined(document.getElementsByClassName('contentText')[0])) {
            detach(document.getElementsByClassName('contentText')[0]);
        }
        if (enteredVal.value !== '') {
            dialogTextElement.innerHTML = enteredVal.value;
        }
        enteredVal.value = '';
    }
    // Sample level code to handle the button click action

```

```
public onOpenDialog = (event: any): void => {  
    // Call the show method to open the Dialog  
    this.template.show();  
}  
}  
function detach(arg0: Element) {  
    throw new Error('Function 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));
```

See Also

- [How to add an icon to dialog buttons](#)
- [How to customize the dialog appearance](#)

Animation in Angular Dialog component

The Dialog can be animated during the open and close actions. Also, user can customize animation's [delay](#), [duration](#) and [effect](#).

<!-- markdownlint-disable MD033 -->

delay	The Dialog animation will start with the mentioned delay
duration	Specifies the animation duration to complete with one animation cycle
effect	<p>Specifies the animation effects of Dialog open and close actions effect.</p> <p>List of supported animation effects: 'Fade' 'FadeZoom' 'FlipLeftDown' 'FlipLeftUp' 'FlipRightDown' 'FlipRightUp' 'FlipXDown' 'FlipXUp' 'FlipYLeft' 'FlipYRight' 'SlideBottom' 'SlideLeft' 'SlideRight' 'SlideTop' 'Zoom' 'None'</p> <p>If the user sets 'Fade' effect, then the Dialog will open with 'FadeIn' effect and close with 'FadeOut' effect</p>

In the below sample, **Zoom** effect is enabled. So, The Dialog will open with **ZoomIn** and close with **ZoomOut** effects.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core';  
import { BrowserModule } from '@angular/platform-browser';  
import { DialogModule } from '@syncfusion/ej2-angular-popups';  
import { Component, ViewChild, OnInit, ElementRef } from '@angular/core';  
import { DialogComponent } from '@syncfusion/ej2-angular-popups';
```

```

import { EmitType } from '@syncfusion/ej2-base';
@Component({
  imports: [

    DialogModule

  ],
  standalone: true,
  selector: 'app-root',
  template: `
    <button class="e-control e-btn" style="position: absolute;"
id="targetButton" (click)="onOpenDialog($event)">Open Dialog</button>
    <div #container class="root-container"></div>
    <ejs-dialog id='dialog' #ejDialog header='Dialog' content='Dialog
enabled with Zoom effect' [target]='targetElement'
    [animationSettings]='animationSettings' width='250px'
[buttons]='buttons'>
    </ejs-dialog>
  `
})
export class AppComponent implements OnInit {
  @ViewChild('ejDialog') ejDialog: DialogComponent | any;
  // Create element reference for dialog target element.
  @ViewChild('container', { read: ElementRef }) container: ElementRef |
any;
  // The Dialog shows within the target element.
  public targetElement?: HTMLElement;
  //To get all element of the dialog component after component get
initialized.
  ngOnInit() {
    this.initilaizeTarget();
  }
  // Initialize the Dialog component target element.
  public initilaizeTarget: EmitType<object> = () => {
    this.targetElement = this.container.nativeElement.parentElement;
  }
  // Hide the Dialog when click the footer button.
  public hideDialog: EmitType<object> = () => {
    this.ejDialog.hide();
  }
  // Sample level code to handle the button click action
  public onOpenDialog = (event:any): void => {
    // Call the show method to open the Dialog
    this.ejDialog.show();
  }
  //Animation options
  public animationSettings: Object = { effect: 'Zoom', duration: 400,
delay: 0 };
  // Enables the footer buttons
  public buttons: Object = [
    {
      'click': this.hideDialog.bind(this),buttonModel:{ content:'OK',
isPrimary: true }
    },
    {
      'click': this.hideDialog.bind(this),buttonModel:{
content:'Cancel' }
    }
  ]
}

```

```
    ];
  }
}
```

MAIN.TS

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

Resize in Angular Dialog component

The Dialog supports resizing feature. To resize the dialog, we have to select and resize it by using its handle (grip) or hovering on any of the edges or borders of the dialog within the sample container.

The resizable dialog can be created by setting the [enableResize](#) property to true, which is used to change the size of a dialog dynamically and view its content with expanded mode. The [resizeHandles](#) property can also be configured for all the which directions in which the dialog should be resized. When you configure the target property along with the [enableResize](#) property, the dialog can be resized within its specified target container.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DialogModule } from '@syncfusion/ej2-angular-popups'
import { Component, ViewChild, OnInit, ElementRef } from '@angular/core';
import { DialogComponent, ResizeDirections } from '@syncfusion/ej2-angular-popups';
import { EmitType } from '@syncfusion/ej2-base';
@Component({
  imports: [
    DialogModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `
    <button class="e-control e-btn" style="position: absolute;"
    id="targetButton" (click)="onOpenDialog($event)">Open Dialog</button>
    <div #container class='root-container'></div>
    <ejs-dialog id='dialog' #ejDialog allowDragging='true'
    enableResize='true' [resizeHandles]='resizeHandleDirection' header='Dialog'
    content='This is a Dialog with resize enabled'
    [target]='targetElement' width='250px' [buttons]='buttons'> </ejs-
    dialog>`
  })
export class AppComponent implements OnInit {
  @ViewChild('ejDialog') ejDialog: DialogComponent | any;
  // Create element reference for dialog target element.
  @ViewChild('container', { read: ElementRef }) container: ElementRef | any;
  // The Dialog shows within the target element.
  public targetElement?: HTMLElement;
  // This will resize the dialog in all the directions.
  public resizeHandleDirection: ResizeDirections[] = ['All'];
```

```

// To get all element of the dialog component after component get initialized.
ngOnInit() {
    this.initilaizeTarget();
}
// Initialize the Dialog component's target element.
public initilaizeTarget: EmitType<object> = () => {
    this.targetElement = this.container.nativeElement.parentElement;
    this.resizeHandleDirection = ['All'];
}
// Hide the Dialog when click the footer button.
public hideDialog: EmitType<object> = () => {
    this.ejDialog.hide();
}
// Enables the footer buttons
public buttons: Object = [
    {
        'click': this.hideDialog.bind(this),
        // Accessing button component properties by buttonModel property
        buttonModel: {
            content: 'OK',
            isPrimary: true
        }
    },
    {
        'click': this.hideDialog.bind(this),
        buttonModel: {
            content: 'Cancel'
        }
    }
];
// Sample level code to handle the button click action
public onOpenDialog = (event: any): void => {
    // Call the show method to open the Dialog
    this.ejDialog.show();
};
}

```

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 Dialog component

Localization library allows to localize the default text content of Dialog. In Dialog, The close button's tooltip text alone will be localize based on culture.

| Locale key | en-US (default) |

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

| close | Close |

Loading translations

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

In the below sample, `French` culture is set to Dialog and change the close button's tooltip text.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DialogModule } from '@syncfusion/ej2-angular-popups'
import { Component, ViewChild, OnInit, ElementRef } from '@angular/core';
import { DialogComponent } from '@syncfusion/ej2-angular-popups';
import { L10n } from '@syncfusion/ej2-base';
import { EmitType } from '@syncfusion/ej2-base';
@Component({
  imports: [

    DialogModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `
    <button class="e-control e-btn" style="position: absolute;"
    id="targetButton" (click)="onOpenDialog($event)">Open Dialog</button>
    <div #container class='root-container'></div>
    <ejs-dialog id='dialog' #ejDialog locale='fr-BE' showCloseIcon='true'
    header='Dialogue' content='Dialogue avec la culture française'
    [target]='targetElement' width='250px'>
      </ejs-dialog>
  `
})
export class AppComponent implements OnInit {
  @ViewChild('ejDialog') ejDialog: DialogComponent | any;
  // Create element reference for dialog target element.
  @ViewChild('container', { read: ElementRef }) container: ElementRef
  | any;
  // The Dialog shows within the target element.
  public targetElement?: HTMLElement;
  // Sample level code to handle the button click action
  public onOpenDialog = (event: any): void => {
    // Call the show method to open the Dialog
    this.ejDialog.show();
  }
  ngOnInit() {
    // Load French culture for Dialog close button tooltip text
    L10n.load({
      'fr-BE': {
        'dialog': {
          'close': "Fermer"
        }
      }
    });
    this.initilaizeTarget();
  }
  // Initialize the Dialog component target element.
```



```

    public initilaizeTarget: EmitType<object> = () => {
        this.targetElement = this.container.nativeElement.parentElement;
    }
}

```

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 Dialog component

The Dialog component followed the accessibility guidelines and standards, including [ADA, Section 508](#), [WCAG 2.2](#) standards, and [WCAG roles](#) that are commonly used to evaluate accessibility.

The accessibility compliance for the Dialog component is outlined below.

| Accessibility Criteria | Compatibility |

| -- | -- |

| [WCAG 2.2 Support](#) | |

| [Section 508 Support](#) | |

| [Screen Reader Support](#) | |

| [Right-To-Left Support](#) | |

| [Color Contrast](#) | |

| [Mobile Device Support](#) | |

| [Keyboard Navigation Support](#) | |

| [Accessibility Checker Validation](#) | |

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

<style>

.post .post-content img {

display: inline-block;

margin: 0.5em 0;

}

```
</style>
```

```
<div> - All features of the component meet the requirement.</div>
```

```
<div> - Some features of the component do not meet the requirement.</div>
```

```
<div> - The component does not meet the requirement.</div>
```

WAI-ARIA attributes

The Dialog is characterized with complete ARIA Accessibility support which helps to be accessible by on-screen readers and other assistive technology devices. This component is designed with the reference of the guidelines document given in [WAI ARIA Accessibility Practices](#).

The Dialog component uses the **Dialog** role and following ARIA properties to its element based on its state.

| Property | Functionalities |

| --- | --- |

| aria-describedby | It indicates the Dialog content description is notified to the user through assistive technologies. |

| aria-labelledby | The Dialog title is notified to the user through assistive technologies. |

| aria-modal | For modal dialog its value is true and non-modal dialog its value is false |

| aria-grabbed | Enable the draggable Dialog and starts dragging its value is true and stopping the drag its value is false |

Keyboard interaction

Keyboard interaction of Dialog component has been designed based on [WAI-ARIA Practices](#) described for Dialog.

User can use the following shortcut keys to interact with the Dialog.

```
<!-- markdownlint-disable MD033 -->
```

Keyboard shortcuts	Actions
Esc	Closes the Dialog. This functionality can be controlled by using closeOnEscape
Enter	When the Dialog button or any input (except text area) is in focus state, when pressing the Enter key, the click event associated with the primary button or button will trigger. Enter key is not working when the Dialog content contains any text area with initial focus
Ctrl + Enter	When the Dialog content contains text area and it is in focus state, and press the Ctrl + Enter key to call the click event function associated with primary button
Tab	Focus will be changed within the Dialog elements

Shift + Tab	The Focus will be changed previous focusable element within the Dialog elements. When focusing afirst focusable element in the Dialog, then press the shift + tab key, it will change the focusto last focusable element
-------------	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DialogModule } from '@syncfusion/ej2-angular-popups'
import { Component, ViewChild, OnInit, ElementRef } from '@angular/core';
import { DialogComponent } from '@syncfusion/ej2-angular-popups';
import { EmitType } from '@syncfusion/ej2-base';
@Component({
  imports: [

    DialogModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `
    <button class="e-control e-btn" style="position: absolute;"
    id="targetButton" (click)="onOpenDialog($event)">Open Dialog</button>
    <div #container class='root-container'></div>
    <ejs-dialog id='dialog' #ejDialog header='Feedback'
    [target]='targetElement'
    width='400px' [buttons]='buttons' showCloseIcon='true' height='300px'>
    <ng-template #content>
      <form>
        <div class='form-group'><label for='email'>Email:</label>
        <input type='email' class='form-control' id='email'>
        </div>
        <div class='form-group'></div>
        <div class='form-group'>
          <label for='comment'>Comments:</label>
          <textarea style='resize: none;' class='form-control'
            rows='5' id='comment'></textarea>
        </div>
      </form>
    </ng-template>
    </ejs-dialog>
  `
})
export class AppComponent implements OnInit {
  @ViewChild('ejDialog') ejDialog: DialogComponent | any;
  // Create element reference for dialog target element.
  @ViewChild('container', { read: ElementRef }) container: ElementRef
  | any;
  public targetElement?: HTMLElement;
  //To get all element of the dialog component after component get
  initialized.
  ngOnInit() {
    this.initilaizeTarget();
  }
  // Initialize the Dialog component target element.
  public initilaizeTarget: EmitType<object> = () => {

```

```

        this.targetElement = this.container.nativeElement.parentElement;
    }
    // Hide the Dialog when click the footer button.
    public hideDialog: EmitType<object> = () => {
        this.ejDialog.hide();
    }
    // Enables the footer buttons
    public buttons: Object = [
        {
            'click': this.hideDialog.bind(this),
            // Accessing button component properties by buttonModel property
            buttonModel: {
                content: 'Submit',
                isPrimary: true
            }
        }
    ];
    // Sample level code to handle the button click action
    public onOpenDialog = (event: any): void => {
        // Call the show method to open the Dialog
        this.ejDialog.show();
    }
}

```

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

- [Show dialog with full-screen](#)

Ensuring accessibility

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

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

See also

- [Accessibility in Syncfusion Angular components](#)

Render a dialog using utility functions in Angular Dialog component

The dialog component provides built-in utility functions to render the alert and confirm dialogs with the minimal code.

The following options are used as an argument on calling the utility functions:

Options	Description
---------	-------------

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

| title | Specifies the title of dialog like the [header](#) property. |

| content | Specifies the value that can be displayed in dialog's content area like the [content](#) property. |

| isModal | Specifies the Boolean value whether the dialog can be displayed as modal or non-modal. For more details, refer to the [isModal](#) property. |

| position | Specifies the value where the alert or confirm dialog is positioned within the document. For more details, refer to the [position](#) property { X: 'center', Y: 'center' } |

| okButton | Configures the OK button that contains button properties with the click events.
 okButton:{ icon:'prefix icon to the button', cssClass:'custom class to the button', click: 'action for OK button click', text: 'Yes' // <-- Default value is 'OK' } |

| cancelButton | Configures the Cancel button that contains button properties with the click events.
 cancelButton:{ icon:'prefix icon to the button', cssClass:'custom class to the button', click: 'action for 'Cancel' button click', text: 'No' // <-- Default value is 'Cancel' } |

| isDraggable | Specifies the value whether the alert or confirm dialog can be dragged by the user. |

| showCloseIcon | When set to true, the close icon is shown in the dialog component. |

| closeOnEscape | When set to true, you can close the dialog by pressing ESC key. |

| animationSettings | Specifies the animation settings of the dialog component. |

| cssClass | Specifies the CSS class name that can be appended to the dialog. |

| zIndex | Specifies the order of the dialog, that is displayed in front or behind of another component. |

| open | Event which is triggered after the dialog is opened. |

| Close | Event which is triggered after the dialog is closed. |

Alert dialog

An alert dialog box is used to display warning like messages to the users. Use the following code to render a simple alert dialog in an application.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Component, ViewChild, OnInit, ViewEncapsulation } from '@angular/core';
import { DialogUtility } from '@syncfusion/ej2-popups';
@Component({
  imports: [
    ],
  standalone: true,
  selector: 'app-root',
  template: `<button class="e-control e-btn" id="targetButton"
(click)="onOpenDialog($event)">Open Alert Dialog</button>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent implements OnInit {
  ngOnInit(): void {
```

```
        throw new Error('Method not implemented.');
```

```
    }  
    public onOpenDialog = function(event: any): void {  
        DialogUtility.alert('This is an Alert Dialog!');
```

```
    }  
}
```

MAIN.TS

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

Render an alert dialog with options

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'  
import { BrowserModule } from '@angular/platform-browser'  
import { Component, ViewChild, OnInit } from '@angular/core';  
import { DialogUtility } from '@syncfusion/ej2-popups';  
@Component({  
  imports: [  
  
    ],  
  standalone: true,  
  selector: 'app-root',  
  template: `<button class="e-control e-btn" id="targetButton"  
(click)="onOpenDialog($event)">Open Alert Dialog</button>`  
})  
export class AppComponent implements OnInit {  
  ngOnInit(): void {  
    throw new Error('Method not implemented.');  }  
  public onOpenDialog = (event: any): void => {  
    DialogUtility.alert({  
      title: 'Alert Dialog',  
      content: "This is an Alert Dialog!",  
      okButton: { text: 'OK', click: this.okClick.bind(this) },  
      showCloseIcon: true,  
      closeOnEscape: true,  
      animationSettings: { effect: 'Zoom' }  
    });  
  }  
  private okClick(): void {  
    alert('you clicked OK button');  }  
}
```

MAIN.TS

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

Confirm dialog

A confirm dialog displays a specified message along with 'OK' and 'Cancel' button.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Component, ViewChild, OnInit } from '@angular/core';
import { DialogUtility } from '@syncfusion/ej2-popups';
@Component({
  imports: [

    ],
  standalone: true,
  selector: 'app-root',
  template: `<button class="e-control e-btn" id="targetButton"
(click)="onOpenDialog($event)">Open Confirm Dialog</button>`
})
export class AppComponent implements OnInit {
  ngOnInit(): void {
    throw new Error('Method not implemented.');
```

```
    public onOpenDialog = function(event: any): void {
      DialogUtility.confirm('This is a Confirmation Dialog!');
    }
  }
}
```

MAIN.TS

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

Render a confirmation dialog with options

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Component, ViewChild, OnInit } from '@angular/core';
import { DialogUtility } from '@syncfusion/ej2-popups';
@Component({
  imports: [

    ],
  standalone: true,
  selector: 'app-root',
  template: `<button class="e-control e-btn" id="targetButton"
(click)="onOpenDialog($event)">Open Confirm Dialog</button>`
})
export class AppComponent implements OnInit {
  ngOnInit(): void {
    throw new Error('Method not implemented.');
```

```
  }
}
```

```
public onOpenDialog = (event: any): void => {
  DialogUtility.confirm({
    title: ' Confirmation Dialog',
    content: "This is a Confirmation Dialog!",
    okButton: { text: 'OK', click: this.okClick.bind(this) },
    cancelButton: { text: 'Cancel', click: this.cancelClick.bind(this)
  },
  showCloseIcon: true,
  closeOnEscape: true,
  animationSettings: { effect: 'Zoom' }
  });
}
private okClick(): void {
  alert('you clicked OK button');
}
private cancelClick(): void {
  alert('you clicked Cancel button');
}
}
```

MAIN.TS

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

Close utility dialog

When rendering an Alert and Confirmation dialog through utility methods, You can close the dialog using the following ways.

- By pressing the escape key if the [closeOnEscape](#) property is enabled.
- By clicking the close button if the [showCloseIcon](#) property is enabled.

You can also manually close the Dialogs by creating an instance to the dialog and call the [hide](#) method.

Below sample demonstrates the different ways of hiding the utility dialog.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Component, ViewChild, OnInit } from '@angular/core';
import { DialogUtility } from '@syncfusion/ej2-popups';
@Component({
  imports: [
    ],
  standalone: true,
  selector: 'app-root',
  template: `<button class="e-control e-btn" id="targetButton"
(click)="onOpenDialog($event)">Open Confirm Dialog</button>`
})
export class AppComponent implements OnInit {
  ngOnInit(): void {
```



```

        throw new Error('Method not implemented.');
```

```

    }
    public DialogObj: any;
    public onOpenDialog = (event: any): void => {
        this.DialogObj = DialogUtility.confirm({
            title: 'Confirmation Dialog',
            content: "This is a Confirmation Dialog!",
            okButton: { text: 'OK', click: this.okClick.bind(this) },
            cancelButton: { text: 'Cancel', click: this.cancelClick.bind(this)
        },
        showCloseIcon: true,
        closeOnEscape: true,
        animationSettings: { effect: 'Zoom' }
    });
    }
    private okClick(): void {
        alert('you clicked OK button');
    }
    private cancelClick(): void {
        //Hide the dialog
        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));

```

Style in Angular Dialog component

The following content provides the exact CSS structure that can be used to modify the control's appearance based on the user preference.

Customizing the dialog header

Use the following CSS to customize the dialog header properties.

```

`CSS

.e-dialog .e-dlg-header {
color: green;
font-size: 20px;
font-weight: normal;
}
`

```

Customizing the dialog content

Use the following CSS to customize the dialog content properties.

```

`CSS

```

```
.e-dialog .e-dlg-content {  
color: red;  
font-size: 10px;  
font-weight: normal;  
line-height: normal;  
}  
,
```

Customizing modal dialog overlay

Use the following CSS to customize the modal dialog overlay.

```
`CSS  
.e-dlg-overlay {  
background-color: slategray;  
opacity: 0.6;  
}  
,
```

Customizing the dialog resize icon

Use the following CSS to customize the dialog resize icon.

```
`CSS  
  
/ To change the icon content /  
.e-dialog .e-south-east::before, .e-dialog .e-south-west::before {  
content: '\f047';  
}  
  
/ To set the icon pack /  
.e-dialog .e-resize-handle {  
font: normal normal normal 14px/1 FontAwesome;  
}  
,
```

The above CSS demonstration uses the font awesome icon.

Customizing the dialog close button

Use the following CSS to customize the dialog close button.

```
`CSS  
  
/ To specify font size and color /  
.e-dialog .e-btn .e-btn-icon.e-icon-dlg-close {  
font-size: 12px;
```

```
color: red;
}
`
```

Customizing the dialog footer button

Use the following CSS to customize the dialog footer button.

```
`CSS
/ To specify font color, background color and border color /
.e-btn.e-flat.e-primary, .e-css.e-btn.e-flat.e-primary {
background-color: transparent;
border-color: transparent;
color: blue;
}
`
```

How to

Create nested dialog in Angular Dialog component

A Dialog can be nested within another Dialog. The below sample contains parent and child Dialog (inner Dialog).

Step 1:

Create two div elements with id `#dialog` and `#innerDialog`.

Step 2:

Initialize the Dialog as mentioned in the below sample.

Step 3:

Set the inner Dialog target as `#dialog`.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DialogModule } from '@syncfusion/ej2-angular-popups'
import { Component, ViewChild, OnInit, ElementRef } from '@angular/core';
import { DialogComponent } from '@syncfusion/ej2-angular-popups';
import { EmitType } from '@syncfusion/ej2-base';
@Component({
  imports: [
    DialogModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `
<button class="e-control e-btn" style="position: absolute;"
id="targetButton" (click)="onOpenDialog($event)">Open Dialog</button>
<div #container class='root-container'></div>
```

```

    <ejs-dialog id='dialog' #ejDialog header='Outer Dialog' height='300px'
width='400px' showOnInit='true' showCloseIcon='true' [content]='content'
[target]='targetElement' [animationSettings]='dialogAnimation'
[closeOnEscape]='closeOnEscape'>
    <button class="e-control e-btn" id="innerButton"
(click)="onOpenInnerDialog($event)" style='margin-left: 18px;'>Open
InnerDialog</button>
    </ejs-dialog>
    <ejs-dialog id='innerDialog' #ejInnerDialog header='Inner Dialog'
height='150px' width='250px' showOnInit='true' showCloseIcon='true'
content='This is a Nested Dialog' target='#dialog'
[animationSettings]='dialogAnimation' [closeOnEscape]='closeOnEscape'>
    </ejs-dialog>
})
export class AppComponent implements OnInit {
    @ViewChild('ejDialog') ejDialog: DialogComponent | any;
    @ViewChild('ejInnerDialog') ejInnerDialog: DialogComponent | any;
    // Create element reference for dialog target element.
    @ViewChild('container', { read: ElementRef }) container: ElementRef
|any;
    // The Dialog shows within the target element.
    public targetElement?: HTMLElement;
content: any;
    //To get all element of the dialog component after component get
initialized.
    ngOnInit() {
        this.initilaizeTarget();
    }
    // Initialize the Dialog component target element.
    public initilaizeTarget: EmitType<object> = () => {
        this.targetElement = this.container.nativeElement.parentElement;
    }
    // Dialog animation
    public dialogAnimation: Object={effect: 'None'};
    // Disable the Esc key option to hide the Dialog
    public closeOnEscape: boolean =false;
    // Dialog header template content
    public header: string='<div title="Installation Complete" class="e-icon-
settings e-icons" style="padding: 3px;"> Installation Complete</div>';
    // Dialog footer template content
    public footerTemplate: string='<span style="float: left;font-size:
14px;padding-left: 15px;color: rgba(0, 0, 0, 0.54);">Click the close button
to Exit</span>';
    // Sample level code to handle the button click action
    public onOpenDialog = (event: any): void => {
        this.ejDialog.show();
    }
    public onOpenInnerDialog = (event: any): void => {
        this.ejInnerDialog.show();
    }
}

```

MAIN.TS

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

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

Position the dialog on center of the page on scrolling in Angular Dialog component

By default, when scroll the page/container Dialog also scrolled along with the page/container.

When a user expects to display the Dialog in the same position without scrolling achieving this in sample level as like below. Here added 'e-fixed' class to Dialog element and prevent the scrolling.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DialogModule } from '@syncfusion/ej2-angular-popups'
import { Component, ViewChild, OnInit, ElementRef } from '@angular/core';
import { DialogComponent } from '@syncfusion/ej2-angular-popups';
import { EmitType } from '@syncfusion/ej2-base';
@Component({
  imports: [

    DialogModule
  ],
  standalone: true,
  selector: 'app-root',
  template: ` <div #container class='root-container'></div>
    <div>
      <b>JavaScript:</b><br />
      JavaScript is a high-level, dynamic, untyped, and interpreted
programming language. It has been standardized in the ECMAScript
      language specification. Alongside HTML and CSS, it is one of the
three essential technologies of World Wide Web
      content production; the majority of websites employ it and it is
supported by all modern Web browsers without
      plug-ins. JavaScript is prototype-based with first-class
functions, making it a multi-paradigm language, supporting
      object - oriented , imperative, and functional programming
styles.
      <br /><br /><br />
      <b>MVC:</b><br />
      Model-view-controller (MVC) is a software architecture pattern
which separates the representation of information from the user's
interaction with it. The model consists of application data, business rules,
logic, and functions. A view can be any output representation of data, such
as a chart or a diagram. Multiple views of the same data are possible, such
as a bar chart for management and a tabular view for accountants. The
controller mediates input, converting it to commands for the model or
view.The central ideas behind MVC are code reusability and in addition to
dividing the application into three kinds of components, the MVC design
defines the interactions between them.
      A controller can send commands to its associated view to change
the view's presentation of the model (e.g., by scrolling through a
document). It can also send commands to the model to update the model's
state (e.g., editing a document).
      A model notifies its associated views and controllers when there
has been a change in its state. This notification allows the views to
```

produce updated output, and the controllers to change the available **set** of commands. A passive implementation of MVC omits these notifications, because the application does not require them or the software platform does not support them.

A view requests **from** the model the information that it needs to generate an output representation to the user.

```

</div>
<ejs-dialog id='dialog' #ejDialog header='Dialog' showOnInit='true'
[target]='targetElement' width='250px' [closeOnEscape]='closeOnEscape'>
  <div style='padding:18px;padding-top:0px;'>
    <button class="e-control e-btn" id="targetButton" role="button"
(click)="onPreventScroll($event)">Prevent Dialog Scroll</button>
  </div>
</ejs-dialog>
`
))
export class AppComponent implements OnInit {
  @ViewChild('ejDialog') ejDialog: DialogComponent | any;
  // Create element reference for dialog target element.
  @ViewChild('container', { read: ElementRef }) container: ElementRef |
any;
  // The Dialog shows within the target element.
  public targetElement?: HTMLElement;
  //To get all element of the dialog component after component get
initialized.
  ngOnInit() {
    this.initilaizeTarget();
  }
  // Initialize the Dialog component target element.
  public initilaizeTarget: EmitType<object> = () => {
    this.targetElement = this.container.nativeElement.parentElement;
  }
  // Disable the Esc key option to hide the Dialog
  public closeOnEscape: boolean =false;
  // Add e-fixed class to Dialog element
  public onPreventScroll = (event: any): void => {
    this.ejDialog.cssClass='e-fixed';
  }
  // Dialog content
  public content:string='<button class="e-control e-btn" id="targetButton"
role="button">Prevent Dialog Scroll</button>';
}

```

MAIN.TS

```

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

```

Load dialog content using ajax in Angular Dialog component

You can load dialog's content dynamically from external source like external file using AJAX library.

The AJAX library can make the request and load dialog's content using its **success** event.

Refer the following link to learn about how to load dialog content using AJAX.

[AJAX Content](#)

Render a dialog without header in Angular Dialog component

The dialog can be rendered without header by setting the header property value as empty string or null. By default, dialog is rendered without header.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DialogModule } from '@syncfusion/ej2-angular-popups'
import { Component, ViewChild, OnInit, ElementRef } from '@angular/core';
import { DialogComponent } from '@syncfusion/ej2-angular-popups';
import { EmitType } from '@syncfusion/ej2-base';
@Component({
  imports: [

    DialogModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `
    <button class="e-control e-btn" style="position: absolute;"
    id="targetButton" (click)="onOpenDialog($event)">Open Dialog</button>
    <div #container class='root-container'></div>
    <ejs-dialog id='dialog' #ejDialog content='This is a dialog without
    header' [target]='targetElement'
    width='250px' [buttons]='buttons'>
    </ejs-dialog>`
})
export class AppComponent implements OnInit {
  @ViewChild('ejDialog') ejDialog: DialogComponent | undefined;
  // Create element reference for dialog target element.
  @ViewChild('container', { read: ElementRef }) container: ElementRef |
  undefined;
  // The Dialog shows within the target element.
  public targetElement?: HTMLElement;
  //To get all element of the dialog component after component get
  initialized.
  ngOnInit() {
    this.initilaizeTarget();
  }
  // Initialize the Dialog component's target element.
  public initilaizeTarget: EmitType<object> = () => {
    this.targetElement = this.container!.nativeElement.parentElement;
  }
  // Hide the Dialog when click the footer button.
  public hideDialog: EmitType<object> = () => {
    this.ejDialog!.hide();
  }
  // Enables the footer buttons
  public buttons: Object = [
    {
      'click': this.hideDialog.bind(this),
      // Accessing button component properties by buttonModel property
    }
  ]
}
```

```

        buttonModel:{
            content:'OK',
            //Enables the primary button
            isPrimary: true
        }
    },
    {
        'click': this.hideDialog.bind(this),
        buttonModel:{
            content:'Cancel'
        }
    }
];
// Sample level code to handle the button click action
public onOpenDialog = (event: any): void => {
    // Call the show method to open the Dialog
    this.ejDialog!.show();
}
}

```

MAIN.TS

```

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

```

Show dialog with full screen in Angular Dialog component

You can show the dialog in fullscreen by passing `true` as argument to the dialog [show](#) method.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DialogModule } from '@syncfusion/ej2-angular-popups'
import { Component, ViewChild, OnInit, ElementRef } from '@angular/core';
import { DialogComponent } from '@syncfusion/ej2-angular-popups';
import { EmitType } from '@syncfusion/ej2-base';
@Component({
  imports: [

      DialogModule

  ],
  standalone: true,
  selector: 'app-root',
  template: `
    <button class="e-control e-btn" style="position: absolute;"
    id="targetButton" (click)="onOpenDialog($event)">Open Dialog</button>
    <div #container class='root-container'></div>
    <ejs-dialog id='dialog' #ejDialog header='Dialog' showCloseIcon='true'
    [visible]='visible' content='This is a dialog with fullscreen display'
    [target]='targetElement'
    width='250px' [buttons]='buttons'>
    </ejs-dialog> `
})

```



```

export class AppComponent implements OnInit {
  @ViewChild('ejDialog') ejDialog: DialogComponent | undefined;
  // Create element reference for dialog target element.
  @ViewChild('container', { read: ElementRef }) container: ElementRef |
  undefined;
  // The Dialog shows within the target element.
  public targetElement?: HTMLElement;
  //To get all element of the dialog component after component get
  initialized.
  ngOnInit() {
    this.initilaizeTarget();
  }
  // Initialize the Dialog component's target element.
  public initilaizeTarget: EmitType<object> = () => {
    this.targetElement = this.container!.nativeElement.parentElement;
  }
  public visible: Boolean = false;
  // Hide the Dialog when click the footer button.
  public hideDialog: EmitType<object> = () => {
    this.ejDialog!.hide();
  }
  // Enables the footer buttons
  public buttons: Object = [
    {
      'click': this.hideDialog.bind(this),
      // Accessing button component properties by buttonModel property
      buttonModel:{
        content:'OK',
        //Enables the primary button
        isPrimary: true
      }
    },
    {
      'click': this.hideDialog.bind(this),
      buttonModel:{
        content:'Cancel'
      }
    }
  ];
  // Sample level code to handle the button click action
  public onOpenDialog = (event: any): void => {
    // Call the show method to open the Dialog
    this.ejDialog!.show(true);
  }
}

```

MAIN.TS

```

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

```

Display a dialog with custom position in Angular Dialog component

By default, the dialog is displayed in the center of the target container. The dialog position can be set using the position property by providing custom X and Y coordinates.

The dialog can be positioned inside the target based on the given X and Y values.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DialogModule } from '@syncfusion/ej2-angular-popups'
import { Component, ViewChild, OnInit, ElementRef } from '@angular/core';
import { DialogComponent } from '@syncfusion/ej2-angular-popups';
import { EmitType } from '@syncfusion/ej2-base';
@Component({
  imports: [

    DialogModule

  ],
  standalone: true,
  selector: 'app-root',
  template: `
    <div #container class='root-container'>
      <ejs-dialog id='firstDialog' #ejDialog header='Position-01'
content='The dialog is positioned at {X: 160, Y: 14 } coordinates.'
[target]='targetElement'
      width='360px' height='120px' [position]='position1'>
      </ejs-dialog>
      <ejs-dialog id='secondDialog' #ejDialog1 header='Position-02'
content='The dialog is positioned at {X: 160, Y: 240} coordinates.'
[target]='targetElement'
      width='360px' height='120px' [position]='position2'>
      </ejs-dialog>
    </div> `
})
export class AppComponent implements OnInit {
  @ViewChild('ejDialog') ejDialog: DialogComponent | undefined;
  @ViewChild('ejDialog1') ejDialog1: DialogComponent | undefined;
  // Create element reference for dialog target element.
  @ViewChild('container', { read: ElementRef }) container: ElementRef |
undefined;
  // The Dialog shows within the target element.
  public targetElement?: HTMLElement;
  //To get all element of the dialog component after component get
initialized.
  ngOnInit() {
    this.initilaizeTarget();
  }
  // Set Dialog position
  public position1: object={ X: 160, Y: 14 };
  public position2: object={ X: 160, Y: 240 };
  // Initialize the Dialog component's target element.
  public initilaizeTarget: EmitType<object> = () => {
    this.targetElement = this.container!.nativeElement.parentElement;
  }
}
```

MAIN.TS

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

Prevent closing of modal dialog in Angular Dialog component

You can prevent closing of modal dialog by setting the [beforeClose](#) event argument cancel value to true.

In the following sample, the dialog is closed when you enter the username value with minimum 4 characters. Otherwise, it will not be closed.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DialogModule } from '@syncfusion/ej2-angular-popups'
import { Component, ViewChild, OnInit, ElementRef } from '@angular/core';
import { DialogComponent } from '@syncfusion/ej2-angular-popups';
import { EmitType } from '@syncfusion/ej2-base';
@Component({
  imports: [

    DialogModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `
    <button class="e-control e-btn" style="position: absolute;"
    id="targetButton" (click)="onOpenDialog($event)">Open Dialog</button>
    <div #container class="root-container">
      <ejs-dialog id="dialog" #ejDialog isModal='true' header='Sign in'
      (beforeClose)="validation($event)" [buttons]='buttons'
      [target]='targetElement' width='300px'>
        <ng-template #content>
          <div className="login-form">
            <div class="wrap">
              <div class="e-float-input e-input-group">
                <input id="textvalue" type="text" required
                (focus)="focusIn($event.target)" (blur)="focusOut($event.target)" />
                <span class="e-float-line"></span>
                <label class="e-float-text">Username</label>
              </div>
              <div class="e-float-input e-input-group">
                <input id="textvalue2" type="password" required
                (focus)="focusIn($event.target)" (blur)="focusOut($event.target)" />
                <span class="e-float-line"></span>
                <label class="e-float-text">Password</label>
              </div>
            </div>
          </div>
        </ng-template>
      </ejs-dialog>
    </div> `
})
```

```

export class AppComponent {
  @ViewChild('ejDialog') ejDialog: DialogComponent | undefined;
  // The Dialog shows within the target element.
  @ViewChild('container', { read: ElementRef }) container: ElementRef |
undefined;
  // The Dialog shows within the target element.
  public targetElement?: HTMLElement;
  //To get all element of the dialog component after component get
initialized.
  ngOnInit() {
    this.initilaizeTarget();
  }
  public focusIn(target: HTMLElement |any): void {
    let parent: HTMLElement = target.parentElement as HTMLElement;
    if (parent.classList.contains('e-input-in-wrap')) {
      parent.parentElement!.classList.add('e-input-focus');
    } else {
      parent.classList.add('e-input-focus');
    }
  }
  public focusOut(target: HTMLElement| any): void {
    let parent: HTMLElement = target.parentElement as HTMLElement;
    if (parent.classList.contains('e-input-in-wrap')) {
      parent.parentElement!.classList.remove('e-input-focus');
    } else {
      parent.classList.remove('e-input-focus');
    }
  }
  // Hide the Dialog when click the footer button.
  public hideDialog: EmitType<object> = () => {
    this.ejDialog!.hide();
  }
  // Enables the footer buttons
  public buttons: Object = [
    {
      'click': this.hideDialog.bind(this),
      // Accessing button component properties by buttonModel property
      buttonModel:{
        content:'Log in',
        //Enables the primary button
        isPrimary: true
      }
    }
  ];
  // Initialize the Dialog component target element.
  public initilaizeTarget: EmitType<object> = () => {
    this.targetElement = this.container!.nativeElement.parentElement;
  }
  public validation (event: any): void {
    let text = document.getElementById('textvalue');
    let text1 = document.getElementById('textvalue2');
    if ((text as any).value === "" && (text1 as any).value === "") {
      event.cancel= true;
      alert("Enter the username and password")
    } else if ((text as any).value === "") {
      event.cancel= true;
      alert("Enter the username")
    }
  }
}

```

```

    } else if ((text1 as any).value === "") {
        event.cancel= true;
        alert("Enter the password")
    } else if ((text as any).value.length < 4) {
        event.cancel= true;
        alert("Username must be minimum 4 characters")
    } else {
        event.cancel= false;
        (document.getElementById("textvalue")! as any).value = "";
        (document.getElementById("textvalue2")! as any).value = "";
    }
}
// Sample level code to handle the button click action
public onOpenDialog = (event: any): void => {
    // Call the show method to open the Dialog
    this.ejDialog!.show();
}
}

```

MAIN.TS

```

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

```

Prevent the focus on the first element in Angular Dialog component

By default, the dialog focuses on the first elements of the content area which can be active and focusable. You can prevent this default focusing behavior using the [open](#) event and by enabling the `preventFocus` argument.

Bind the open event and enable the `preventFocus` argument within an event like the below sample.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DialogModule } from '@syncfusion/ej2-angular-popups'
import { Component, ViewChild, OnInit, ElementRef } from '@angular/core';
import { DialogComponent } from '@syncfusion/ej2-angular-popups';
import { EmitType } from '@syncfusion/ej2-base';
@Component({
  imports: [

      DialogModule

  ],
  standalone: true,
  selector: 'app-root',
  template: `
    <button class="e-control e-btn" style="position: absolute;"
    id="targetButton" (click)="onOpenDialog($event)">Open Dialog</button>
    <div #container class='root-container'></div>
    <ejs-dialog id='dialog' #ejDialog header='Sign In' [buttons]='buttons'
    [target]='targetElement' width='300px' (open)="onOpen($event)">
      <ng-template #content>

```

```

        <div class='form-group'><label for='email'>Email:</label>
          <input type='email' class='form-control' id='email'>
        </div>
        <div class='form-group'>
          <label for='comment'>Password:</label>
          <input type='password' class='form-control' id='password'>
        </div>
      </ng-template>
    </ejs-dialog>`
  })
  export class AppComponent implements OnInit {
    @ViewChild('ejDialog') ejDialog: DialogComponent | undefined;
    // Create element reference for dialog target element.
    @ViewChild('container', { read: ElementRef }) container: ElementRef |
    undefined;
    // The Dialog shows within the target element.
    public targetElement?: HTMLElement;
    // To get all element of the dialog component after component get
    initialized.
    ngOnInit() {
      this.initilaizeTarget();
    }
    // Initialize the Dialog component target element.
    public initilaizeTarget: EmitType<object> = () => {
      this.targetElement = this.container!.nativeElement.parentElement;
    }
    // Sample level code to handle the button click action
    public onOpenDialog = (event: any): void => {
      // Call the show method to open the Dialog
      this.ejDialog!.show();
    };
    public onOpen(args: any): void {
      //Preventing the default dialog focus
      args.preventDefault = true;
    }
    // Hide the Dialog when click the footer button.
    public hideDialog: EmitType<object> = () => {
      this.ejDialog!.hide();
    }
    // Enables the footer buttons
    public buttons: Object = [
      {
        'click': this.hideDialog.bind(this),
        // Accessing button component properties by buttonModel property
        buttonModel: {
          content: 'OK',
          // Enables the primary button
          isPrimary: true
        }
      },
      {
        'click': this.hideDialog.bind(this),
        buttonModel: {
          content: 'Cancel'
        }
      }
    ]
  };

```

```
}

```

MAIN.TS

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

Prevent opening of the dialog in Angular Dialog component

You can prevent opening of the dialog by setting the [beforeOpen](#) event argument cancel value to true.

In the following sample, the success dialog is opened when you enter the username value with minimum 4 characters. Otherwise, it will not be opened.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DialogModule } from '@syncfusion/ej2-angular-popups'
import { Component, ViewChild, OnInit, ElementRef } from '@angular/core';
import { DialogComponent } from '@syncfusion/ej2-angular-popups';
import { EmitType } from '@syncfusion/ej2-base';
@Component({
  imports: [

    DialogModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `
<div class="login-form">
  <div class="wrap">
    <div id="heading">Sign in</div>
    <div class="e-float-input e-input-group">
      <input id="textvalue" type="text" required
(focus)="focusIn($event.target)" (blur)="focusOut($event.target)" />
      <span class="e-float-line"></span>
      <label class="e-float-text">Username</label>
    </div>
    <div class="e-float-input e-input-group">
      <input id="textvalue2" type="password" required
(focus)="focusIn($event.target)" (blur)="focusOut($event.target)" />
      <span class="e-float-line"></span>
      <label class="e-float-text">Password</label>
    </div>
    <div class="button-contain">
      <button class="e-control e-btn e-info" id="targetButton"
(click)="onOpenDialog($event)">Log in</button>
    </div>
  </div>
</div>
<div #container class='root-container'>
  <ejs-dialog id='dialog' #ejDialog isModal='true'
content='Congratulations! Login Success' [visible]='visible'
```

```

header='Success' (beforeOpen)="validation($event)"[buttons]='buttons'
[target]='targetElement' width='280px'>
    </ejs-dialog>
</div> `
}))
export class AppComponent {
    @ViewChild('ejDialog') ejDialog: DialogComponent | undefined;
    // The Dialog shows within the target element.
    @ViewChild('container', { read: ElementRef }) container: ElementRef |
undefined;
    // The Dialog shows within the target element.
    public targetElement?: HTMLElement;
    //To get all element of the dialog component after component get
initialized.
    ngOnInit() {
        this.initilaizeTarget();
    }
    public focusIn(target: HTMLElement| any): void {
        let parent: HTMLElement = target.parentElement;
        if (parent.classList.contains('e-input-in-wrap')) {
            parent.parentElement!.classList.add('e-input-focus');
        } else {
            parent.classList.add('e-input-focus');
        }
    }
    public focusOut(target: HTMLElement |any): void {
        let parent: HTMLElement = target.parentElement;
        if (parent.classList.contains('e-input-in-wrap')) {
            parent.parentElement!.classList.remove('e-input-focus');
        } else {
            parent.classList.remove('e-input-focus');
        }
    }
    public visible: Boolean = false;
    // Hide the Dialog when click the footer button.
    public hideDialog: EmitType<object> = () => {
        this.ejDialog!.hide();
    }
    // Enables the footer buttons
    public buttons: Object = [
        {
            'click': this.hideDialog.bind(this),
            // Accessing button component properties by buttonModel property
            buttonModel:{
                content:'Dismiss',
                //Enables the primary button
                isPrimary: true
            }
        }
    ];
    // Initialize the Dialog component target element.
    public initilaizeTarget: EmitType<object> = () => {
        this.targetElement = this.container!.nativeElement.parentElement;
    }
    public validation (event: any): void {
        let text = document.getElementById('textvalue');
        let text1 = document.getElementById('textvalue2');

```



```

        if ((text as any).value === "" && (text1 as any).value === "") {
            event.cancel= true;
            alert("Enter the username and password")
        } else if ((text as any).value === "") {
            event.cancel= true;
            alert("Enter the username")
        } else if ((text1 as any).value === "") {
            event.cancel= true;
            alert("Enter the password")
        } else if ((text as any).value.length < 4) {
            event.cancel= true;
            alert("Username must be minimum 4 characters")
        } else {
            event.cancel= false;
            (document.getElementById("textvalue")! as any).value = "";
            (document.getElementById("textvalue2")! as any).value = "";
        }
    }
    // Sample level code to handle the button click action
    public onOpenDialog = (event: any): void => {
        // Call the show method to open the Dialog
        this.ejDialog!.show();
    }
}

```

MAIN.TS

```

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

```

Read all the values from dialog on button click in Angular Dialog component

You can read the dialog element values by binding the action handler to the footer buttons. The buttons property provides the options to bind events to the action buttons.

For detailed information about buttons , refer to the [footer](#) section.

In the below sample, value of input elements within the dialog has been checked in the footer button click event and send the values as the content of confirmation dialog.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DialogModule } from '@syncfusion/ej2-angular-popups'
import { Component, ViewChild, OnInit, ElementRef } from '@angular/core';
import { DialogComponent } from '@syncfusion/ej2-angular-popups';
import { EmitType } from '@syncfusion/ej2-base';
@Component({
  imports: [

      DialogModule

  ],
  standalone: true,
  selector: 'app-root',

```

```

    template: `
    <button class="e-control e-btn" style="position: absolute;"
id="targetButton" (click)="onOpenDialog($event)">Open Dialog</button>
    <div #container class='root-container'>
        <ejs-dialog id='dialog' #ejDialog header='User details' width='400px'
[buttons]='buttons' [visible]='visible1' showCloseIcon='true'
[target]='targetElement' [animationSettings]='dialogAnimation'
[closeOnEscape]='closeOnEscape'>
            <ng-template #content>
                <form>
                    <div class='form-group'>
                        <label for='name'>Name:</label>
                        <input type='name' class='form-control' id='name' />
                    </div>
                    <div class='form-group'>
                        <label for='email'>Email Id:</label>
                        <input type='email' placeholder='user@syncfusion.com'
class='form-control' id='email' />
                    </div>
                    <div class='form-group'>
                        <label for='contact'>Contact Number:</label>
                        <input type='contact' class='form-control' id='contact' />
                    </div>
                    <div class='form-group'><label for='address'>Address:</label>
                        <textarea class='form-control' rows='2'
id='address'></textarea>
                    </div>
                </form>
            </ng-template>
        </ejs-dialog>
    </div>
    <ejs-dialog id='innerDialog' #ejInnerDialog header='User details'
[animationSettings]='animationSettings' [isModal]='isModal' width='400px'
[visible]='visible' showCloseIcon='true' content='This is a Nested Dialog'
[target]='targetElement' [animationSettings]='dialogAnimation'
[closeOnEscape]='closeOnEscape'>
    </ejs-dialog> </div> `
    })
    export class AppComponent implements OnInit {
        @ViewChild('ejDialog') ejDialog: DialogComponent | undefined;
        @ViewChild('ejInnerDialog') ejInnerDialog: DialogComponent | undefined;
        // Create element reference for dialog target element.
        @ViewChild('container', { read: ElementRef }) container: ElementRef |
        undefined;
        // The Dialog shows within the target element.
        public targetElement?: HTMLElement;
        //To get all element of the dialog component after component get
        initialized.
        ngOnInit() {
            this.initilaizeTarget();
        }
        public isModal: boolean = true;
        public visible: boolean = false;
        public visible1: boolean = false;
        // Initialize the Dialog component target element.
        public initilaizeTarget: EmitType<object> = () => {
            this.targetElement = this.container!.nativeElement.parentElement;
        }
    }

```

```

    // Dialog animation
    public dialogAnimation: Object= { effect: 'Zoom', duration: 400, delay:
0 };
    public animationSettings: Object = { effect: 'Zoom', duration: 400,
delay: 0 };
    // Disable the Esc key option to hide the Dialog
    public closeOnEscape: boolean =false;
    // Dialog header template content
    header: string='<div title="Installation Complete" class="e-icon-
settings e-icons" style="padding: 3px;"> Installation Complete</div>';
    // Dialog footer template content
    footerTemplate: string='<span style="float: left;font-size:
14px;padding-left: 15px;color: rgba(0, 0, 0, 0.54);">Click the close button
to Exit</span>';
    // Sample level code to handle the button click action
    public onOpenDialog = (event: any): void => {
        this.ejDialog!.show();
    }
    public getDynamicContent: EmitType<object> = () => {
        let input: HTMLInputElement =
document.getElementById('dialog')!.querySelector('#name') as
HTMLInputElement;
        let email: HTMLInputElement =
document.getElementById('dialog')!.querySelector('#email') as
HTMLInputElement;
        let contact: HTMLInputElement =
document.getElementById('dialog')!.querySelector('#contact') as
HTMLInputElement;
        let address: HTMLTextAreaElement =
document.getElementById('dialog')!.querySelector('#address') as
HTMLTextAreaElement;
        let template: string = "<div class='row'><div class='col-xs-6 col-sm-6
col-lg-6 col-md-6'><b>Confirm your details</b></div>" +
        "</div><div class='row'><div class='col-xs-6 col-sm-6 col-lg-6 col-md-
6'><span id='name'> Name: </span>" +
        "</div><div class='col-xs-6 col-sm-6 col-lg-6 col-md-6'><span
id='nameValue'>"+ input.value + "</span> </div></div>" +
        "<div class='row'><div class='col-xs-6 col-sm-6 col-lg-6 col-md-
6'><span id='email'> Email: </span>" +
        "</div><div class='col-xs-6 col-sm-6 col-lg-6 col-md-6'><span
id='emailValue'>"+ email.value +
        "</span></div></div><div class='row'><div class='col-xs-6 col-sm-6
col-lg-6 col-md-6'>"+
        "<span id='Contact'> Contact number: </span></div><div class='col-xs-6
col-sm-6 col-lg-6 col-md-6'>"+
        "<span id='contactValue'>"+ contact.value + " </span></div></div><div
class='row'><div class='col-xs-6 col-sm-6 col-lg-6 col-md-6'>"+
        "<span id='Address'> Address: </span> </div><div class='col-xs-6 col-
sm-6 col-lg-6 col-md-6'><span id='AddressValue'>"+ address.value
+ "</span></div></div>"
        return template;
    }
    public nestedbuttonClick: EmitType<object> = () => {
        this.ejInnerDialog!.hide();
        this.ejDialog!.show();
    }
    public footerbuttonclick: EmitType<object> = () => {

```

```

        this.ejInnerDialog!.hide();
    }
    public onOpenInnerDialog: EmitType<object> = () => {
        this.ejDialog!.hide();
        this.ejInnerDialog!.content = this.getDynamicContent();
        this.ejInnerDialog!.buttons = [{click:
this.footerbuttonclick.bind(this), buttonModel: { content: 'Yes', isPrimary:
true }},
        {click: this.nestedbuttonClick.bind(this), buttonModel: { content:
'No', isPrimary: true } }];
        this.ejInnerDialog!.show();
    }
    // Enables the footer buttons
    public buttons: Object = [
        {
            'click': this.onOpenInnerDialog.bind(this),
            // Accessing button component properties by buttonModel property
            buttonModel:{
                content:'Submit',
                //Enables the primary button
                isPrimary: 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 dialog appearance in Angular Dialog component

You can customize the dialog appearance by providing dialog template as string or HTML element to the [content](#) property. In the following sample, dialog is customized as error window appearance.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DialogModule } from '@syncfusion/ej2-angular-popups'
import { Component, ViewChild, OnInit, ElementRef } from '@angular/core';
import { DialogComponent } from '@syncfusion/ej2-angular-popups';
import { EmitType } from '@syncfusion/ej2-base';
@Component({
    imports: [
        DialogModule
    ],
    standalone: true,
    selector: 'app-root',
    template: `<button class="e-control e-btn" style="position: absolute;"
id="targetButton" (click)="onOpenDialog($event)">Open Dialog</button>
<div #container class='root-container'>

```

```

    <ejs-dialog id='dialog' #ejDialog
[animationSettings]='animationSettings' header='File and Folder Rename'
[target]='targetElement'
    width='400px' [showCloseIcon]='showCloseIcon' [buttons]='buttons'>
    <ng-template #content>
        <div class = 'dialog-content'>
            <div class='msg-wrapper col-lg-12'>
                <span class ='e-icons close-icon col-lg-2'></span>
                <span class ='error-msg col-lg-10'>Can not rename 'pictures'
because a file or folder with that name already exists </span>
            </div>
            <div class ='error-detail col-lg-8'>
                <span>Specify a different name</span>
            </div>
        </div>
    </ng-template>
    </ejs-dialog> </div> `
    })
    export class AppComponent implements OnInit {
        @ViewChild('ejDialog') ejDialog: DialogComponent | undefined;
        // Create element reference for dialog target element.
        @ViewChild('container', { read: ElementRef }) container: ElementRef |
undefined;
        // The Dialog shows within the target element.
        public targetElement?: HTMLElement;
        //To get all element of the dialog component after component get
initialized.
        ngOnInit() {
            this.initilaizeTarget();
        }
        // Initialize the Dialog component's target element.
        initilaizeTarget: EmitType<object> = () => {
            this.targetElement = this.container!.nativeElement.parentElement;
        }
        //Animation options
        public animationSettings: Object = { effect: 'Zoom', duration: 400,
delay: 0 };
        public showCloseIcon: boolean = true;
        // Hide the Dialog when click the footer button.
        public hideDialog: EmitType<object> = () => {
            this.ejDialog!.hide();
        }
        // Enables the footer buttons
        public buttons: Object = [
            {
                'click': this.hideDialog.bind(this),
                // Accessing button component properties by buttonModel property
                buttonModel:{
                    content:'Close',
                    //Enables the primary button
                    isPrimary: true
                }
            }
        ];
        // Sample level code to handle the button click action
        public onOpenDialog = (event: any): void => {
            // Call the show method to open the Dialog

```

```

        this.ejDialog!.show();
    }
}

```

MAIN.TS

```

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

```

Close dialog while click on outside of dialog in Angular Dialog component

By default, dialog can be closed by pressing Esc key and clicking the close icon on the right of dialog header. It can also be closed by clicking outside of the dialog using hide method.

Set the [closeOnEscape](#) property value to false to prevent closing of the dialog when pressing Esc key.

In the following sample, dialog is closed when clicking outside the dialog area using [hide](#) method.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DialogModule } from '@syncfusion/ej2-angular-popups'
import { Component, ViewChild, OnInit, ElementRef } from '@angular/core';
import { DialogComponent } from '@syncfusion/ej2-angular-popups';
import { EmitType } from '@syncfusion/ej2-base';
@Component({
  imports: [
    DialogModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<button class="e-control e-btn" style="position: absolute;"
id="targetButton" (click)="onOpenDialog($event)">Open Dialog</button>
<div #container class='root-container'>
  <ejs-dialog id='dialog' #ejDialog
[animationSettings]='animationSettings' content='Are you sure you want to
permanently delete all of these items?' header='Delete Multiple Items'
[target]='targetElement'
width='300px' [showCloseIcon]='showCloseIcon' [buttons]='buttons'>
</ejs-dialog> </div> `
})
export class AppComponent implements OnInit {
  @ViewChild('ejDialog') ejDialog: DialogComponent | undefined;
  // Create element reference for dialog target element.
  @ViewChild('container', { read: ElementRef }) container: ElementRef |
undefined;
  // The Dialog shows within the target element.
  public targetElement?: HTMLElement;
  //To get all element of the dialog component after component get
initialized.
  ngOnInit() {
    this.initilaizeTarget();
  }
}

```

```

ngAfterViewInit(): void {
    document.onclick = (args: any) : void => {
        if(args.target.tagName === 'BODY') {
            this.ejDialog!.hide();
        }
    }
}

// Initialize the Dialog component's target element.
initilaizeTarget: EmitType<object> = () => {
    this.targetElement = document.body;
}

//Animation options
public animationSettings: Object = { effect: 'Zoom', duration: 400,
delay: 0 };
public showCloseIcon: boolean = true;
// Hide the Dialog when click the footer button.
public hideDialog: EmitType<object> = () => {
    this.ejDialog!.hide();
}

// Enables the footer buttons
public buttons: Object = [
    {
        'click': this.hideDialog.bind(this),
        // Accessing button component properties by buttonModel property
        buttonModel:{
            content:'Yes',
            //Enables the primary button
            isPrimary: true
        }
    },
    {
        'click': this.hideDialog.bind(this),
        // Accessing button component properties by buttonModel property
        buttonModel:{
            content:'No',
        }
    }
];

// Sample level code to handle the button click action
public onOpenDialog = (event: any): void => {
    // Call the show method to open the Dialog
    this.ejDialog!.show();
}
}

```

MAIN.TS

```

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

```

In the following sample, the dialog is rendered based on the target container. It can also be closed by clicking outside of the dialog.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DialogModule } from '@syncfusion/ej2-angular-popups'
import { Component, ViewEncapsulation, ViewChild, ElementRef, AfterViewInit } from '@angular/core';
import { DialogComponent, PositionDataModel } from '@syncfusion/ej2-angular-popups';
import { EmitType } from '@syncfusion/ej2-base';
/**
 * Modal Dialog Component
 */
@Component({
  imports: [

    DialogModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<button class="e-control e-btn" style="position: absolute;"
id="targetButton" (click)="onOpenDialog($event)">Open Dialog</button>
<div id="modalTarget" #container class="root-container">
  <ejs-dialog id="dialog" #ejDialog
[animationSettings]='animationSettings' [position]='position' content='Are
you sure you want to permanently delete all of these items?' header='Delete
Multiple Items' [target]='targetElement'
width='300px' [showCloseIcon]='showCloseIcon' [buttons]='buttons'>
  </ejs-dialog> </div> `,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild('ejDialog') ejDialog: DialogComponent | undefined;
  // Create element reference for dialog target element.
  @ViewChild('container', { read: ElementRef }) container: ElementRef |
undefined;
  // The Dialog shows within the target element.
  public targetElement?: HTMLElement;
  public position: PositionDataModel = { X: 'center', Y: 'center' };
  //To get all element of the dialog component after component get
initialized.
  ngOnInit() {
    this.initilaizeTarget();
  }
  ngAfterViewInit(): void {
    document.onclick = (args: any) : void => {
      let currentTarget = args.target.closest('ejs-dialog, button');
      if(currentTarget === null) {
        this.ejDialog!.hide();
      }
    }
  }
  // Initialize the Dialog component's target element.
  initilaizeTarget: EmitType<object> = () => {
    this.targetElement = document.getElementById("modalTarget") as any;
  }
  //Animation options

```



```

    public animationSettings: Object = { effect: 'Zoom', duration: 400,
delay: 0 };
    public showCloseIcon: boolean = true;
    // Hide the Dialog when click the footer button.
    public hideDialog: EmitType<object> = () => {
        this.ejDialog!.hide();
    }
    // Enables the footer buttons
    public buttons: Object = [
        {
            'click': this.hideDialog.bind(this),
            // Accessing button component properties by buttonModel property
            buttonModel:{
                content:'Yes',
                //Enables the primary button
                isPrimary: true
            }
        },
        {
            'click': this.hideDialog.bind(this),
            // Accessing button component properties by buttonModel property
            buttonModel:{
                content:'No',
            }
        }
    ];
    // Sample level code to handle the button click action
    public onOpenDialog = (event: any): void => {
        // Call the show method to open the Dialog
        (this as any).ejDialog.show();
    }
}

```

MAIN.TS

```

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

```

Add an icons to dialog buttons in Angular Dialog component

You can add icons to the dialog buttons using the [buttons](#) property or [footerTemplate](#) property . For detailed information about dialog buttons, refer to the [documentation](#) section.

In the following sample, dialog footer buttons are customized with icons using **buttons** property.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DialogModule } from '@syncfusion/ej2-angular-popups'
import { Component, ViewChild, OnInit, ElementRef } from '@angular/core';
import { DialogComponent } from '@syncfusion/ej2-angular-popups';
import { EmitType } from '@syncfusion/ej2-base';
@Component({
  imports: [

```

```

        DialogModule
    ],
    standalone: true,
    selector: 'app-root',
    template: `<button class="e-control e-btn" style="position: absolute;"
id="targetButton" (click)="onOpenDialog($event)">Open Dialog</button>
    <div #container class='root-container'>
        <ejs-dialog id='dialog' #ejDialog
[animationSettings]='animationSettings' content='Are you sure you want to
permanently delete all of these items?' header='Delete Multiple Items'
[target]='targetElement'
        width='300px' [showCloseIcon]='showCloseIcon' [buttons]='buttons'>
    </ejs-dialog> </div> `
    })
export class AppComponent implements OnInit {
    @ViewChild('ejDialog') ejDialog: DialogComponent | undefined;
    // Create element reference for dialog target element.
    @ViewChild('container', { read: ElementRef }) container: ElementRef |
undefined;
    // The Dialog shows within the target element.
    public targetElement?: HTMLElement;
    //To get all element of the dialog component after component get
initialized.
    ngOnInit() {
        this.initilaizeTarget();
    }
    // Initialize the Dialog component's target element.
    initilaizeTarget: EmitType<object> = () => {
        this.targetElement = this.container!.nativeElement.parentElement;
    }
    //Animation options
    public animationSettings: Object = { effect: 'Zoom', duration: 400,
delay: 0 };
    public showCloseIcon: boolean = true;
    // Hide the Dialog when click the footer button.
    public hideDialog: EmitType<object> = () => {
        this.ejDialog!.hide();
    }
    // Enables the footer buttons
    public buttons: Object = [
        {
            'click': this.hideDialog.bind(this),
            // Accessing button component properties by buttonModel property
            buttonModel:{
                content:'Yes',
                iconCss: 'e-icons e-ok-icon',
                //Enables the primary button
                isPrimary: true
            }
        },
        {
            'click': this.hideDialog.bind(this),
            // Accessing button component properties by buttonModel property
            buttonModel:{
                content:'No',
                iconCss: 'e-icons e-close-icon'
            }
        }
    ]
}

```

```

    }
  }
};
// Sample level code to handle the button click action
public onOpenDialog = (event: any): void => {
  // Call the show method to open the Dialog
  this.ejDialog!.show();
}
}

```

MAIN.TS

```

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

```

In the following sample, dialog footer buttons are customized with icons using `footerTemplate` property.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DialogModule } from '@syncfusion/ej2-angular-popups'
import { Component, ViewChild, OnInit, ElementRef } from '@angular/core';
import { DialogComponent } from '@syncfusion/ej2-angular-popups';
import { EmitType } from '@syncfusion/ej2-base';
@Component({
  imports: [

    DialogModule

  ],
  standalone: true,
  selector: 'app-root',
  template: `<button class="e-control e-btn" style="position: absolute;"
id="targetButton" (click)="onOpenDialog($event)">Open Dialog</button>
<div #container class='root-container'></div>
<ejs-dialog id='dialog' #ejDialog
[animationSettings]='animationSettings' (close)='dialogClose()' content='Are
you sure you want to permanently delete all of these items?' header='Delete
Multiple Items' [target]='targetElement'
width='300px' [showCloseIcon]='showCloseIcon'>
<ng-template #footerTemplate>
<div>
<button id="Button1" class="e-control e-btn e-primary e-flat"
(click)="btnClick()" data-ripple="true">
<span class="e-btn-icon e-icons e-ok-icon e-icon-
left"></span>Yes</button>
<button id="Button2" class="e-control e-btn e-flat"
(click)="btnClick()" data-ripple="true">
<span class="e-btn-icon e-icons e-close-icon e-icon-
left"></span>No</button>
</div>
</ng-template>
</ejs-dialog>
`
})

```

```

    })
    export class AppComponent implements OnInit {
    dialogClose() {
    throw new Error('Method not implemented.');
```

```

    }

    @ViewChild('ejDialog') ejDialog: DialogComponent | undefined;
    // Create element reference for dialog target element.
    @ViewChild('container', { read: ElementRef }) container: ElementRef |
    undefined;
    // The Dialog shows within the target element.
    public targetElement?: HTMLElement;
    //To get all element of the dialog component after component get
    initialized.
    ngOnInit() {
        this.initilaizeTarget();
    }
    // Initialize the Dialog component's target element.
    initilaizeTarget: EmitType<object> = () => {
        this.targetElement = this.container!.nativeElement.parentElement;
    }
    //Animation options
    public animationSettings: Object = { effect: 'Zoom', duration: 400,
    delay: 0 };
    public showCloseIcon: boolean = true;
    // Hide the Dialog when click the footer button.
    public hideDialog: EmitType<object> = () => {
        this.ejDialog!.hide();
    }
    public btnclick = (): void => {
        this.ejDialog!.hide();
    }
    // Sample level code to handle the button click action
    public onOpenDialog = (event: any): void => {
        // Call the show method to open the Dialog
        this.ejDialog!.show();
    }
    }
}
```

MAIN.TS

```

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

Use dialog in ng routing in Angular Dialog component

You can use dialog inside the angular routing application. For more details, refer to the [Angular Documentation](#).

The following example demonstrates how to update the dialog content using angular service in routing application.

[Dialog with ng-routing sample](#)

Render a dialog using ng content in Angular Dialog component

You can render your own custom component to the dialog content. The below example shows that rendering child element in your custom component, by using ng-content.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DialogModule } from '@syncfusion/ej2-angular-popups'
import { Component, ViewChild } from '@angular/core';
import { DialogComponent } from '@syncfusion/ej2-angular-popups';
@Component({
  imports: [

    DialogModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<div class="control-section">
    <button class="e-btn" id='dlgbtn' #ButtonInstance
    (click)="BtnClick()">Open</button>
    <!-- Render Button to open the Dialog -->
    <ejs-dialog #Dialog [buttons]='dlgButtons' [header]='header'
    [animationSettings]='animationSettings' [showCloseIcon]='showCloseIcon'
    [width]='width' [height]='height' (open)="dialogOpen()"
    (close)="dialogClose()">
      <app-ng-content>
      <span> 10% of battery remaining </span>
    </app-ng-content>
    </ejs-dialog>
  </div>`
})
export class AppComponent {
  @ViewChild('Dialog')
  public Dialog?: DialogComponent;
  public showCloseIcon: Boolean = true;
  public width: string = '250px';
  public height: string = '150px';
  public animationSettings: Object = { effect: 'None' };
  public header: string = 'Low Battery';
  BtnClick() {
    this.Dialog!.show();
  }
  dialogClose() {
    document.getElementById('dlgbtn')!.style.display = 'block';
  }
  // On Dialog open, 'Open' Button will be hidden
  dialogOpen() {
    document.getElementById('dlgbtn')!.style.display = 'none';
  }
  public dlgButtons: Object[] = [{
    click: this.dlgBtnClick.bind(this),
    buttonModel: { content: 'OK', isPrimary: 'true' } },
    { click: this.dlgBtnClick.bind(this), buttonModel: { content:
'Cancel' }
  }];
  dlgBtnClick() {
```

```

        this.Dialog!.hide();
    }
}
@Component({
  selector: 'app-ng-content',
  template: `<div class="dialog-content-wrapper">
    <ng-content></ng-content>
  </div>`
})
export class NgContentComponent {
}

```

MAIN.TS

```

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

```

Use template driven forms inside dialog in Angular Dialog component

The following sample demonstrates how to use the template-driven forms with required validation inside the dialog. For more details, refer to the [Angular Documentation](#)

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule, ReactiveFormsModule } from '@angular/forms'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { DialogModule } from '@syncfusion/ej2-angular-popups'
import { UploaderModule } from '@syncfusion/ej2-angular-inputs'
import { Component, OnInit, ViewChild, ViewEncapsulation } from '@angular/core';
import { UploaderComponent } from '@syncfusion/ej2-angular-inputs';
import { DialogComponent } from '@syncfusion/ej2-angular-popups';
import { isNullOrUndefined, EmitType } from '@syncfusion/ej2-base';
@Component({
  imports: [ FormsModule, DialogModule, UploaderModule, ButtonModule,
    ReactiveFormsModule ],
  standalone: true,
  selector: 'app-root',
  template: `<div class="control-section">
    <div class="col-lg-12">
      <div class="control_wrapper" id="control_wrapper" style="margin:
10px auto;">
        <ejs-dialog id="FormDialog" #FormDialog width= '450px'
          [showCloseIcon]='false' target='.control-section'>
          <ng-template #content>
            <form id="template_driven" #userForm="ngForm" novalidate>
              <div class="form-group" style="padding-top: 11px;">
                <div class="e-float-input">
                  <input type="text" id="name" #nameval='ngModel'
name="name" required ngModel>
                  <span class="e-float-line"></span>

```

```

        <label class="e-float-text e-label-top"
for="name">Name</label>
        <div *ngIf="(nameval.invalid && (nameval.dirty ||
nameval.touched))">
            <div class="e-error"
*ngIf="nameval.errors!['required']">
                * Enter your name
            </div>
        </div>
    </div>
    <div class="form-group" style="padding-top: 11px;">
        <div class="e-float-input">
            <input type="text" id="email" #emailval='ngModel'
name="email" required ngModel>
            <span class="e-float-line"></span>
            <label class="e-float-text e-label-top"
for="email">Email</label>
            <div *ngIf="(emailval.invalid && (emailval.dirty ||
emailval.touched))">
                <div class="e-error"
*ngIf="emailval.errors!['required']">
                    * Enter your email address
                </div>
            </div>
        </div>
        <div class="form-group" style="padding-top: 11px;">
            <div class="e-float-input">
                <input type="number" id="contact"
#contactval='ngModel' name="contact" required ngModel>
                <span class="e-float-line"></span>
                <label class="e-float-text e-label-top"
for="contact">Contact No</label>
                <div *ngIf="(contactval.invalid && (contactval.dirty
|| contactval.touched))">
                    <div class="e-error"
*ngIf="contactval.errors!['required']">
                        * Enter your contact number
                    </div>
                </div>
            </div>
        </div>
        <div class="selected-files">
            <div class="textboxes">
                <div class="form-group" style="padding-top: 11px;
width:">
                    <div class="e-float-input upload-area">
                        <input type="text" id="upload"
#uploadval='ngModel' [(ngModel)]="uploadInput" readonly name="upload"
required ngModel>
                        <span class="e-float-line"></span>
                        <label class="e-float-text e-label-top"
for="upload">Choose a file</label>
                    </div>
                </div>
            </div>
        </div>
    </div>

```

```

        <div class="uploader-section">
            <button id="browse" class="e-control e-btn e-info"
(click)='browseClick()'>Browse...</button>
            <ejs-uploader #defaultupload id='fileupload'
allowedExtensions="image/*" [autoUpload]=false [multiple]='multiple'
(selected)='onFileSelect($event)'></ejs-uploader>
        </div>
    </div>
    <span class='error-root' *ngIf="(uploadval.invalid &&
(uploadval.dirty || uploadval.touched))">
        <span class="e-error errorClass"
*ngIf="uploadval.errors!['required']">
            * Select a file
        </span>
    </span>
    <div class="form-group" style="padding-top: 11px;">
        <div class="submitBtn">
            <button class="submit-btn e-btn" id="submit-btn"
[disabled]="userForm.invalid" type="reset" (click)=
"Submit()">Submit</button>
            <div class="desc"><span>*This button is not a submit
type and the form submit handled from externally.</span></div>
        </div>
    </div>
</form>
</ng-template>
</ejs-dialog>
</div>
</div>
</div>
    <ejs-dialog id="confirmationDialog" #Dialog [buttons]='dlgButtons'
[animationSettings]='animationSettings' [header]='formHeader'
[showCloseIcon]='showCloseIcon' [content]='contentData' [target]='target'
[width]='width' [visible]="visible" [isModal]="isModal" >
        </ejs-dialog>`
    })
    export class AppComponent {
        @ViewChild('formUpload')
        public uploadObj?: UploaderComponent;
        @ViewChild('Dialog')
        public dialogObj?: DialogComponent;
        public width: string = '335px';
        public visible: boolean = false;
        public multiple: boolean = false;
        public showCloseIcon: Boolean = true;
        public formHeader: string = 'Success';
        public contentData: string = 'Your details have been updated successfully,
Thank you.';
        public target: string = '#FormDialog';
        public isModal: boolean = true;
        public animationSettings: object = {
            effect: 'Zoom'
        };
        public uploadInput: string = '';
        public dlgBtnClick: EmitType<object> = () => {
            this.dialogObj!.hide();
        }
    }

```



```

    public dlgButtons: Object[] = [{ click: this.dlgBtnClick.bind(this),
    buttonModel: { content: 'Ok', isPrimary: true } }];
    @ViewChild('formElement') element: any;
    public browseClick() {
        document.getElementsByClassName('e-file-select-
wrap')[0].querySelector('button')!.click(); return false;
    }
    public Submit(): void {
        this.onFormSubmit();
    }
    public onFileSelect: EmitType<Object> = (args: any) => {
        this.uploadInput = args.filesData[0].name;
    }
    public onFormSubmit(): void {
        this.dialogObj!.show();
    }
}

```

MAIN.TS

```

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

```

Use reactive forms inside dialog in Angular Dialog component

The following sample demonstrates how to use the reactive forms with required validation inside the dialog.

For more details, refer to the [Angular Documentation](#)

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule, ReactiveFormsModule } from '@angular/forms'
import { UploaderModule } from '@syncfusion/ej2-angular-inputs'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { DialogModule } from '@syncfusion/ej2-angular-popups'
import { Component, Inject, OnInit, ViewChild, ViewEncapsulation } from
'@angular/core';
import { FormGroup, FormBuilder, Validators, FormControl } from
'@angular/forms';
import { EmitType } from '@syncfusion/ej2-base';
import { UploaderComponent } from '@syncfusion/ej2-angular-inputs';
import { DialogComponent } from '@syncfusion/ej2-angular-popups';
@Component({
    imports: [ FormsModule, DialogModule, UploaderModule, ButtonModule,
    ReactiveFormsModule ],
    standalone: true,
    selector: 'app-root',
    template: `<div class="control-section">
        <div class="col-lg-12">
            <div class="control_wrapper" id="control_wrapper" style="margin:
25px auto;">
                <ejs-dialog id="reactDialog" #FormDialog width= '450px'

```

```

[showCloseIcon]='false' target='.control-section'>
<form id="reactive" [formGroup]="form">
  <div class="form-group" style="padding-top: 11px;">
    <div class="e-float-input">
      <input type="text" id="name" name="name"
class="required" formControlName="name">
      <span class="e-float-line"></span>
      <label class="e-float-text e-label-top"
for="name">Name</label>
    </div>
    <app-field-error-display
[displayError]="isFieldValid('name')" errorMsg="* Please Enter your name">
    </app-field-error-display>
  </div>
  <div class="form-group" style="padding-top: 11px;">
    <div class="e-float-input">
      <input type="text" id="email" name="email"
class="required" formControlName="email">
      <span class="e-float-line"></span>
      <label class="e-float-text e-label-top"
for="email">Email</label>
    </div>
    <app-field-error-display
[displayError]="isFieldValid('email')" errorMsg="* * Enter your email
address">
    </app-field-error-display>
  </div>
  <div class="form-group" style="padding-top: 11px;">
    <div class="e-float-input">
      <input type="number" id="contact" name="contact"
class="required" formControlName="contact">
      <span class="e-float-line"></span>
      <label class="e-float-text e-label-top"
for="contact">Contact No</label>
    </div>
    <app-field-error-display
[displayError]="isFieldValid('contact')" errorMsg="* Enter your contact
number">
    </app-field-error-display>
  </div>
  <div class="form-group upload-area" style="padding-top:
11px;">
    <div class="e-float-input upload-area">
      <input type="text" id="upload" name="upload"
[(ngModel)]="uploadInput" readonly formControlName="upload"
class="required">
      <span class="e-float-line"></span>
      <label class="e-float-text e-label-top"
for="upload">Choose a file</label>
    </div>
    <button id="browse" class="e-control e-btn e-info"
(click)='browseClick()'>Browse...</button>
    <ejs-uploader #defaultupload id='fileupload'
allowedExtensions="image/*" [autoUpload]=false [multiple]='multiple'
(selected)='onFileSelect($event)'></ejs-uploader>
    <app-field-error-display
[displayError]="isFieldValid('upload')" errorMsg="* Select any file">

```

```

        </app-field-error-display>
      </div>
      <div class="form-group" style="padding-top: 11px;">
        <div class="submitBtn">
          <button class="submit-btn e-btn" id="submit-btn"
[disabled]="form!.invalid" (click)= "Submit()">Submit</button>
          <div class="desc"><span>*This button is not a submit type
and the form submit handled from externally.</span></div>
        </div>
      </div>
    </form>
  </ejs-dialog>
</div>
</div>
</div>
<ejs-dialog id="confirmationDialog" #Dialog [buttons]='dlgButtons'
[animationSettings]='animationSettings' [header]='formHeader'
[showCloseIcon]='showCloseIcon' [content]='contentData' [target]='target'
[width]='width' [visible]="visible" [isModal]="isModal" >
  </ejs-dialog>`
  })
  export class AppComponent {
    @ViewChild('defaultupload')
    public uploadObj?: UploaderComponent;
    @ViewChild('Dialog')
    public dialogObj?: DialogComponent;
    public form?: FormGroup | any;
    public width: string = '335px';
    public visible: boolean = false;
    public multiple: boolean = false;
    public showCloseIcon: Boolean = true;
    public formHeader: string = 'Success';
    public contentData: string = 'Your details have been updated
successfully, Thank you.';
    public target: string = '#reactDialog';
    public isModal: boolean = true;
    public animationSettings: any = {
      effect: 'Zoom'
    };
    private formSumitAttempt: boolean | any;
    public dlgBtnClick: EmitType<object> = () => {
      this.dialogObj!.hide();
    }
    public dlgButtons: Object[] = [{ click: this.dlgBtnClick.bind(this),
buttonModel: { content: 'Ok', isPrimary: true } }];
    public uploadInput: string = '';
    public browseClick() {
      document.getElementsByClassName('e-file-select-
wrap')[0].querySelector('button')!.click(); return false;
    }
    public Submit(): void {
      this.onFormSubmit();
    }
    public onFileSelect: EmitType<Object> = (args: any) => {
      this.uploadInput = args.filesData[0].name;
    }
    public onFormSubmit(): void {

```

```

        this.formSumitAttempt = true;
        if (this.form!.valid) {
            this.dialogObj!.show();
            this.form!.reset();
        } else {
            this.validateAllFormFields(this.form!);
        }
    }
    constructor(@Inject(FormBuilder) public formBuilder: FormBuilder) {}
    ngOnInit() {
        this.form = this.formBuilder.group({
            name: [null, Validators.required],
            email: [null, Validators.required],
            contact: [null, Validators.required],
            upload: [null, Validators.required],
        });
    }
    isFieldValid(field: string) {
        return (((this.form as any).get(field).valid && (this.form as any).get(field).touched) ||
            ((this.form as any).get(field).untouched && this.formSumitAttempt));
    }
    validateAllFormFields(formGroup: FormGroup) {
        Object.keys(formGroup.controls).forEach(field => {
            const control = formGroup.get(field);
            if (control instanceof FormControl) {
                control.markAsTouched({ onlySelf: true });
            } else if (control instanceof FormGroup) {
                this.validateAllFormFields(control);
            }
        });
    }
}

```

MAIN.TS

```

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

```

Bind the event to the buttons in footer in Angular Dialog component

You can bind the actions to the buttons inside the footer using the buttons property. In the following example, dialog will be closed when you click on the buttons.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DialogModule } from '@syncfusion/ej2-angular-popups'
import { Component, ViewChild, OnInit, ElementRef } from '@angular/core';
import { DialogComponent } from '@syncfusion/ej2-angular-popups';
import { EmitType } from '@syncfusion/ej2-base';
@Component({
    imports: [

```

```

        DialogModule
    ],
    standalone: true,
    selector: 'app-root',
    template: `<div class="control-section">
        <button class="e-btn" id="dlgbtn" #ButtonInstance
        (click)="BtnClick()">Open</button>
        <!-- Render Button to open the Dialog -->
        <ejs-dialog #Dialog [buttons]='dlgButtons' [header]='header'
        [animationSettings]='animationSettings' [content]='content'
        [showCloseIcon]='showCloseIcon' [target]='target' [width]='width'
        (open)="dialogOpen()"
        (close)="dialogClose()">
            </ejs-dialog>
    </div>`
    })
    export class AppComponent implements OnInit {
        ngOnInit(): void {
            throw new Error('Method not implemented.');
        }
        @ViewChild('Dialog')
        public Dialog?: DialogComponent;
        BtnClick() {
            this.Dialog!.show();
        }
        public header: string = 'Delete Multiple Items';
        public content: string = 'Are you sure you want to permanently delete
these items ?'
        public showCloseIcon: Boolean = true;
        public width: string = '300px';
        public animationSettings: Object = { effect: 'None' };
        public hide?: any;
        ngAfterViewInit():void{
            document.getElementById('dlgbtn')!.focus();
        }
        // On Dialog close, 'Open' Button will be shown
        dialogClose() {
            document.getElementById('dlgbtn')!.style.display = '';
        }
        // On Dialog open, 'Open' Button will be hidden
        dialogOpen() {
            document.getElementById('dlgbtn')!.style.display = 'none';
        }
        public dlgButtons: Object[] = [{ click: this.dlgBtnClick.bind(this),
        buttonModel: { content: 'Yes', isPrimary:'true' } }, { click:
        this.dlgBtnClick.bind(this), buttonModel: { content: 'No' } }];
        this.dlgBtnClick.bind(this), buttonModel: { content: 'No' } }];
        dlgBtnClick() {
            this.Dialog!.hide();
        }
        public target: string = '.control-section';
    }

```

MAIN.TS

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

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

To access the current component within the button actions, add **bind(this)** along with function name in action binding.

Render a dialog using ng template in Angular Dialog component

You can provide the HTML elements as header, footer, and content of the dialog using ng-template directives. For more details, refer to the [Angular Documentation](#).

In this [example](#), dialog header, footer, and content are rendered using ng-template directives

How to destroy the dialog component when navigate pages using Angular routing

By default, the dialog component appends into the body element when you did not specify any target to the dialog. When navigate pages using Angular routing, the elements inside the routing pages gets destroyed. So, the dialog elements are not destroyed properly while routing. It will cause the memory leak problem in DOM when continuously navigating the pages.

You can avoid this problem using one of the following solutions:

Solution 1

Set the target to the dialog component to solve the destroy related issue in the DOM. If you set the target property to dialog component, the dialog appends inside the target element that is placed on the routing partial page. It destroys both dialog component and it's all elements when switching between the routing component page.

Refer to this [sample](#).

Solution 2

If you do not want to set the target property to dialog, use this solution. Destroy the dialog elements using the ngOnDestroy method from your application. The ngOnDestroy method is called before the component/directive destroy by Angular.

Refer to this [sample](#).

Add a minimize maximize buttons in Angular Dialog component

Angular Dialog allows end users to either minimize or maximize the Dialog component. You can add minimize and maximize custom buttons near the close icon in the Dialog header using the [headerTemplate](#) property and handle the actions in the button click events, as shown in the following sample.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DialogModule } from '@syncfusion/ej2-angular-popups'
import { Component, ViewChild, OnInit, ElementRef } from '@angular/core';
import { DialogComponent } from '@syncfusion/ej2-angular-popups';
import { EmitType } from '@syncfusion/ej2-base';

@Component({
  imports: [
    DialogModule
  ],
```

```

standalone: true,
  selector: 'app-root',
  template: `
    <button class="e-control e-btn" style="position: absolute;"
id="targetButton" (click)="onOpenDialog($event)">Open Dialog</button>
    <div #container class='root-container'></div>
    <ejs-dialog id='dialog' #Dialog showCloseIcon='true'
[visible]='visible' content='This is a dialog with minimize and maximize
buttons' [target]='targetElement'
width='250px' [buttons]='buttons'>
    <ng-template #header>
      <span class='title'>Dialog</span>
      <span class='e-icons sf-icon-Maximize' id='max-btn'
title='Maximize' (click)='maximize()'></span>
      <span class='e-icons sf-icon-Minimize' id='min-btn'
title='Minimize' (click)='minimize()'></span>
    </ng-template>
    </ejs-dialog> `
  })
export class AppComponent implements OnInit {
  @ViewChild('Dialog') dialogObj: DialogComponent | undefined;
  // Create element reference for dialog target element.
  @ViewChild('container', { read: ElementRef }) container: ElementRef |
undefined;
  // The Dialog shows within the target element.
  public targetElement?: HTMLElement;
  public width: string = '350px';
  public hide?: any;
  public isFullScreen?: Boolean;
  public dialogOldPositions?: any;
  //To get all the elements of the dialog component after the component is
initialized.
  ngOnInit() {
    this.initilaizeTarget();
  }
  // Initialize the Dialog component's target element.
  public initilaizeTarget: EmitType<object> = () => {
    this.targetElement = this.container!.nativeElement.parentElement;
  }
  public visible: Boolean = false;
  // Hide the Dialog when click the footer button.
  public hideDialog: EmitType<object> = () => {
    this.dialogObj!.hide();
  }
  // Enables the footer buttons
  public buttons: Object = [
    {
      'click': this.hideDialog.bind(this),
      // Accessing button component properties by using the
buttonModel property
      buttonModel: {
        content: 'OK',
        //Enables the primary button
        isPrimary: true
      }
    },
    {

```

```

        'click': this.hideDialog.bind(this),
        buttonModel: {
            content: 'Cancel'
        }
    };
    public onOpenDialog = (event: any): void => {
        // Call the show method to open the Dialog
        this.dialogObj!.show();
    }
    public maximize(): void {
        var maximizeIcon;
        if (this.dialogObj!.element.classList.contains('dialog-minimized')) {
            this.dialogObj!.element.querySelector('#min-btn')!.classList.add('sf-
            icon-Minimize');
            this.dialogObj!.element.querySelector('#min-
            btn')!.classList.remove('sf-icon-Restore');
            this.dialogObj!.element.querySelector('#min-
            btn')!.setAttribute('title', 'Minimize');
        }
        if (!this.dialogObj!.element.classList.contains('dialog-maximized') &&
        !this.isFullScreen) {
            maximizeIcon = this.dialogObj!.element.querySelector(".e-dlg-header-
            content .sf-icon-Maximize");
        } else {
            maximizeIcon = this.dialogObj!.element.querySelector(".e-dlg-header-
            content .sf-icon-Restore");
        }
        if (!this.dialogObj!.element.classList.contains('dialog-maximized')) {
            this.dialogObj!.element.classList.add('dialog-maximized');
            this.dialogObj!.show(true);
            maximizeIcon!.classList.add('sf-icon-Restore');
            maximizeIcon!.setAttribute('title', 'Restore');
            maximizeIcon!.classList.remove('sf-icon-Maximize');
            this.dialogObj!.element.querySelector('.e-dlg-
            content')!.classList.remove('hide-content');
            this.isFullScreen = true;
        } else {
            this.dialogObj!.element.classList.remove('dialog-maximized');
            this.dialogObj!.show(false);
            maximizeIcon!.classList.remove('sf-icon-Restore');
            maximizeIcon!.classList.add('sf-icon-Maximize');
            maximizeIcon!.setAttribute('title', 'Maximize');
            this.dialogObj!.element.querySelector('.e-dlg-
            content')!.classList.remove('hide-content');
            this.dialogObj!.position = this.dialogOldPositions;
            this.dialogObj!.dataBind();
            this.isFullScreen = false;
        }
    }
    public minimize(): void {
        var minimizeIcon = this.dialogObj!.element.querySelector(".e-dlg-header-
        content .sf-icon-Minimize");
        if (!this.dialogObj!.element.classList.contains('e-dlg-fullscreen')) {
            if (!this.dialogObj!.element.classList.contains('dialog-minimized')) {
                this.dialogOldPositions = { X: this.dialogObj!.position.X, Y:
                this.dialogObj!.position.Y }
            }
        }
    }

```



```

        this.dialogObj!.element.classList.add('dialog-minimized');
        this.dialogObj!.element.classList.remove('dialog-maximized');
        this.dialogObj!.element.querySelector('.e-dlg-
content')!.classList.add('hide-content');
        this.dialogObj!.position = { X: 'center', Y: 'bottom' };
        this.dialogObj!.dataBind();
        minimizeIcon!.classList.add('sf-icon-Restore');
        minimizeIcon!.setAttribute('title', 'Restore');
    } else {
        this.dialogObj!.element.classList.remove('dialog-minimized');
        this.dialogObj!.element.querySelector('.e-dlg-
content')!.classList.remove('hide-content');
        minimizeIcon!.classList.add('sf-icon-Minimize');
        minimizeIcon!.setAttribute('title', 'Minimize');
        minimizeIcon!.classList.remove('sf-icon-Restore');
        this.dialogObj!.position = this.dialogOldPositions;
        this.dialogObj!.dataBind();
    }
    } else {
        this.dialogObj!.show(false);
        this.dialogObj!.element.classList.remove('dialog-maximized');
        this.dialogObj!.element.classList.add('dialog-minimized');
        this.dialogObj!.element.querySelector('.e-dlg-
content')!.classList.add('hide-content');
        minimizeIcon!.classList.remove('sf-icon-Minimize');
        minimizeIcon!.removeAttribute('title');
        this.dialogObj!.position = { X: 'center', Y: 'bottom' };
        this.dialogObj!.dataBind();
        this.isFullScreen = 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 max height to the dialog in Angular Dialog component

By default, the maxHeight for the Dialog is calculated based on the target. If the target is not specified externally, the Dialog consider the body as target and will calculate the maxHeight based on it. We have an option to set the maxHeight of the Dialog in the [beforeOpen](#) event.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DialogModule } from '@syncfusion/ej2-angular-popups'
import { Component, ViewChild, OnInit, ElementRef } from '@angular/core';
import { DialogComponent } from '@syncfusion/ej2-angular-popups';
import { EmitType } from '@syncfusion/ej2-base';
@Component({
  imports: [

```

```

        DialogModule
    ],
    standalone: true,
    selector: 'app-root',
    template: `
<button class="e-control e-btn" style="position: absolute;"
id="targetButton" (click)="onOpenDialog($event)">Open Dialog</button>
<div #container class='root-container'></div>
<ejs-dialog id='dialog' #ejDialog header='Dialog' showCloseIcon='true'
[visible]='visible' [target]='targetElement'
width='800px' (created)="onCreated()"
(beforeOpen)="onBeforeOpen($event)" [position]='position'>
<ng-template #content>
    <div id="dlgContent" style="display: none">
        <div>
            <b>JavaScript:</b><br />
            JavaScript is a high-level, dynamic, untyped, and
interpreted programming language. It has been standardized in the ECMAScript
language specification. Alongside HTML and CSS, it is
one of the three essential technologies of World Wide Web
content production; the majority of websites employ it
and it is supported by all modern Web browsers without
plug-ins. JavaScript is a prototype-based programming
language with first-class functions, making it a multi-paradigm language,
supporting object-oriented , imperative, and functional
programming styles.
            <br /><br /><br />
            <b>MVC:</b><br />
            Model-view-controller (MVC) is a software architecture
pattern which separates the representation of information from the user's
interaction with it. The model consists of application data, business rules,
logic, and functions. A view can be any output representation of data, such
as a chart or a diagram. Multiple views of the same data are possible, such
as a bar chart for management and a tabular view for accountants. The
controller mediates input, converting it to commands for the model or
view. The central ideas behind MVC are code reusability and in addition to
dividing the application into three kinds of components, the MVC design
defines the interactions between them.
            A controller can send commands to its associated view to
change the view's presentation of the model (e.g., by scrolling through a
document). It can also send commands to the model to update the model's
state (e.g., editing a document).
            A model notifies its associated views and controllers
when there is a change in its state. This notification allows the views to
produce updated output, and the controllers to change the available set of
commands. A passive implementation of MVC omits these notifications because
the application does not require them or the software platform does not
support them.
            A view requests from the model the information that it
needs to generate an output representation to the user.
        </div>
    </div>
</ng-template>
</ejs-dialog> `
    })
    export class AppComponent implements OnInit {

```

```

    @ViewChild('ejDialog') ejDialog: DialogComponent | any;
    // Create element reference for dialog target element.
    @ViewChild('container', { read: ElementRef }) container: ElementRef |
any;
    // The Dialog shows within the target element.
    public targetElement?: HTMLElement;
    position = { X: 'center', Y: 'center' };
    //To get all element of the dialog component after component get
    initialized.
    ngOnInit() {
        this.initilaizeTarget();
    }
    // Initialize the Dialog component's target element.
    public initilaizeTarget: EmitType<object> = () => {
        this.targetElement = this.container.nativeElement.parentElement;
    }
    public visible: Boolean = false;
    // Hide the Dialog when click the footer button.
    public hideDialog: EmitType<object> = () => {
        this.ejDialog.hide();
    }
    // Sample level code to handle the button click action.
    public onOpenDialog = (event: any): void => {
        // Call the show method to open the Dialog.
        this.ejDialog.show();
    }
    public onCreated = (): void => {
        document.getElementById('dlgContent')!.style.display = 'block';
        this.ejDialog.refreshPosition();
    }
    public onBeforeOpen = function(args: any): void {
        // setting maxHeight to the Dialog.
        args.maxHeight = '300px';
    }
}

```

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 Dialog component

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

Accessibility and Localization

<!-- markdownlint-disable MD033 -->

Behavior	Property in Essential JS 1	Property in Essential JS 2
----------	----------------------------	----------------------------

--	--	--

| Keyboard Navigation | **Property** : allowKeyboardNavigation
 <ej-dialog id='dialog'[allowKeyboardNavigation]='true'></ej-dialog> | No separate Property for enable/disable keyboard navigation. Its enabled by default. |

| Localization | **Property** : locale
 <ej-dialog id='dialog'[locale]='es-ES'></ej-dialog> | **Property** : locale
 <ejs-dialog id='#dialog' locale='fr-BE'></ejs-dialog>
ngOnInit() { // Load French culture for Dialog close button tooltip text
 L10n.load({ 'fr-BE': { 'dialog': { 'close': 'Fermer' }}}); } |

| Right to left | **Property**: enableRTL
 <ej-dialog id='dialog' [enableRTL]='true'></ej-dialog> | **Property**: enableRTL
 <ejs-dialog id='dialog' [enableRTL]='true'></ejs-dialog> |

Header

<!-- markdownlint-disable MD033 -->

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

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

| Header Content | **Property** : title
 <ej-dialog id='dialog' [enableRTL]='true' title='EJ1 Dialog header'></ej-dialog>
 Method : setTitle
 \$(' #dialog').ejDialog('setTitle', 'EJ1 Dialog Header'); | **Property** : header
 <ejs-dialog id='dialog' header='EJ2 Dialog header'></ejs-dialog> |

| close button | **Property** : actionButtons
 <ej-dialog id='dialog' [actionButtons]='actionbuttons'></ej-dialog>
 TS : actionbuttons: any;
constructor() { this.actionbuttons = ['close']; } | **Property** : showCloseIcon
 <ejs-dialog id='dialog' [showCloseIcon]='true'></ejs-dialog> |

| Event triggers when click on action buttons | **Event**: actionButtonClick
 <ej-dialog id='dialog' [actionButtons]='actionbuttons' (actionButtonClick)='buttonAction(\$event)'></ej-dialog>
 TS:
 actionbuttons: any;
constructor() { this.actionbuttons = ['close']; } buttonAction(event){ } | Not Applicable |

| Minimize | **Property** : actionButtons
 <ej-dialog id='dialog' [actionButtons]='actionbuttons'></ej-dialog>
 TS:
 actionbuttons: any;
constructor() { this.actionbuttons = ['minimize']; } | Not Applicable |

| Maximize | **Property** : actionButtons
 <ej-dialog id='dialog' [actionButtons]='actionbuttons'></ej-dialog>
 TS:
actionbuttons: any;
 constructor() { this.actionbuttons = ['maximize']; } | Not Applicable |

| Collapse /Expand | **Property** : actionButtons
 Method : collapse(), expand ()
 <ej-dialog id='dialog' [actionButtons]='actionbuttons'></ej-dialog>
 TS:
actionbuttons: any;
constructor() { this.actionbuttons = ['collapsible '];}
 \$(' #dialog').ejDialog('collapse');
 \$(' #dialog').ejDialog('expand') | Not Applicable |

| Event triggers when expanding the collapsed dialog | **Event**: expand
 <ej-dialog id='dialog' (expand)='expandAction(\$event)'></ej-dialog>
 TS:
 expandAction (event){ } | Not Applicable |

| Event triggers when collapsing the expanded dialog | **Event:** collapse
 <ej-dialog id='dialog' (collapse)='collapseAction(\$event)'></ej-dialog>
 TS:
 collapseAction (event){} | Not Applicable |

| Pin | **Property** : actionButtons
 <ej-dialog id='dialog' [actionButtons]='actionbuttons'></ej-dialog>
 TS:
 actionbuttons: any;
 constructor() { this.actionbuttons = [pin];} | Not Applicable |

| Header visibility | **Property:** showHeader
 <ej-dialog id='dialog' [showHeader]='true'></ej-dialog> | Not Applicable |

| Close on escape key press | **Property** : closeOnEscape
 <ej-dialog id='dialog' [closeOnEscape]='true'></ej-dialog> | **Property** : closeOnEscape
 <ejs-dialog id='dialog' [closeOnEscape]='true'></ejs-dialog> |

Footer

<!-- markdownlint-disable MD033 -->

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

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

| Footer Content | **Property** : footerTemplateId
 <ej-dialog id='dialog' [footerTemplateId]='sample'></ej-dialog> | **Property:** footerTemplate
 <ejs-dialog id='dialog' [footerTemplate]='<button>Submit</button>'></ejs-dialog> |

| Footer action buttons | Not applicable | **Property** : buttons
 <ejs-dialog id='dialog' [buttons]='buttons'></ejs-dialog>
 TS:
 public buttons: Object=[{ click: dialogBtnClick, buttonModel: {content: 'OK', isPrimary: true} }] |

| Footer visibility | **Property** : showFooter
 <ej-dialog id='dialog' [showFooter]='true'></ej-dialog> | Not Applicable |

Content

<!-- markdownlint-disable MD033 -->

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

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

| Dialog content | **Method** : setContent
 <ej-dialog id='dialog'></ej-dialog>
 \$(' #dialog').ejDialog('setContent', 'Dialog Content') | **Property** : content
 <ejs-dialog id='dialog' content=Dialog Content></ejs-dialog> |

| Loading content using AJAX request | **Property** : contentType, contentUrl
 <ej-dialog id='dialog' contentType='ajax' contentUrl=' '></ej-dialog> | Not Applicable |

| Event triggers after the dialog content loaded in DOM | **Event:** contentLoad
 <ej-dialog id='dialog' (contentLoad)='onLoad(\$event)'></ej-dialog>
 TS:
 onLoad (event){} | Not Applicable |

| Event trigger when fails to load ajax content | **Event:** ajaxError
 <ej-dialog id='dialog' (ajaxError)='onAjaxError (\$event)'></ej-dialog>
 TS:
 onAjaxError (event){} | Not Applicable |

| Event trigger when load ajax content successfully | **Event:** ajaxSuccess
 <ej-dialog id='dialog'
(ajaxSuccess)='onAjaxSuccess (\$event)'></ej-dialog>
 TS:
 onAjaxSuccess (event){} | Not
Applicable |

Animation

<!-- markdownlint-disable MD033 -->

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

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

| Enabling Animation | **Property** : enableAnimation
 <ej-dialog id='dialog'
[enableAnimation]='true'></ej-dialog> | Not Applicable |

| Animation effects | **Property** : animation.show.effect
 <ej-dialog id='dialog'
[animation]='animation'></ej-dialog>
 TS:
 public animation: object = {show: {effect:
'slide'}}; | **Property** : animationSettings.effect
 <ejs-dialog id='dialog' [animationSettings]='
animation '></ejs-dialog>
 TS:
 public animation: Object = { effect: 'Zoom'}; |

| Animation duration | **Property:** animation.show.duration
 <ej-dialog id='dialog'
[animation]='animation'></ej-dialog>
 TS:
 public animation: object = {show: {effect: 'slide',
duration: 500}}; | **Property** : animationSettings.duration
 <ejs-dialog id='dialog'
[animationSettings]='animation'></ejs-dialog>
 TS:
 public animation: Object = { effect:
'Zoom',duration: 500}; |

| Animation delay | Not applicable | **Property:** animationSettings.delay
 <ejs-dialog id='dialog'
[animationSettings]=' animationDelay '></ejs-dialog>
 TS:
 public animationDelay: Object
= { effect: 'Zoom',duration: 500,delay: 300 }; |

Draggable and resizing

<!-- markdownlint-disable MD033 -->

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

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

| Draggable dialog | **Property** : allowDraggable
 <ej-dialog id='dialog'
[allowDraggable]='true'></ej-dialog> | **Property** : allowDragging
 <ejs-dialog id='dialog'
[allowDragging]=' true '></ejs-dialog> |

| Event triggers when the user drags the dialog | **Event:** drag
 <ej-dialog id='dialog'
(drag)='onDrag (\$event)'></ej-dialog>
 TS:
 onDrag (event){} | **Event:** drag
 <ejs-
dialog id='dialog' (drag)='onDrag()'></ejs-dialog>
 TS:
 public onDrag: EmitType<Object> =
() => {} |

| Event triggers when the start to drag the dialog | **Event:** dragStart
 <ej-dialog id='dialog'
(dragStart)='onDragStart (\$event)'></ej-dialog>
 TS:
 onDragStart (event){} | **Event:**
dragStart
 <ejs-dialog id='dialog' (dragStart)='onDragStart()'></ejs-dialog>
 TS:

public onDragStart: EmitType<Object> = () => {} |

| Event triggers when the stops to drag the dialog | **Event:** dragStop
 <ej-dialog id='dialog'
(dragStop)='onDragStop (\$event)'></ej-dialog>
 TS:
 onDragStop (event){} | **Event:**

dragStop
 <ejs-dialog id='dialog' (dragStop)='onDragStop()'></ejs-dialog>
TS:

public onDragStop: EmitType<Object> = () => {} |

| Resizing dialog | **Property** : enableResize
 <ej-dialog id='dialog' [enableResize]='true'></ej-dialog> | Not applicable |

| Event triggers when resizing the dialog | **Event:** resize
 <ej-dialog id='dialog' (resize)='onReSize (\$event)'></ej-dialog>
 TS:
 onReSize (event){} | Not Applicable |

| Event triggers when starts to resizing the dialog | **Event:** resizeStart
 <ej-dialog id='dialog' (resizeStart)='onResizeStart (\$event)'></ej-dialog>
 TS:
 onResizeStart (event) {} | Not Applicable |

| Event triggers when the stops to resizing the dialog | **Event:** resizeStop
 <ej-dialog id='dialog' (resizeStop)='onResizeStop (\$event)'></ej-dialog>
 TS:
 onResizeStop (event) {} | Not Applicable |

Target

<!-- markdownlint-disable MD033 -->

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

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

| Target element to append dialog in document | **Property** : target
 <ej-dialog id='dialog' target='#dialogTarget'></ej-dialog> | **Property:** target
 <ejs-dialog id='dialog' target='#dialogTarget'></ejs-dialog> |

| Element for draggable area | **Property** : containment
 <ej-dialog id='dialog' containment='#dragArea'></ej-dialog> | Not applicable |

Position

<!-- markdownlint-disable MD033 -->

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

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

| Customizing dialog position using X, Y coordinate values | **Property** : position
 <ej-dialog id='dialog' [position]='dialogPosition'></ej-dialog>
public dialogPosition: any = { position: { X: 300, Y: 100 }} | **Property** : position
 <ejs-dialog id='dialog' [position]='dialogPosition'></ejs-dialog>
 public dialogPosition: object = { position: { X: 300, Y: 100 }} |

| positioning dialog using position values | Not Applicable | **Property:** position
 <ejs-dialog id='dialog' [position]=' dialogPosition'></ejs-dialog>
 public dialogPosition: object = { position: { X: 'Center', Y: 'Center'}} |

Visibility

<!-- markdownlint-disable MD033 -->

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

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

| Render dialog in visible/hidden state | **Property:** showOnInit
 <ej-dialog id='dialog' [showOnInit]= 'true'></ej-dialog> | **Property:** visible
 <ejs-dialog id='dialog' [visible]='true'></ejs-dialog> |

Dialog Mode

<!-- markdownlint-disable MD033 -->

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

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

| Render modal dialog | **Property :** enableModal
 <ej-dialog id='dialog' [enableModal]='true'></ej-dialog> | **Property :** isModal
 <ejs-dialog id='dialog' [isModal]= 'true'></ejs-dialog> |

Tooltip

<!-- markdownlint-disable MD033 -->

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

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

| Sets the tooltip for dialog buttons | **Property :** tooltip
 <ej-dialog id='dialog' [tooltip]='tooltip'></ej-dialog>
 TS:
 public tooltip: object { close: 'Exit' }; | No Separate Property for tooltip. It renders based on locale text. |

Control State

<!-- markdownlint-disable MD033 -->

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

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

| Enable/Disable the control | **Property :** enabled
 <ej-dialog id='dialog' [enabled]='false'></ej-dialog> | Not Applicable |

| Enable/ Disable page scrolling | **Property:** backgroundScroll
 <ej-dialog id='dialog' [backgroundScroll]= 'false'></ej-dialog> | No separate Property for disabling page scroll. By default, scrolling prevented for modal dialog |

State Maintenance

<!-- markdownlint-disable MD033 -->

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

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

| Save the model values in local storage or cookies | **Property :** enablePersistence
 <ej-dialog id='dialog' [enablePersistence]= 'true'></ej-dialog> | **Property :** enablePersistence
 <ejs-dialog id='dialog' [enablePersistence]='true'></ejs-dialog> |

Common

<!-- markdownlint-disable MD033 -->

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

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

| Adjusting Height | **Property** : height
 <ej-dialog id='dialog' height='400'></ej-dialog> | **Property** : height
 <ejs-dialog id='dialog' height='50%'></ejs-dialog> |

| Adjusting width | **Property**: width
 <ej-dialog id='dialog' width='400'></ej-dialog> | **Property** : width
 <ejs-dialog id='dialog' width='50%'></ejs-dialog> |

| Adding custom class | **Property**: cssClass
 <ej-dialog id='dialog' cssClass = ' custom-class'></ej-dialog> | **Property**: cssClass
 <ejs-dialog id='dialog' cssClass ='custom-class'></ejs-dialog> |

| Adding zIndex | **Property**: zIndex
 <ej-dialog id='dialog' [zIndex] ='2000'></ej-dialog> | **Property**: zIndex
 <ejs-dialog id='dialog' [zIndex]='2000'></ejs-dialog> |

| Maximum height | **Property**: maxHeight
 <ej-dialog id='dialog' maxHeight ='600'></ej-dialog> | Not Applicable |

| Maximum width | **Property**: maxWidth
 <ej-dialog id='dialog' maxWidth ='600'></ej-dialog> | Not Applicable |

| Minimum height | **Property**: minHeight
 <ej-dialog id='dialog' minHeight ='300'></ej-dialog> | Not Applicable |

| Minimum width | **Property**: minWidth
 <ej-dialog id='dialog' minWidth ='300'></ej-dialog> | Not Applicable |

| Adding html attributes | **Property**: htmlAttributes
 <ej-dialog id='dialog' [htmlAttributes] ='htmlAttributes'></ej-dialog>
 TS:
 public htmlAttributes: object { class: 'my-class' } | Not Applicable |

| Custom icon in the header | **Property**: faviconCSS
 <ej-dialog id='dialog' [faviconCSS] ='custom-icon'></ej-dialog> | Not Applicable |

| Rounded corner appearance | **Property**: showRoundedCorner
 <ej-dialog id='dialog' [showRoundedCorner] ='true'></ej-dialog> | Not Applicable |

| Make control flexible for mobile view | **Property**: isResponsive
 <ej-dialog id='dialog' [isResponsive] ='true'></ej-dialog> | Not Applicable |

| Close the Dialog | **Method**: close()
 <ej-dialog id='dialog'></ej-dialog>
 \$(''#dialog').ejDialog('close') | **Method** : hide()
 <ejs-dialog id='dialog' #defaultDialog></ejs-dialog>
 TS:
 @ViewChild('defaultDialog') public defaultDialog: DialogComponent;
 defaultDialog.hide() |

| Event triggers before the dialog closes | **Event**: beforeClose
 <ej-dialog id='dialog' (beforeClose)='onBeforeClose (\$event)'></ej-dialog>
 TS:
 onBeforeClose (event) {} | **Event**: beforeClose
 <ejs-dialog id='dialog' (beforeClose)='onBeforeClose ()'></ejs-dialog>
 TS:
 onBeforeClose () {} |

| Event triggers when the dialog closes | **Event**: close
 <ej-dialog id='dialog' (close)='onClose (\$event)'></ej-dialog>
 TS:
 onClose (event) {} | **Event**: close
 <ejs-dialog id='dialog' (close)='onClose ()'></ejs-dialog>
 TS:
 onClose () {} |

| Destroy the control | **Method:** destroy()
 <ej-dialog id='dialog'></ej-dialog>
 \$(' #dialog').ejDialog('destroy') | **Method:** destroy()
 <ejs-dialog id='dialog' #defaultDialog></ejs-dialog>
 TS:
 @ViewChild('defaultDialog') public defaultDialog: DialogComponent;
 defaultDialog. destroy () |

| Focus the dialog element | **Method:** focus()
 <ej-dialog id='dialog'></ej-dialog>
 \$(' #dialog').ejDialog('focus') | Not Applicable |

| Check whether the dialog is open | **Method:** isOpen()
 <ej-dialog id='dialog'></ej-dialog>
 \$(' #dialog').ejDialog('isOpen') | Not Applicable |

| Maximize the dialog | **Method:** maximize()
 <ej-dialog id='dialog'></ej-dialog>
 \$(' #dialog').ejDialog('maximize') | Not Applicable |

| Minimize the dialog | **Method:** minimize()
 <ej-dialog id='dialog'></ej-dialog>
 \$(' #dialog').ejDialog('minimize') | Not Applicable |

| Open the dialog | **Method:** open()
 <ej-dialog id='dialog'></ej-dialog>
 \$(' #dialog').ejDialog('open') | **Method :** show()
 <ejs-dialog id='dialog' #defaultDialog></ejs-dialog>
 TS:
 @ViewChild('defaultDialog') public defaultDialog: DialogComponent; defaultDialog. show (); |

| Event trigger before the dialog opens | **Event:** beforeOpen
 <ej-dialog id='dialog' (beforeOpen)='onBeforeOpen (\$event)'></ej-dialog>
 TS:
 onBeforeOpen (event) {} | **Event:** beforeOpen
 <ejs-dialog id='dialog' (beforeOpen)=' onBeforeOpen()'></ejs-dialog>
 TS:
 onBeforeOpen () {} |

| Event triggers when the opens the dialog | **Event:** open
 <ej-dialog id='dialog' (open)='onOpen (\$event)'></ej-dialog>
 TS:
 onOpen (event) {} | **Event:** open
 <ejs-dialog id='dialog' (open)=' onOpen()'></ejs-dialog>
 TS:
 onOpen () {} |

| Refresh the dialog | **Method:** refresh()
 <ej-dialog id='dialog'></ej-dialog>
 \$(' #dialog').ejDialog('refresh') | **Method :** refreshPosition()
 <ejs-dialog id='dialog' #defaultDialog></ejs-dialog>
 TS:
 @ViewChild('defaultDialog') public defaultDialog: DialogComponent; defaultDialog. refreshPosition(); |

| Pin/ unpin the dialog | **Method:** pin
 <ej-dialog id='dialog'></ej-dialog>
 \$(' #dialog').ejDialog('pin');
 \$(' #dialog').ejDialog('unpin'); | **Not Applicable** |

| Event triggers after the dialog created successfully | **Event:** create
 <ej-dialog id='dialog' (create)='onCreate (\$event)'></ej-dialog>
 TS:
 onCreate (event) {} | **Event :** created
 <ejs-dialog id='dialog' (created)=' onCreated()'></ejs-dialog>
 TS:
 onCreated () {} |

| Event triggers when the control destroyed successfully | **Event:** destroy
 <ej-dialog id='dialog' (destroy)='onDestroy (\$event)'></ej-dialog>
 TS:
 onDestroy (event) {} | Not Applicable |

| Event triggers on clicking on modal dialog overlay | **Not Applicable** | **Event :** overlayClick
 <ejs-dialog id='dialog' (overlayClick)=' onOverlayClick()'></ejs-dialog>
 TS:
 onOverlayClick () {} |

DocumentEditor

Overview

The Document Editor component is used to create, edit, view, and print Word documents in web applications. All the user interactions and editing operations that run purely in the client-side provides much faster editing experience to the users.

Key Features

- [Opens](#) the native Syncfusion Document Text (*.sfdt) format documents in the client-side.
- [Saves the documents](#) in the client-side as Syncfusion Document Text (.sfdt) and Word document (.docx).
- Supports document elements like text, [image](#), [table](#), fields, [bookmark](#), [shapes](#), [section](#), [header and footer](#).
- Supports the commonly used fields like [hyperlink](#), page number, page count, and table of contents.
- Supports formats like [text](#), [paragraph](#), [bullets and numbering](#), [table](#), [page settings](#), etc.
- Provides support to create, edit, and apply [paragraph and character styles](#).
- Provides support to [find and replace](#) text within the document.
- Supports all the common editing and formatting operations along with [undo and redo](#).
- Provides support to [cut](#), [copy](#), and [paste](#) rich text contents within the component. Also allows pasting simple text to and from other applications.
- Provides support to insert, and edit [form fields](#).
- Provides support to insert, and edit [comments](#).
- Provides support to track the [inserted and deleted content](#).
- Provides support to perform [spell checking](#) for any input text
- Allows user interactions like [zoom](#), [scroll](#), select contents through touch, mouse, and keyboard.
- Provides intuitive UI options like context menu, [dialogs](#), and [navigation pane](#).
- [Localizes](#) all the static text to any desired language.
- Allows to create a lightweight Word viewer using module injection to view and [prints](#) Word documents.
- Provides a [server-side helper library](#) to open the Word documents like DOCX, DOC, WordML, RTF, and Text, by converting it to SFDT file format.

Supported Web platforms

- [Javascript\(ES5\)](#)
- [Javascript](#)
- [Angular](#)
- [React](#)
- [Vue](#)
- [ASP.NET Core](#)
- [ASP.NET MVC](#)
- [Blazor](#)

Supported platforms for server-side dependencies

You can deploy web APIs for server-side dependencies of Document Editor component in the following platforms.

- [ASP.NET Core](#)
- [ASP.NET MVC](#)
- [Java](#)

To know more about server-side dependencies, refer this [page](#).

Which operations require server-side interaction

- Open file formats other than SFDT
- Paste with formatting
- Restrict editing
- Spellcheck
- Save as file formats other than SFDT and DOCX

Note: If you don't require the above functionalities then you can deploy as pure client-side component without any server-side interactions.

Getting started with Angular Document editor component

This section explains the steps to create a Word document editor within your application and demonstrates the basic usage of the Document Editor component.

Dependencies

The list of dependencies required to use the Document Editor component in your application is given below:

```
`javascript
|-- @syncfusion/ej2-angular-documenteditor
|-- @syncfusion/ej2-angular-base
|-- @syncfusion/ej2-documenteditor
|-- @syncfusion/ej2-base
|-- @syncfusion/ej2-buttons
|-- @syncfusion/ej2-compression
|-- @syncfusion/ej2-data
|-- @syncfusion/ej2-dropdowns
|-- @syncfusion/ej2-file-utils
|-- @syncfusion/ej2-inputs
|-- @syncfusion/ej2-lists
|-- @syncfusion/ej2-navigations
|-- @syncfusion/ej2-popups
|-- @syncfusion/ej2-splitbuttons
|-- @syncfusion/ej2-charts
`
```

Server side dependencies

The Document Editor component requires server-side interactions for the following operations:

- [Open file formats other than SFDT](#)
- [Paste with formatting](#)
- [Restrict editing](#)
- [SpellCheck](#)
- [Save as file formats other than SFDT and DOCX](#)

Note: If you don't require the above functionalities then you can deploy as pure client-side component without any server-side interactions.

To know about server-side dependencies, please refer this [page](#).

Setup your development environment

- To setup basic Angular sample use following commands.

```
`javascript
git clone https://github.com/angular/quickstart.git quickstart
cd quickstart
npm install
`
```

For more information, refer to [Angular sample setup](#)

Installing Syncfusion DocumentEditor 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-documenteditor](#) package to the application.

```
`bash
ng add @syncfusion/ej2-angular-documenteditor
`
```

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-documenteditor@ngcc](#) package to the application.

```
`bash
```

```
npm install @syncfusion/ej2-angular-documenteditor@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-documenteditor:"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.

Configuring system JS

[Syncfusion DocumentEditor packages](#) need to be mapped in `systemjs.config.js` configuration file.

Syncfusion `ej2-angular-documenteditor` packages have to be mapped in the `systemjs.config.js` configuration file.

```
`javascript
```

```
/
```

- System configuration for Angular samples
- Adjust as necessary for your application needs.

```
*/
```

```
(function (global) {
```

```
System.config({
```

```
paths: {
```

```
// paths serve as alias
```

```
'npm:': 'node_modules/',
```

```
'syncfusion:': './node_modules/@syncfusion/',
```

```
},
```

```
// map tells the System loader where to look for things
```

```
map: {
```

```
// our app is within the app folder
```

```
'app': 'app',
```

```
// angular bundles
```

```
'@angular/core': 'npm:@angular/core/bundles/core.umd.js',
```

```
'@angular/common': 'npm:@angular/common/bundles/common.umd.js',
```

```
'@angular/compiler': 'npm:@angular/compiler/bundles/compiler.umd.js',
 '@angular/platform-browser': 'npm:@angular/platform-browser/bundles/platform-browser.umd.js',
 '@angular/platform-browser-dynamic': 'npm:@angular/platform-browser-dynamic/bundles/platform-
 browser-dynamic.umd.js',
 '@angular/http': 'npm:@angular/http/bundles/http.umd.js',
 '@angular/router': 'npm:@angular/router/bundles/router.umd.js',
 '@angular/forms': 'npm:@angular/forms/bundles/forms.umd.js',
 // syncfusion bundles
 "@syncfusion/ej2-base": "syncfusion:ej2-base/dist/ej2-base.umd.min.js",
 "@syncfusion/ej2-buttons": "syncfusion:ej2-buttons/dist/ej2-buttons.umd.min.js",
 "@syncfusion/ej2-splitbuttons": "syncfusion:ej2-splitbuttons/dist/ej2-splitbuttons.umd.min.js",
 "@syncfusion/ej2-data": "syncfusion:ej2-data/dist/ej2-data.umd.min.js",
 "@syncfusion/ej2-dropdowns": "syncfusion:ej2-dropdowns/dist/ej2-dropdowns.umd.min.js",
 "@syncfusion/ej2-inputs": "syncfusion:ej2-inputs/dist/ej2-inputs.umd.min.js",
 "@syncfusion/ej2-lists": "syncfusion:ej2-lists/dist/ej2-lists.umd.min.js",
 "@syncfusion/ej2-navigations": "syncfusion:ej2-navigations/dist/ej2-navigations.umd.min.js",
 "@syncfusion/ej2-compression": "syncfusion:ej2-compression/dist/ej2-compression.umd.min.js",
 "@syncfusion/ej2-popups": "syncfusion:ej2-popups/dist/ej2-popups.umd.min.js",
 "@syncfusion/ej2-file-utils": "syncfusion:ej2-file-utils/dist/ej2-file-utils.umd.min.js",
 "@syncfusion/ej2-office-chart": "syncfusion:ej2-office-chart/dist/ej2-office-chart.umd.min.js",
 "@syncfusion/ej2-calendars": "syncfusion:ej2-calendars/dist/ej2-calendars.umd.min.js",
 "@syncfusion/ej2-pdf-export": "syncfusion:ej2-pdf-export/dist/ej2-pdf-export.umd.min.js",
 "@syncfusion/ej2-svg-base": "syncfusion:ej2-svg-bases/dist/ej2-svg-base.umd.min.js",
 "@syncfusion/ej2-charts": "syncfusion:ej2-charts/dist/ej2-charts.umd.min.js",
 "@syncfusion/ej2-documenteditor": "syncfusion:ej2-documenteditor/dist/ej2-
 documenteditor.umd.min.js",
 "@syncfusion/ej2-angular-base": "syncfusion:ej2-angular-base/dist/ej2-angular-base.umd.min.js",
 "@syncfusion/ej2-angular-documenteditor": "syncfusion:ej2-angular-documenteditor/dist/ej2-angular-
 documenteditor.umd.min.js",
 // other libraries
 'rxjs': 'npm:rxjs',
 'angular-in-memory-web-api': 'npm:angular-in-memory-web-api/bundles/in-memory-web-api.umd.js'
 },
 // packages tells the System loader how to load when no filename and/or no extension
```

```
packages: {
  app: {
    defaultExtension: 'js',
    meta: {
      './*.js': {
        loader: 'systemjs-angular-loader.js'
      }
    },
  },
  rxjs: {
    defaultExtension: 'js'
  }
};
})(this);
`
```

Adding CSS reference

To render the Documenteditor component, need to import document editor and its dependent component's styles.

This can be referenced in your `[src/styles/styles.css]` using the 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-lists/styles/material.css';
@import '../node_modules/@syncfusion/ej2-navigations/styles/material.css';
@import '../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-dropdowns/styles/material.css';
@import '../node_modules/@syncfusion/ej2-documenteditor/styles/material.css';
`
```

To know about individual component CSS, please refer to [Individual Component theme files](#) section.

In case, if you want to make use of the combined CSS files of entire components, then you can avail it from the root folder of Essential JS 2 package and reference it with the code shown below.


```
`css
@import '../node_modules/@syncfusion/ej2/material.css';
`
```

Adding Component

You can add `DocumentEditorContainer` Component with predefined toolbar and properties pane options or `DocumentEditor` component with customize options.

Note: Starting from `v19.3.0.x`, we have optimized the accuracy of text size measurements such as to match Microsoft Word pagination for most Word documents. This improvement is included as default behavior along with an optional API [to disable it and retain the document pagination behavior of older versions](#).

DocumentEditor Component

DocumentEditor Component is used to create , view and edit word documents. In this , you can customize the UI options based on your requirements to modify the document.

Registering DocumentEditor Module

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

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

Adding DocumentEditor component

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

Add the Angular Document Editor by using `<ejs-documenteditor>` 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 DocumentEditor component
  template: <ejs-documenteditor
  serviceUrl="https://ej2services.syncfusion.com/production/web-
  services/api/documenteditor/"> </ejs-documenteditor>
})
export class AppComponent implements OnInit {
  ngOnInit(): void {
  }
}
`

```

Run the DocumentEditor application

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

```

`javascript
npm start
`

```

Output will be displayed as follows.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DocumentEditorAllModule } from '@syncfusion/ej2-angular-
documenteditor'
import { Component, ViewEncapsulation } from '@angular/core';
import {
  DocumentEditorComponent, PrintService, SfdtExportService,
  WordExportService, TextExportService, SelectionService,
  SearchService, EditorService, ImageResizerService, EditorHistoryService,
  ContextMenuService,
  OptionsPaneService, HyperlinkDialogService, TableDialogService,
  BookmarkDialogService, TableOfContentsDialogService,
  PageSetupDialogService, StyleDialogService, ListDialogService,
  ParagraphDialogService, BulletsAndNumberingDialogService,
  FontDialogService, TablePropertiesDialogService,
  BordersAndShadingDialogService, TableOptionsDialogService,
  CellOptionsDialogService, StylesDialogService
} from '@syncfusion/ej2-angular-documenteditor';
@Component({
  imports: [

    DocumentEditorAllModule
  ],
  standalone: true,

```

```

        selector: 'app-container',
        template: `<ejs-documenteditor id="container"
serviceUrl="https://ej2services.syncfusion.com/production/web-
services/api/documenteditor/" height="330px" style="display:block"
[isReadOnly]=false [enableSelection]=true
        [enablePrint]=true [enableSfdtExport]=true [enableWordExport]=true
[enableOptionsPane]=true [enableContextMenu]=true
        [enableHyperlinkDialog]=true [enableBookmarkDialog]=true
[enableTableOfContentsDialog]=true [enableSearch]=true
        [enableParagraphDialog]=true [enableListDialog]=true
[enableTablePropertiesDialog]=true [enableBordersAndShadingDialog]=true
        [enablePageSetupDialog]=true [enableStyleDialog]=true
[enableFontDialog]=true [enableTableOptionsDialog]=true
        [enableTableDialog]=true [enableImageResizer]=true [enableEditor]=true
[enableEditorHistory]=true>
        </ejs-documenteditor>`,
        encapsulation: ViewEncapsulation.None,
        providers: [PrintService, SfdtExportService, WordExportService,
TextExportService, SelectionService, SearchService, EditorService,
        ImageResizerService, EditorHistoryService, ContextMenuService,
OptionsPaneService, HyperlinkDialogService, TableDialogService,
        BookmarkDialogService, TableOfContentsDialogService,
PageSetupDialogService, StyleDialogService, ListDialogService,
        ParagraphDialogService, BulletsAndNumberingDialogService,
FontDialogService, TablePropertiesDialogService,
        BordersAndShadingDialogService, TableOptionsDialogService,
CellOptionsDialogService, StylesDialogService]
    })
    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));

```

DocumentEditorContainer Component

DocumentEditorContainer is a predefined component which wraps DocumentEditor, Toolbar, Properties pane, and Status bar into a single component. And the toolbar and properties pane is used to view and modify the document in DocumentEditor through public APIs available in it.

Registering DocumentEditorContainer Module

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

`typescript

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { DocumentEditorContainerModule } from '@syncfusion/ej2-angular-documenteditor';
import { AppComponent } from './default.component';

```

```

/
    • Module
*/
@NgModule({
  imports: [
    BrowserModule,
    DocumentEditorContainerModule
  ],
  declarations: [AppComponent],
  bootstrap: [AppComponent]
})
export class AppModule { }
`

```

Adding DocumentEditorContainer component

Modify the template in [src/app/app.component.ts] file to render the Document Editor Container component.

Add the Angular Document Editor Container by using `<ejs-documenteditor>` selector in `template` section of the app.component.ts file.

```

`typescript
import { Component, OnInit } from '@angular/core';
import { ToolbarService } from '@syncfusion/ej2-angular-documenteditor';
@Component({
  selector: 'app-container',
  // specifies the template string for the DocumentEditorContainer component
  template: <ejs-documenteditorcontainer
  serviceUrl="https://ej2services.syncfusion.com/production/web-
  services/api/documenteditor/" [enableToolbar]=true> </ejs-documenteditorcontainer>,
  providers: [ToolbarService]
})
export class AppComponent implements OnInit {
  ngOnInit(): void {
  }
}
`

```

Run the DocumentEditorContainer application

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

```
`javascript
```

```
npm start
```

```
,
```

DocumentEditorContainer output will be displayed as follows.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DocumentEditorContainerModule } from '@syncfusion/ej2-angular-documenteditor'
import { Component, OnInit } from '@angular/core';
import { ToolbarService } from '@syncfusion/ej2-angular-documenteditor';
@Component({
  imports: [

    DocumentEditorContainerModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the DocumentEditorContainer component
  template: `<ejs-documenteditorcontainer
serviceUrl="https://ej2services.syncfusion.com/production/web-services/api/documenteditor/" height="600px" style="display:block"
[enableToolbar]=true> </ejs-documenteditorcontainer>`,
  providers: [ToolbarService]
})
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));
```

Frequently Asked Questions

- [How to localize the Documenteditor container.](#)
- [How to load the document by default.](#)
- [How to customize tool bar.](#)
- [How to resize Document editor component?](#)

Feature module in Angular Document editor component

Document Editor features are segregated into individual feature-wise modules to enable selective referencing. By default, the document editor displays the document in read-only mode. The required modules should be injected to extend its functionality. The following are the selective modules of document editor that can be included as required:

- **PrintService** - Prints the document.
- **SfdtExportService** - Exports the document as Syncfusion Document Text (.SFDt) file.
- **SelectionService** - Selects a portion of the document and copy it to the clipboard.
- **SearchService** - Searches specific text and navigate between the results.
- **WordExportService** - Exports the document as Word Document (.DOCX) file.
- **TextExportService** - Exports the document as Text Document (.TXT) file.
- **EditorService** - Performs all kind of editing operations.
- **EditorHistoryService** - Maintains the history of editing operations so that you can perform undo and redo at any time.
- User interface options such as context menu, options pane, image resizer, and dialog are available as individual modules.

In addition to injecting the required modules in your application, enable corresponding properties to extend the functionality for a document editor instance.

Refer to the following table.

Module	Dependent modules to be injected for extending the functionality of document editor in your application	Property to enable the functionality for a document editor instance
--------	---------------------------------------------------------------------------------------------------------	---------------------------------------------------------------------

---	---	---
PrintService	PrintService	<ejs-documenteditor [enablePrint]=true></ejs-documenteditor>
SfdtExportService	SfdtExportService	<ejs-documenteditor [enableSfdtExport]=true></ejs-documenteditor>
SelectionService	SelectionService	<ejs-documenteditor [enableSelection]=true></ejs-documenteditor>
SearchService	SelectionService, SearchService	<ejs-documenteditor [enableSearch]=true></ejs-documenteditor>
WordExportService	SfdtExportService, WordExportService	<ejs-documenteditor [enableWordExport]=true></ejs-documenteditor>
TextExportService	SfdtExportService, TextExportService	<ejs-documenteditor [enableTextExport]=true></ejs-documenteditor>
EditorService	SelectionService, EditorService	<ejs-documenteditor [isReadOnly]: false [enableEditor]=true></ejs-documenteditor>
EditorHistoryService	SelectionService, EditorService, EditorHistoryService	<ejs-documenteditor [isReadOnly]: false [enableEditor]=true [enableEditorHistory]=true></ejs-documenteditor>
OptionsPaneService(Find)	SelectionService, SearchService, OptionsPaneService	<ejs-documenteditor [enableSearch]=true [enableOptionsPane]=true></ejs-documenteditor>

```

|OptionsPaneService(Find and Replace)|SelectionService, SearchService, EditorService,
OptionsPaneService|<ejs-documenteditor [isReadOnly]: false [enableEditor]=true
[enableSearch]=true [enableOptionsPane]=true></ejs-documenteditor>|

|ContextMenuService|SelectionService, ContextMenuService|<ejs-documenteditor
[enableSelection]=true [enableContextMenu]=true></ejs-documenteditor>|

|ImageResizerService|SelectionService, EditorService, ImageResizerService|<ejs-documenteditor
[isReadOnly]: false [enableEditor]=true [enableImageResizer]=true></ejs-documenteditor>|

|HyperlinkDialogService|SelectionService, EditorService, HyperlinkDialogService|<ejs-
documenteditor [isReadOnly]: false [enableEditor]=true [enableHyperlinkDialog]=true></ejs-
documenteditor>|

|TableDialogService|SelectionService, EditorService, TableDialogService|<ejs-documenteditor
[isReadOnly]: false [enableEditor]=true [enableTableDialog]=true></ejs-documenteditor>|

|FontDialogService|SelectionService, EditorService, FontDialogService|<ejs-documenteditor
[isReadOnly]: false [enableEditor]=true [enableFontDialog]=true></ejs-documenteditor>|

|ParagraphDialogService|SelectionService, EditorService, ParagraphDialogService|<ejs-
documenteditor [isReadOnly]: false [enableEditor]=true [enableParagraphDialog]=true></ejs-
documenteditor>|

|BookmarkDialogService|SelectionService, EditorService, BookmarkDialogService|<ejs-
documenteditor [isReadOnly]: false [enableEditor]=true [enableBookmarkDialog]=true></ejs-
documenteditor>|

|PageSetupDialogService|SelectionService, EditorService, PageSetupDialogService|<ejs-
documenteditor [isReadOnly]: false [enableEditor]=true [enablePageSetupDialog]=true></ejs-
documenteditor>|

|TableOfContentsDialogService|SelectionService, EditorService,
TableOfContentsDialogService|<ejs-documenteditor [isReadOnly]: false [enableEditor]=true
[enableTableOfContentsDialog]=true></ejs-documenteditor>|

|ListDialog|SelectionService, EditorService, ListDialog|<ejs-documenteditor [isReadOnly]: false
[enableEditor]=true [enableListDialog]=true></ejs-documenteditor>|

|TablePropertiesDialog|SelectionService, EditorService, TablePropertiesDialog|<ejs-
documenteditor [isReadOnly]: false [enableEditor]=true
[enableTablePropertiesDialog]=true></ejs-documenteditor>|

|CellOptionsDialogService|SelectionService, EditorService, CellOptionsDialogService|<ejs-
documenteditor [isReadOnly]: false [enableEditor]=true
[enableTablePropertiesDialog]=true></ejs-documenteditor>|

|BordersAndShadingDialogService|SelectionService, EditorService,
BordersAndShadingDialogService|<ejs-documenteditor [isReadOnly]: false [enableEditor]=true
[enableBordersAndShadingDialog]=true></ejs-documenteditor>|

```

```
|TableOptionsDialogService|SelectionService, EditorService, TableOptionsDialogService|<ejs-
documenteditor [isReadOnly]: false [enableEditor]=true
[enableTableOptionsDialog]=true></ejs-documenteditor>|
```

```
|StylesDialogService|SelectionService, EditorService, StylesDialogService,StyleDialogService|<ejs-
documenteditor [isReadOnly]: false [enableEditor]=true [enableStyleDialog]=true
,[enableStylesDialog]=true></ejs-documenteditor>|
```

```
|StyleDialogService|SelectionService, EditorService, StyleDialogService|<ejs-documenteditor
[isReadOnly]: false [enableEditor]=true [enableStyleDialog]=true></ejs-documenteditor>|
```

```
|BulletsAndNumberingDialogService|SelectionService, EditorService,
BulletsAndNumberingDialogService|<ejs-documenteditor [isReadOnly]: false
[enableEditor]=true [enableStyleDialog]=true></ejs-documenteditor>|
```

These modules should be injected into the `providers` section of root `NgModule` or component class.

Import in Angular Document editor component

In Document Editor, the documents are stored in its own format called **Syncfusion Document Text (SFDT)**.

The following example shows how to open SFDT data in Document Editor.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { DocumentEditorAllModule } from '@syncfusion/ej2-angular-
documenteditor'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import {
    DocumentEditorComponent
} from '@syncfusion/ej2-angular-documenteditor';
@Component({
    imports: [

        ButtonModule,
        DocumentEditorAllModule
    ],
    standalone: true,
    selector: 'app-container',
    //specifies the template string for the Document Editor component
    template: `<div>
        <ejs-documenteditor #document_editor id="container"
        (created)="onCreated()" height="330px" style="display:block"></ejs-
        documenteditor>
        </div>`,
    encapsulation: ViewEncapsulation.None,
})
export class AppComponent {
    @ViewChild('document_editor')
    public documentEditor?: DocumentEditorComponent;
    onCreated() {
        if ((this.documentEditor as
        DocumentEditorComponent).isDocumentLoaded) {
```



```

        let sfdt: string = `{
            "sections": [
                {
                    "blocks": [
                        {
                            "inlines": [
                                {
                                    "characterFormat": {
                                        "bold": true,
                                        "italic": true
                                    },
                                    "text": "Hello World"
                                }
                            ]
                        }
                    ]
                },
                "headersFooters": {
                }
            ]
        }`;
        //Open the sfdt document in Document Editor.
        (this.documentEditor as DocumentEditorComponent).open(sfdt);
    }
}

```

MAIN.TS

```

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

```

Import document from local machine

The following example shows how to import document from local machine.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { DocumentEditorAllModule } from '@syncfusion/ej2-angular-documenteditor'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import {
    DocumentEditorComponent
} from '@syncfusion/ej2-angular-documenteditor';
@Component({
    imports: [
        ButtonModule,
        DocumentEditorAllModule
    ],
    standalone: true,
    selector: 'app-container',

```

```

//specifies the template string for the Document Editor component
template: `<div>
  <input id="open_sfdt" #open_sfdt style="position:fixed; left:-100em"
type="file" (change)="onFileChange($event)" accept=".sfdt"/>
  <button ejs-button (click)="onFileOpenClick()" >Import</button>
  <ejs-documenteditor #document_editor id="container" height="330px"
style="display:block"></ejs-documenteditor>
</div>`,
encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild('document_editor')
  public documentEditor?: DocumentEditorComponent;
  public onFileOpenClick(): void {
    //Open file picker.
    (document.getElementById('open_sfdt') as HTMLElement).click();
  }
  public onFileChange(e: any): void {
    if (e.target.files[0]) {
      //Get the selected file.
      let file = e.target.files[0];
      if (file.name.substr(file.name.lastIndexOf('.')) === '.sfdt') {
        let fileReader: FileReader = new FileReader();
        fileReader.onload = (e: any) => {
          let contents: string = e.target.result;
          //Open the sfdt document in Document Editor.
          (this.documentEditor as
DocumentEditorComponent).open(contents);
        };
        //Read the input file.
        fileReader.readAsText(file);
        (this.documentEditor as
DocumentEditorComponent).documentName = file.name.substr(0,
file.name.lastIndexOf('.'));
      }
    }
  }
}

```

MAIN.TS

```

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

```

Convert word documents into SFDT

You can convert word documents into SFDT format using the .NET Standard library

[Syncfusion.EJ2.WordEditor.AspNet.Core](#) by the web API service implementation. This library helps you to convert word documents (.dotx,.docx,.docm,.dot,.doc), rich text format documents (.rtf), and text documents (.txt) into SFDT format.

Note: The Syncfusion Document Editor component's document pagination (page-by-page display) can't be guaranteed for all the Word documents to match the pagination of Microsoft Word application. For

more information about [why the document pagination \(page-by-page display\) differs from Microsoft Word](#)

Please refer the following example for converting word documents into SFDT.

```
`typescript
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import {
  DocumentEditorComponent
} from '@syncfusion/ej2-angular-documenteditor';
@Component({
  selector: 'app-container',
  template: `<div>
    <input id="opensfdt" #opensfdt style="position:fixed; left:-100em" type="file"
    (change)="onFileChange($event)" accept=".dotx,.docx,.docm,.dot,.doc,.rtf,.txt,.xml,.sfdt"/>
    <button ej2-button (click)="onFileOpenClick()" >Import</button>
    <ejs-documenteditor #document_editor id="container" height="330px" style="display:block"></ejs-
    documenteditor>
  </div>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild('document_editor')
  public documentEditor: DocumentEditorComponent;
  public onFileOpenClick(): void {
    //Open file picker.
    document.getElementById('open_sfdt').click();
  }
  public onFileChange(e: any): void {
    if (e.target.files[0]) {
      //Get selected file.
      let file = e.target.files[0];
      if (file.name.substr(file.name.lastIndexOf('.') != '.sfdt') {
        this.loadFile(file);
      }
    }
  }
}
```

```
}  
public loadFile(file: File): void {  
    let ajax: XMLHttpRequest = new XMLHttpRequest();  
    ajax.open('POST', 'https://localhost:4000/api/documenteditor/Import', true);  
    ajax.onreadystatechange = () => {  
        if (ajax.readyState === 4) {  
            if (ajax.status === 200 || ajax.status === 304) {  
                // open SFDI text in document editor  
                this.documentEditor.open(ajax.responseText);  
            }  
        }  
    }  
    let formData: FormData = new FormData();  
    formData.append('files', file);  
    ajax.send(formData);  
}
```

Here's how to handle the server-side action for converting word document in to SFDI.

```
`csharp  
[AcceptVerbs("Post")]  
public string Import(Microsoft.AspNetCore.Http.IFormCollection data)  
{  
    if (data.Files.Count == 0)  
        return null;  
    System.IO.Stream stream = new System.IO.MemoryStream();  
    Microsoft.AspNetCore.Http.IFormFile file = data.Files[0];  
    int index = file.FileName.LastIndexOf('.');  
    string type = index > -1 && index < file.FileName.Length - 1 ?  
        file.FileName.Substring(index) : ".docx";  
    file.CopyTo(stream);  
    stream.Position = 0;
```

```
Syncfusion.EJ2.DocumentEditor.WordDocument document =
Syncfusion.EJ2.DocumentEditor.WordDocument.Load(stream, GetFormatType(type.ToLower()));
string sfdt = Newtonsoft.Json.JsonConvert.SerializeObject(document);
document.Dispose();
return sfdt;
}

internal static Syncfusion.EJ2.DocumentEditor.FormatType GetFormatType(string format)
{
if (string.IsNullOrEmpty(format))
throw new System.NotSupportedException("EJ2 DocumentEditor does not support this file format.");
switch (format.ToLower()) {
case ".dotx":
case ".docx":
case ".docm":
case ".dotm":
return Syncfusion.EJ2.DocumentEditor.FormatType.Docx;
case ".dot":
case ".doc":
return Syncfusion.EJ2.DocumentEditor.FormatType.Doc;
case ".rtf":
return Syncfusion.EJ2.DocumentEditor.FormatType.Rtf;
case ".txt":
return Syncfusion.EJ2.DocumentEditor.FormatType.Txt;
case ".xml":
return Syncfusion.EJ2.DocumentEditor.FormatType.WordML;
default:
throw new System.NotSupportedException("EJ2 DocumentEditor does not support this file format.");
}
}
`
```

To know about server-side action, please refer this [page](#).

Compatibility with Microsoft Word

Syncfusion Document Editor is a minimal viable Word document viewer/editor product for web applications. As most compatible Word editor, the product vision is adding valuable feature sets of

Microsoft Word, and not to cover 100% feature sets of Microsoft Word desktop application. You can even see the feature sets difference between Microsoft Word desktop and their Word online application. So kindly don't misunderstand this component as a complete replacement for Microsoft Word desktop application and expect 100% feature sets of it.

How Syncfusion accepts the feature request for Document Editor

Syncfusion accepts new feature request as valid based on feature value and technological feasibility, then plan to implement unsupported features incrementally in future releases in a phase-by-phase manner.

How to report the problems in Document Editor

You can report the problems with displaying, or editing Word documents in Document Editor component through [feedback portal](#). Kindly share the Word document for replicating the problem easily in minimal time. If you have confidential data, you can replace it and attach the document.

Why the document pagination differs from Microsoft Word

For your understanding about the Word document structure and the workflow of Word viewer/editor components, the Word document is a flow document in which content will not be preserved page by page; instead, the content will be preserved sequentially like a HTML file. Only the Word viewer/editor paginates the content of the Word document page by page dynamically, when opened for viewing or editing and this page-wise position information will not be preserved in the document level (it is Word file format specification standard). Syncfusion Document Editor component also does the same.

At present there is a known technical limitation related to slight difference in text size calculated using HTML element based text measuring approach. Even though the text size is calculated with correct font and font size values, the difference lies; it is as low as 0.00XX to 0. XXXX values compared to that of Microsoft Word application's display. Hence the document pagination (page-by-page display) can't be guaranteed for all the Word documents to match the pagination of Microsoft Word application.

How Syncfusion address the document pagination difference compared to Microsoft Word

The following table illustrates the reasons for pagination (page-by-page display) difference compared to Microsoft Word in your documents and how Syncfusion address it.

Root causes	How is it solved?
----- -----	-----
Any mistake (wrong behavior handled) in lay outing the supported elements and formatting	Customer can report to Syncfusion support and track the status through bug report link. Syncfusion fixes the bugs in next possible weekly patch release and service pack or main releases.
Font missing in deployment environment	Customer can either report to Syncfusion support and get suggestion or solve it on their own by installing the missing fonts in their deployment environment.
Any unsupported elements or formatting present in your document	Customer can report to Syncfusion support and track the status through feature request link. Syncfusion implements unsupported features incrementally in future releases based on feature importance, customer interest, efforts involved, and technological feasibility. Also, suggests alternate approach for possible cases.
Technical limitation related to framework	For example, there is a known case with slight fractional difference in text size measured using HTML and Microsoft Word's display. Customer can report to Syncfusion support and track the status through feature request link. Syncfusion does research about alternate approaches to overcome the technical limitation/behaviors and process it same as a feature.

>Note: Here the challenge is, time schedule for implementation varies based on the alternate solution and its reliability.|

See Also

- [Feature modules](#)
- [How to show and hide spinner while opening document in DocumentEditor](#)

Export in Angular Document editor component

Document Editor exports the document into various known file formats in client-side such as Microsoft Word document (.docx), text document (.txt), and its own format called **Syncfusion Document Text (.sfdt)**.

We are providing two types of save APIs as mentioned below.

API name	Purpose
save(filename,FormatType):void	FormatType: Sfdt or Docx or Txt Creates the document with specified file name and format type. Then, the created file is downloaded in the client browser by default.
saveAsBlob(FormatType):Blob	Creates the document in specified format type and returns the created document as Blob. This blob can be uploaded to your required server, database, or file path.

SFDT export

The following example shows how to export documents in document editor as Syncfusion document text (.sfdt).

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { DocumentEditorAllModule } from '@syncfusion/ej2-angular-documenteditor'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import {
    DocumentEditorComponent, EditorService, SelectionService,
    SfdtExportService, EditorHistoryService, BookmarkDialogService
} from '@syncfusion/ej2-angular-documenteditor';
@Component({
    imports: [
        ButtonModule,
        DocumentEditorAllModule
    ],
    standalone: true,
    selector: 'app-container',
    //specifies the template string for the Document Editor Container component
    template: `<div style="width:100%;"><button ej2-button
    (click)="saveAsSfdt()" >Save</button>
```

```

    <ejs-documenteditor #document_editor id="container" height="330px"
    style="display:block" [isReadOnly]=false [enableEditor]=true
    [enableSfdtExport]=true> </ejs-documenteditor></div>`,
    encapsulation: ViewEncapsulation.None,
    providers: [EditorService, SelectionService, SfdtExportService]
  })
  export class AppComponent {
    @ViewChild('document_editor')
    public documentEditor?: DocumentEditorComponent;
    public saveAsSfdt(): void {
      //Download the document as SFDT.
      (this.documentEditor as DocumentEditorComponent).save('sample',
      'Sfdt');
    }
  }
}

```

MAIN.TS

```

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

```

Word export

The following example shows how to export the document as Word document (.docx).

Note: The Syncfusion Document Editor component's document pagination (page-by-page display) can't be guaranteed for all the Word documents to match the pagination of Microsoft Word application. For more information about [why the document pagination \(page-by-page display\) differs from Microsoft Word](#)

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { DocumentEditorAllModule } from '@syncfusion/ej2-angular-documenteditor'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import {
  DocumentEditorComponent, EditorService, SelectionService,
  SfdtExportService, WordExportService
} from '@syncfusion/ej2-angular-documenteditor';
@Component({
  imports: [
    ButtonModule,
    DocumentEditorAllModule
  ],
  standalone: true,
  selector: 'app-container',
  //specifies the template string for the Document Editor component
  template: `<div style="width:100%"><button ej2-button
(click)="saveAsDocx()" >Save</button>

```



```

    <ejs-documenteditor #document_editor id="container" height="330px"
    style="display:block" [isReadOnly]=false [enableEditor]=true
    [enableWordExport]=true> </ejs-documenteditor></div>`,
    encapsulation: ViewEncapsulation.None,
    providers: [EditorService, SelectionService, SfdtExportService,
    WordExportService]
  })
  export class AppComponent {
    @ViewChild('document_editor')
    public documentEditor?: DocumentEditorComponent;
    public saveAsDocx(): void {
      //Export the document in docx format.
      (this.documentEditor as DocumentEditorComponent).save('sample',
      'Docx');
    }
  }
}

```

MAIN.TS

```

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

```

Template export

The following example shows how to export the document as Word Template (.dotx).

Note: The Syncfusion Document Editor component's document pagination (page-by-page display) can't be guaranteed for all the Word documents to match the pagination of Microsoft Word application. For more information about [why the document pagination (page-by-page display) differs from Microsoft Word] ([../document-editor/import/#why-the-document-pagination-differs-from-microsoft-word](#))

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { DocumentEditorAllModule } from '@syncfusion/ej2-angular-documenteditor'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import {
  DocumentEditorComponent, EditorService, SelectionService,
  SfdtExportService, WordExportService
} from '@syncfusion/ej2-angular-documenteditor';
@Component({
  imports: [
    ButtonModule,
    DocumentEditorAllModule
  ],
  standalone: true,
  selector: 'app-container',
  //specifies the template string for the Document Editor component
  template: `<div style="width:100%"><button ej2-button
  (click)="saveAsDotx()" >Save</button>

```

```

    <ejs-documenteditor #document_editor id="container" height="330px"
    style="display:block" [isReadOnly]=false [enableEditor]=true
    [enableWordExport]=true> </ejs-documenteditor></div>`,
    encapsulation: ViewEncapsulation.None,
    providers: [EditorService, SelectionService, SfdtExportService,
    WordExportService]
  })
  export class AppComponent {
    @ViewChild('document_editor')
    public documentEditor?: DocumentEditorComponent;
    public saveAsDotx(): void {
      //Export the document in docx format.
      (this.documentEditor as DocumentEditorComponent).save('sample',
      'Dotx');
    }
  }
}

```

MAIN.TS

```

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

```

Text export

The following example shows how to export document as text document (.txt).

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { DocumentEditorAllModule } from '@syncfusion/ej2-angular-
documenteditor'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import {
  DocumentEditorComponent, EditorService, SelectionService,
  SfdtExportService, TextExportService
} from '@syncfusion/ej2-angular-documenteditor';
@Component({
  imports: [

    ButtonModule,
    DocumentEditorAllModule
  ],
  standalone: true,
  selector: 'app-container',
  //specifies the template string for the Document Editor component
  template: `<div style="width:100%;"><button ej2-button
  (click)="saveAsTxt()" >Save</button>
  <ejs-documenteditor #document_editor id="container" height="330px"
  style="display:block" [isReadOnly]=false [enableEditor]=true
  [enableTextExport]=true> </ejs-documenteditor></div>`,
  encapsulation: ViewEncapsulation.None,
  providers: [EditorService, SelectionService, SfdtExportService,
  TextExportService]
})

```

```

    })
    export class AppComponent {
        @ViewChild('document_editor')
        public documentEditor?: DocumentEditorComponent;
        public saveAsTxt(): void {
            //Download the document as txt file.
            (this.documentEditor as DocumentEditorComponent).save('sample',
            'Txt');
        }
    }
}

```

MAIN.TS

```

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

```

Export as blob

Document Editor also supports API to store the document into a blob. Refer to the following sample to export document into blob in client-side.

`typescript

```

import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import {
    DocumentEditorComponent, EditorService, SelectionService, SfdtExportService, WordExportService
} from '@syncfusion/ej2-angular-documenteditor';
@Component({
    selector: 'app-container',
    //specifies the template string for the Document Editor component
    template: `<div><button ej-button (click)="saveAsBlob()" >Save</button>
    <ejs-documenteditor #document_editor id="container" height="330px" style="display:block"
    [isReadOnly]=false [enableEditor]=true [enableWordExport]=true [enableSfdtExport]=true> </ejs-
    documenteditor></div>`,
    encapsulation: ViewEncapsulation.None,
    providers: [EditorService, SelectionService, SfdtExportService, TextExportService]
})
export class AppComponent {
    @ViewChild('document_editor')
    public documentEditor: DocumentEditorComponent;
    public saveAsBlob(): void {
        //Export the document as Blob object.
    }
}

```

```
this.documentEditor.saveAsBlob('Docx').then((exportedDocument: Blob) => {  
  // The blob can be processed further  
});  
}  
}
```

For instance, to export the document as Rich Text Format file, implement an ASP.NET MVC web API controller using DocIO library by passing the DOCX blob. Refer to the following code example.

```
`csharp  
//API controller for the conversion.  
[HttpPost]  
public HttpResponseMessage ExportAsRtf()  
{  
  System.Web.HttpPostedFile data = HttpContext.Current.Request.Files[0];  
  //Opens document stream  
  WordDocument wordDocument = new WordDocument(data.InputStream);  
  MemoryStream stream = new MemoryStream();  
  //Converts document stream as RTF  
  wordDocument.Save(stream, FormatType.Rtf);  
  wordDocument.Close();  
  stream.Position = 0;  
  return new HttpResponseMessage() { Content = new StreamContent(stream) };  
}
```

In client-side, you can consume this web service and save the document as Rich Text Format (.rtf) file. Refer to the following example.

```
`typescript  
public saveAsBlob():void {  
  this.documentEditor.saveAsBlob('Docx').then((exportedDocument: Blob) => {  
    // The blob can be processed further  
    let formData: FormData = new FormData();  
    formData.append('fileName', 'sample.docx');  
    formData.append('data', exportedDocument);  
    saveAsRtf(formData);  
  });  
}
```

```
});  
}  
  
function saveAsRtf(formData: FormData): void {  
  let httpRequest: XMLHttpRequest = new XMLHttpRequest();  
  httpRequest.open('POST', '/api/DocumentEditor/ExportAsRtf', true);  
  httpRequest.onreadystatechange = () => {  
    if (httpRequest.readyState === 4) {  
      if (httpRequest.status === 200 || httpRequest.status === 304) {  
        if (!(!navigator.msSaveBlob)) {  
          navigator.msSaveBlob(httpRequest.response, 'sample.rtf');  
        } else {  
          let downloadLink: HTMLAnchorElement =  
            document.createElementNS('http://www.w3.org/1999/xhtml', 'a') as HTMLAnchorElement;  
          download('sample.rtf', 'rtf', httpRequest.response, downloadLink, 'download' in downloadLink);  
        }  
      } else {  
        console.error(httpRequest.response);  
      }  
    }  
  }  
  httpRequest.responseType = 'blob';  
  httpRequest.send(formData);  
}  
  
function download(fileName: string, extension: string, buffer: Blob, downloadLink:  
HTMLAnchorElement, hasDownloadAttribute: Boolean): void {  
  if (hasDownloadAttribute) {  
    downloadLink.download = fileName;  
    let dataUrl: string = window.URL.createObjectURL(buffer);  
    downloadLink.href = dataUrl;  
    let event: MouseEvent = document.createEvent('MouseEvent');  
    event.initEvent('click', true, true);  
    downloadLink.dispatchEvent(event);  
    setTimeout(() : void => {  
      window.URL.revokeObjectURL(dataUrl);  
    });  
  }  
}
```

```
dataUrl = undefined;
});
} else {
if (extension !== 'docx' && extension !== 'xlsx') {
let url: string = window.URL.createObjectURL(buffer);
let isPopupBlocked: Window = window.open(url, '_blank');
if (!isPopupBlocked) {
window.location.href = url;
}
}
}
}
}
```

See Also

- [Feature modules](#)

Web services in Angular Document editor component

You can deploy web APIs for server-side dependencies of Document Editor component in the following platforms.

- [ASP.NET Core](#)
- [ASP.NET MVC](#)
- [Java](#)

Which operations require server-side interaction

|Operations|When client-server communication will be triggered?|What type of data will be transferred between client and server?|

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

|[Open file formats other than SFDT](#)|When opening the document other than SFDT (Syncfusion Document Editor's native file format), the server-side web API is invoked from client-side script. |**Client:** Sends the input file.
Server: Receives the input file and sends the converted SFDT back to the client.

The server-side web API internally extracts the content from the document (DOCX, DOC, WordML, RTF, HTML) using Syncfusion Word library (DocIO) and converts it into SFDT for opening the document in Document Editor. |

|[Paste with formatting](#)|When pasting the formatted content (HTML/RTF) received from system clipboard. For converting HTML/RTF to SFDT format.

 Note: Whereas plain text received from system clipboard will be pasted directly in the client-side. |**Client:** Sends the input Html or Rtf string.
Server: Receives the input Html or Rtf string and sends the converted SFDT back to the client. |

| [Restrict editing](#) | When protecting the document, for generating hash. | **Client:** Sends the input data for hashing algorithm.
 Server: Receives the input data for hashing algorithm and sends the result hash information back to the client. |

| [Spellcheck](#) (default) | When the spellchecker is enabled on client-side Document Editor, and it performs the spell check validation for words in the document. | **Client:** Sends the words (string) with their language for spelling validation.
 Server: Receives the words (string) with their language for spelling validation and sends the validation result as JSON back to the client. |

| [SpellCheckByPage](#) | Document editor provides options to spellcheck page by page when loading the documents. By [enabling optimized spell check](#) in client-side, you can perform spellcheck page by page when loading the documents. | **Client:** Sends the words (string) with their language for spelling validation.
 Server: Receives the words (string) with their language for spelling validation and sends the validation result as JSON back to the client. |

| [Save as file formats other than SFDT and DOCX](#) (optional API) | You can configure this API, if you want to save the document in file format other than DOCX and SFDT.

 For saving the files as WordML, DOC, RTF, HTML, ODT, Text using Syncfusion Word library (DocIO) and PDF using Syncfusion Word (DocIO) and PDF libraries. | You can transfer document from client to server either as SFDT or DOCX format.

 First option (SFDT):
 Client: Sends the SFDT.
 Server: Receives the SFDT and saves the converted document as any file format supported by [Syncfusion Word library \(DocIO\)](#) in server or sends the saved file to the client browser.

 Second option (DOCX):
 Client: Sends the DOCX file.
 Server: Receives the DOCX file and saves the converted document as any file format supported by [Syncfusion Word library \(DocIO\)](#) in server or sends the saved file to the client browser. |

Note: If you don't require the above functionalities then you can deploy as pure client-side component without any server-side interactions.

Please refer the [example from GitHub](#) to configure the web service and set the [serviceUrl](#).

If your running web service Url is `http://localhost:62869/`, set the serviceUrl like below:

```
`typescript
this.container.serviceUrl = "http://localhost:62869/api/documenteditor/";
`
```

Required Web API structure

Please check below table for expected web API structure.

Expected method name	Parameters	Return type
----------------------	------------	-------------

-----	----	----
-------	------	------

Import	Files(IFormCollection)	json(sfdt format)
--------	------------------------	-------------------

SystemClipboard	CustomerParameter: content(type string either rtf or html) and type(either .rtf or .html)	json(sfdt format)
-----------------	-------------------------------------------------------------------------------------------	-------------------

RestrictEditing	Parameter of type CustomRestrictParameter	 public class CustomRestrictParameter
	{ 	public string passwordBase64 { get; set; }
	public string saltBase64 { get; set; } 	public int spinCount { get; set; } } result hash information

|SpellCheck(default) |Parameter: SpellCheckJsonData
public class SpellCheckJsonData
{
public int LanguageID { get; set; }
public string TexttoCheck { get; set; }
public bool CheckSpelling { get; set; }
public bool CheckSuggestion { get; set; }
public bool AddWord { get; set; }
} |Json type of Spellcheck containing details of spell checked word |

|SpellCheckByPage |Parameter: SpellCheckJsonData
public class SpellCheckJsonData
{
public int LanguageID { get; set; }
public string TexttoCheck { get; set; }
public bool CheckSpelling { get; set; }
public bool CheckSuggestion { get; set; }
public bool AddWord { get; set; }
} |Json type of Spellcheck containing details of spell checked word

Note: Document editor provides options to spellcheck page by page when loading the documents. By [enabling optimized spell check](#) in client-side, you can perform spellcheck page by page when loading the documents. |

|Save(optional API) |parameter: SaveParameter
public class SaveParameter
{
public string Content { get; set; }
public string FileName { get; set; }
} |void(Save the file as file stream) |

|ExportSFDT(optional API) |parameter: SaveParameter
public class SaveParameter
{
public string Content { get; set; }
public string FileName { get; set; }
} |FileStreamResult (to save the document in client-side) |

|Export(optional API) |Files(IFormCollection) |FileStreamResult (to save the document in client-side) |

Customize the expected method name

Document editor component provides an option to customize the expected method name for Import, SystemClipboard, RestrictEditing and SpellCheck using [serverActionSettings](#).

The following example code illustrates how to customize the method name using serverActionSettings.

```
`typescript
```

```
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import { ToolbarService, DocumentEditorContainerComponent } from '@syncfusion/ej2-angular-documenteditor';

@Component({
  selector: 'app-container',
  // specifies the template string for the DocumentEditorContainer component
  template: <ejs-documenteditorcontainer #document_editor [enableToolbar]=true [serverActionSettings]="settings"> </ejs-documenteditorcontainer>,
  providers: [ToolbarService]
})
export class AppComponent {
  @ViewChild('document_editor')
  public container: DocumentEditorContainerComponent;
  // Customize the API name
  public settings = { import: 'Import1', systemClipboard: 'SystemClipboard1', spellCheck: 'SpellCheck1', restrictEditing: 'RestrictEditing1' };
```



```
}
,
```

Add the custom headers to XMLHttpRequest

Document editor component provides an option to add custom headers of XMLHttpRequest using the [headers](#).

```
`typescript
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import { ToolbarService, DocumentEditorContainerComponent } from '@syncfusion/ej2-angular-
documenteditor';
@Component({
  selector: 'app-container',
  // specifies the template string for the DocumentEditorContainer component
  template: <ejs-documenteditorcontainer #document_editor [enableToolbar]=true
[headers]="customHeaders"> </ejs-documenteditorcontainer>,
  providers: [ToolbarService]
})
export class AppComponent {
  @ViewChild('document_editor')
  public container: DocumentEditorContainerComponent;
  // custom headers
  public customHeaders = [{ 'Authorization': 'Bearer YOURACCESSTOKEN' }, { 'Content-Type':
'application/json' }];
}
,
```

Modify the XMLHttpRequest before request send

Document editor component provides an option to modify the XMLHttpRequest object (setting additional headers, if needed) using [beforeXmlHttpRequestSend](#) event and it gets triggered before a server request.

You can customize the required [XMLHttpRequest](#) properties.

The following example code illustrates how to modify the XMLHttpRequest using [beforeXmlHttpRequestSend](#).

```
`typescript
import { DocumentEditorContainer, XmlHttpRequestEventArgs } from '@syncfusion/ej2-
documenteditor';
let container: DocumentEditorContainer = new DocumentEditorContainer({
  enableToolbar: true,
```

```
height: '590px',
});
// Below action, cancel all server-side interactions expect spell check
container.beforeXmlHttpRequestSend = (args: XmlHttpRequestEventArgs): void => {
//Here, modifying the request headers
args.headers = [{ syncfusion: 'true' }];
args.withCredentials = true;
switch (args.serverActionType) {
case 'Import':
case 'RestrictEditing':
case 'SystemClipboard':
args.cancel = true;
break;
}
};
container.appendTo('#container');
```

Note: Find the customizable serverActionType values are `'Import'` | `'RestrictEditing'` | `'SpellCheck'` | `'SystemClipboard'`.

Server Deployment

Word processor server docker image overview in Angular Document editor component

The Syncfusion **Word Processor (also known as Document Editor)** is a component with editing capabilities like Microsoft Word. It is used to create, edit, view, and print Word documents. It provides all the common word processing abilities, including editing text; formatting contents; resizing images and tables; finding and replacing text; importing, exporting, and printing Word documents; and using bookmarks and tables of contents.

This Docker image is the predefined Docker container of Syncfusion's Word Processor backend. You can deploy it quickly to your infrastructure.

Word Processor is a commercial product, and it requires a valid license to use it in a production environment ([request license or trial key](#)).

The Word Processor is supported in the JavaScript, Angular, React, Vue, ASP.NET Core, ASP.NET MVC, and Blazor platforms.

Prerequisites

Have [Docker](#) installed in your environment:

- On Windows, install [Docker for Windows](#).
- On macOS, install [Docker for Mac](#).

How to deploy Word Processor Docker image

Step 1: Pull the word-processor-server image from Docker Hub.

```
`console
docker pull syncfusion/word-processor-server
```

Step 2: Create the docker-compose.yml file with the following code in your file system.

```
`yaml
version: '3.4'

services:

word-processor-server:
image: syncfusion/word-processor-server:latest
environment:
Provide your license key for activation
SYNCFUSIONLICENSEKEY: YOURLICENSEKEY
ports:
```

- "6002:80"

Step 3: In a terminal tab, navigate to the directory where you've placed the docker-compose.yml file and execute the following.

```
`console
docker-compose up
```

Now the Word Processor server Docker instance runs in the localhost with the provided port number `http://localhost:6002`. Open this link in a browser and navigate to the Word Processor Web API control `http://localhost:6002/api/documenteditor`. It returns the default get method response.

Step 4: Append the Docker instance running the URL (`http://localhost:6002/api/documenteditor`) to the service URL in the client-side Word Processor control. For more information about how to get started with the Word Processor control, refer to this [getting started page](#).

How to configure spell checker dictionaries path in Docker compose file

Step 1: In the Docker compose file, mount the local directory as a container volume using the following code.

```
`yaml
version: '3.4'

services:

word-processor-server:
```

image: syncfusion/word-processor-server:latest

environment:

[Provide your license key for activation](#)

SYNCFUSIONLICENSEKEY: YOURLICENSEKEY

volumes:

- ./data:/app/data

ports:

- "6002:80"

,

This YAML definition binds the data folder that is available in the Docker compose file directory.

Step 2: In the data folder, include the dictionary files (.dic, .aff) and JSON file. The JSON file should contain the language based dictionary file configuration in the following format.

`yaml

[

{

"LanguageID": 1036,

"DictionaryPath": "fr_FR.dic",

"AffixPath": "fr_FR.aff",

"PersonalDictPath": "customDict.dic"

},

{

"LanguageID": 1033,

"DictionaryPath": "en_US.dic",

"AffixPath": "en_US.aff",

"PersonalDictPath": "customDict.dic"

}

]

,

Note: By default, the json file name should be "spellcheck.json". You can also use different file name by mounting the file name to 'SPELLCHECK/JSONFILENAME' attribute in Docker compose file as below,

`yaml

version: '3.4'

services:

word-processor-server:

image: syncfusion/word-processor-server:latest

environment:

[Provide your license key for activation](#)

SYNCFUSIONLICENSEKEY: YOURLICENSEKEY

SPELLCHECKDICTIONARYPATH: data

SPELLCHECKJSONFILENAME: spellcheck1.json

volumes:

- ./data:/app/data

ports:

- "6002:80"

,

Step 3: For handling the personal dictionary, place an empty .dic file (e.g., customDict.dic file) in the data folder.

Step 4: Provide the configured volume path to the environment variable like in the following in the Docker compose file.

`yaml

version: '3.4'

services:

word-processor-server:

image: syncfusion/word-processor-server:latest

environment:

[Provide your license key for activation](#)

SYNCFUSIONLICENSEKEY: YOURLICENSEKEY

SPELLCHECKDICTIONARYPATH: data

volumes:

- ./data:/app/data

ports:

- "6002:80"

,

How to copy template Word documents to Docker image

You can copy the required template Word documents into docker container while deploying the docker image to server. You can open these Word documents present in the server by passing the document path (name with relative path) to LoadDocument() web API.

Note: Place the word files in the data folder mentioned in the volumes section(i.e., C:/Docker/Data) of the docker-compose.yml file. All the files present in the folder path (C:/Docker/Data) mentioned in the volumes section of 'docker-compose.yml' file will be copied to the respective folder (/app/Data) of docker container. The Word documents copied to docker container can be processed using the 'LoadDocument' web API.

The following code example shows how to use LoadDocument() API in document editor.

```
`typescript
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import { ToolbarService } from '@syncfusion/ej2-angular-documenteditor';

@Component({
  selector: 'app-container',
  // specifies the template string for the DocumentEditorContainer component
  template: <ejs-documenteditorcontainer #document_editor height="600px"
style="display:block" (created)="onCreated()" [enableToolbar]=true> </ejs-
documenteditorcontainer>,
  providers: [ToolbarService]
})
export class AppComponent {
  @ViewChild('document_editor')
  public container: DocumentEditorContainerComponent;

  onCreated() {
    var dataContext = this;
    var uploadDocument = new FormData();
    uploadDocument.append('DocumentName', 'Getting Started.docx');
    var baseUrl = 'http://localhost:6002/api/documenteditor/LoadDocument';
    var httpRequest = new XMLHttpRequest();
    httpRequest.open('POST', baseUrl, true);
    httpRequest.onreadystatechange = function () {
      if (httpRequest.readyState === 4) {
        if (httpRequest.status === 200 || httpRequest.status === 304) {
          //Open the document in Document Editor.
          dataContext.container.documentEditor.open(httpRequest.responseText);
        }
      }
    };
    httpRequest.send(uploadDocument);
  }
}
```

```
}  
}  
};  
httpRequest.send(uploadDocument);  
}  
}  
`
```

Refer to these getting started pages to create a Word Processor in [Typescript](#), [React](#), [Vue](#), [ASP.NET MVC](#), [ASP.NET Core](#), and [Blazor](#).

How to deploy word processor server docker container in azure app service in Angular
Document editor component

Prerequisites

- Have [Azure account](#) and [Azure CLI](#) setup in your environment.
- Run the following command to open the Azure login page. Sign into your [Microsoft Azure account](#).

```
`  
az login  
`
```

Step 1: Create a resource group.

Create a resource group using the [az group create](#) command.

The following example creates a resource group named documenteditorresourcegroup in the eastus location.

```
`  
az group create --name documenteditorresourcegroup --location "East US"  
`
```

Step 2: Create an Azure App Service plan.

Create an App Service plan in the resource group with the [az appservice plan create](#) command.

The following example creates an App Service plan named documenteditorappservice in the Standard pricing tier (--sku S1) and in a Linux container (--is-linux).

```
`  
az appservice plan create --name documenteditorappservice --resource-group  
documenteditorresourcegroup --sku S1 --is-linux  
`
```

Step 3: Create a Docker Compose app.

Create a multi-container [web app](#) in the documenteditorappservice App Service plan with the [az webapp create](#) command. The following command creates the web app using the provided Docker compose file. Please look into the section for getting started with Docker compose to create the Docker compose file for the document editor server and use the created Docker compose file here.

```
az webapp create --resource-group documenteditorresourcegroup --plan documenteditorappservice --name documenteditor-server --multicontainer-config-type compose --multicontainer-config-file documenteditor-server-compose.yml
```

Step 4: Browse to the app.

Browse to the deployed app at `http://<app_name>.azurewebsites.net`, i.e. `http://documenteditor-server.azurewebsites.net`. Browse this link and navigate to the Document Editor Web API control `http://documenteditor-server.azurewebsites.net/api/documenteditor`. It returns the default get method response.

Append the app service running the URL `http://documenteditor-server.azurewebsites.net/api/documenteditor/` to the service URL in the client-side Document Editor control. For more information about the Document Editor control, refer to this [getting started page](#).

For more information about the app container service, please look deeper into the [Microsoft Azure Container Service](#) for a production-ready setup.

How to deploy word processor server docker container in azure kubernetes service in Angular Document editor component

Prerequisites

- Have [Azure account](#) and [Azure CLI](#) setup in your environment.
- Run the following command to open the Azure login page. Sign into your [Microsoft Azure account](#).

```
az login
```

Step 1: Create a resource group.

Create a resource group using the [az group create](#) command.

The following example creates a resource group named documenteditorresourcegroup in the eastus location.

```
az group create --name documenteditorresourcegroup --location "East US"
```

Step 2: Create AKS cluster.

Use the [az aks create](#) command to create an AKS cluster. The following example creates a cluster named documenteditorcluster with one node.

`

```
az aks create --resource-group documenteditorresourcegroup --name documenteditorcluster --node-count 1
```

`

Step 3: Connect to the cluster.

Install the [kubect!](#) into the workspace using the following command.

`

```
az aks install-cli
```

`

To configure kubectl to connect to your Kubernetes cluster, use the [az aks get-credentials](#) command. This command downloads credentials and configures the Kubernetes CLI to use them.

`

```
az aks get-credentials --resource-group documenteditorresourcegroup --name documenteditorcluster
```

`

Step 4: Create Kubernetes Services and Deployments

[Kubernetes Services](#) and [Deployments](#) can be configured in a file. To run the Document Editor server, you must define a Service and a Deployment documenteditorserver. To do this, create the documenteditor-server.yml file in the current directory using the following code.

```
`yaml
```

```
apiVersion: apps/v1
```

```
kind: Deployment
```

```
metadata:
```

```
labels:
```

```
app: documenteditorserver
```

```
name: documenteditorserver
```

```
spec:
```

```
replicas: 1
```

```
selector:
```

```
matchLabels:
```

```
app: documenteditorserver
```

```
strategy: {}
```

```
template:
```

```
metadata:
```

labels:

app: documenteditorserver

spec:

containers:

- image: syncfusion/word-processor-server:latest

name: documenteditorserver

ports:

- containerPort: 80

env:

- name: SYNCFUSIONLICENSEKEY

value: "YOURLICENSEKEY"

apiVersion: v1

kind: Service

metadata:

labels:

app: documenteditorserver

name: documenteditorserver

spec:

ports:

- port: 80

targetPort: 80

selector:

app: documenteditorserver

type: LoadBalancer

,

Step 5: To create all Services and Deployments needed to run the Document Editor server, execute the following.

`console

kubectl create -f ./documenteditor-server.yml

,

Run the following command to get the Kubernetes cluster deployed service details and copy the external IP address of the Document Editor service.

```
`console
```

```
kubectll get all
```

```
`
```

Browse the copied external IP address and navigate to the Document Editor Web API control `http://<external-ip>/api/documenteditor`. It returns the default get method response.

Step 6: Append the Kubernetes service running the URL `http://<external-ip>/api/documenteditor/` to the service URL in the client-side Document Editor control. For more information about the Document Editor control, refer to this [getting started page](#).

For more details about the Azure Kubernetes service, please look deeper into [Microsoft Azure Kubernetes Service](#) for a production-ready setup.

How to deploy Word Processor server in Amazon Kubernetes Service

Prerequisites

- **AWS Account** :Have Amazon account
- **AWS CLI**: Install the AWS Command Line Interface (CLI) on your local machine.
- **Kubectll** : Install the Kubernetes command-line tool kubectll on your local machine.
- **Docker**: Install Docker on your local machine.
- **Word Processor Docker Image**: Have a Docker image of the Word Processor server ready to deploy.

To deploy the Word Processor server docker image, need to follow the below process

- Push Docker Image to Registry (Amazon Elastic Registry)
- Deploy Docker Image on Amazon Kubernetes Service

Lets us discuss briefly about the each process

Push the Docker image to the Amazon Elastic Registry

Step 1: Dockerize the Word Processor Server Application with the image name [syncfusion/word-processor-server](#)

```
`
```

```
docker build -t <your-image-name>
```

```
`
```

Step 2: Create a private repository name 'documenteditor' in Amazon Elastic Container Registry (ECR) using the AWS CLI or AWS Console to push the docker image

```
`
```

```
aws ecr create-repository --repository-name <repository-name>
```

```
`
```

Step 3: Tag the image to push to ECR

docker tag <your-image-name>:latest <yourECRregistryURI>/<yourrepository_name>:latest

Refer to the following example to tag the image

docker tag syncfusion/word-processor-server:latest 123456.dkr.ecr.us-east-1.amazonaws.com/documenteditor:latest

Get the ECR registry URI from AWS console or using the below AWS CLI command

aws ecr describe-repositories --repository-names <repository-name> --query 'repositories[*].repositoryUri' --output text

Step 4: Login to the ECR registry using the AWS CLI to push the docker image

aws ecr get-login-password --region <region> | docker login --username AWS --password-stdin <yourECRregistry_URI>

Replace <region> with your AWS region and <yourECRregistry_URI> with your ECR registry URI.

Step 5: Push the tagged image to ECR

docker push <yourECRregistryURI>/<yourimage_name>:latest

[Deploy Docker Image on Amazon Kubernetes Service](#)

Follow the below steps to deploy the Docker image from the Amazon Elastic Registry(ECR) to Amazon Kubernetes Services (EKS)

Step 1: Create Amazon EKS cluster using the AWS Console

Step 2: Authenticate with AWS ECR

Need to get the authenticate with the AWS ECR to pull the image from the registry

aws ecr get-login-password --region <your-region> | docker login --username AWS --password-stdin <your-aws-account-id>.dkr.ecr.<your-region>.amazonaws.com

Step 3: : Configure Kubernetes to Communicate with the Cluster

```
aws eks --region <region> update-kubeconfig --name <cluster-name>
```

,

Step 4: Tag the Docker Image to the created cluster

,

```
docker tag <your-aws-account-id>.dkr.ecr.<your-region>.amazonaws.com/<repository-name>:<tag>  
<your-eks-cluster-id>.dkr.ecr.<your-region>.amazonaws.com/<repository-name>:<tag>
```

,

Refer to the following example

,

```
docker tag 12345.dkr.ecr.us-east-1.amazonaws.com/documenteditor:latest 98765ABCD.dkr.ecr.us-east-1.amazonaws.com/documenteditor:latest
```

,

In this command:

- should be replaced with your EKS cluster ID, which is the base part of your EKS cluster endpoint (e.g., abcd1234).
- should be replaced with your AWS region.

Step 5: To create Kubernetes deployment write Kubernetes manifest

i. Create a Kubernetes Deployment manifest (deployment.yaml) for your application. Specify the Docker image location.

,

```
apiVersion: apps/v1
```

```
kind: Deployment
```

```
metadata:
```

```
name: your-deployment
```

```
spec:
```

```
replicas: 3
```

```
selector:
```

```
matchLabels:
```

```
app: your-app
```

```
template:
```

```
metadata:
```

```
labels:
```

```
app: your-app
```

```
spec:
```

containers:

- name: your-container

image: <account-id>.dkr.ecr.<region>.amazonaws.com/your-repository-name:tag

ports:

- containerPort: 80

,

ii. Apply the Deployment manifest to create the deployment in your EKS cluster

,

kubectl apply -f deployment.yaml

,

iii. Use port forwarding to access the Word Processor Server application locally and verify its functionality

,

kubectl port-forward pod-name local-port:container-port

,

Get the pod names in AWS Console or using the below AWS CLI command

,

kubectl get pods

,

Finally you can get the sample in the localhost `http://<your-ip>/api/documenteditor/`

How to publish documenteditor web api application in azure app service from visual studio in Angular Document editor component

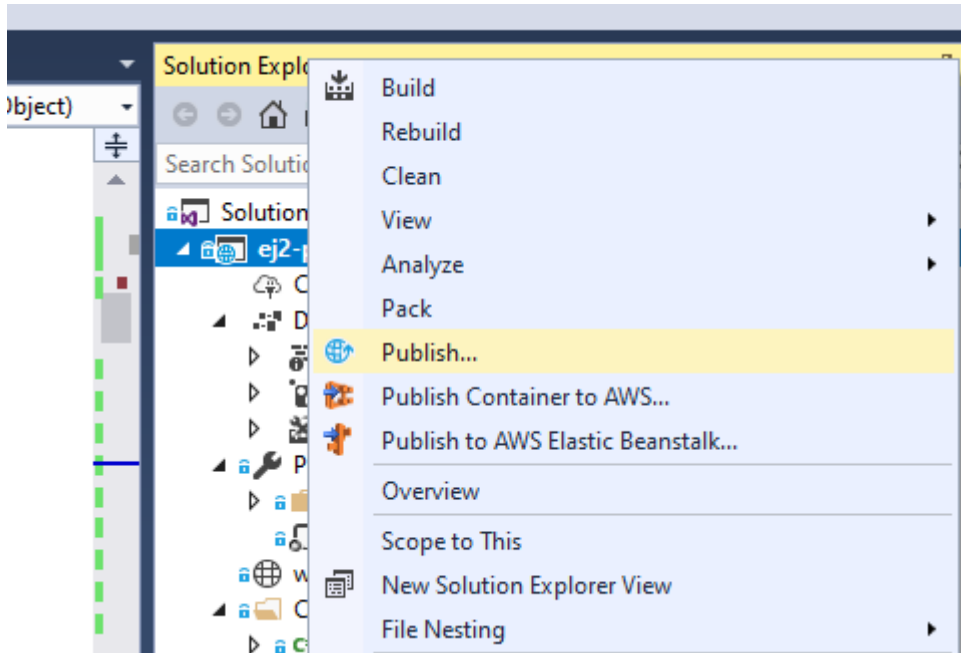
Prerequisites

- Visual Studio 2017 or 2019.
- An [Azure subscription](#).
- The Document Editor Web API controller application from [here](#).

Make sure you build the project using the Build > Build Solution menu command before following the deployment steps.

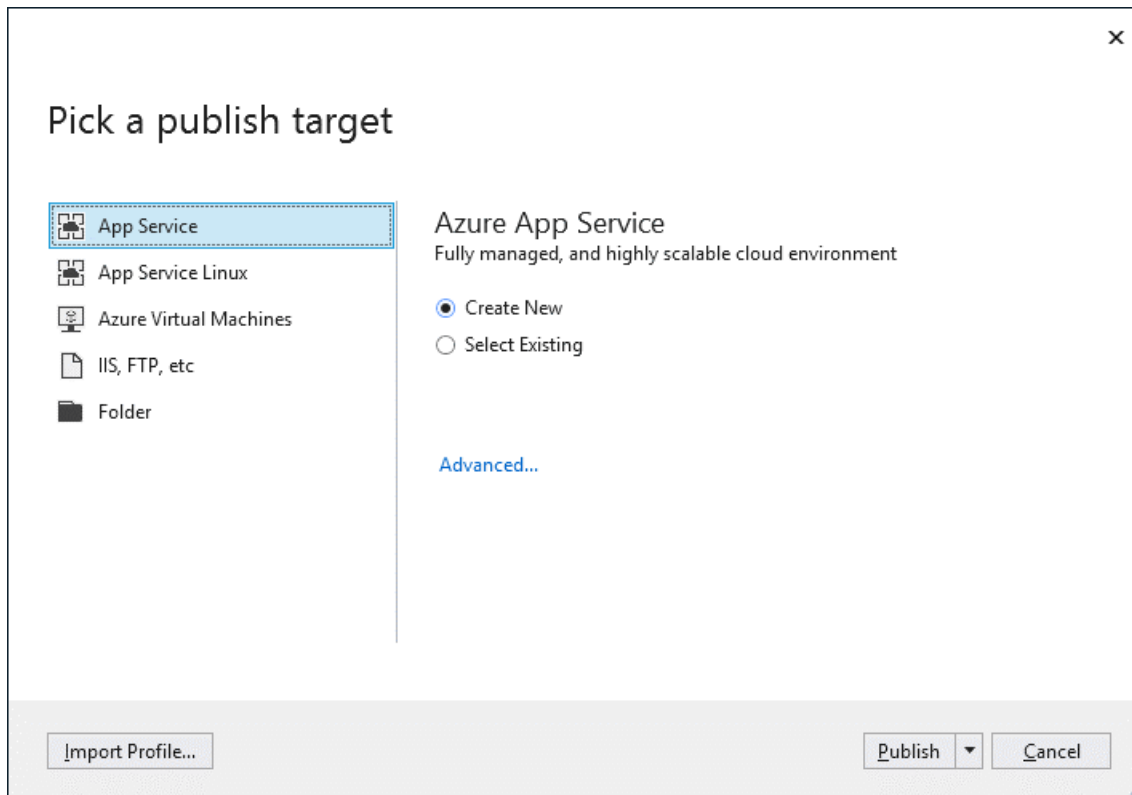
Publish to Azure App Service

Step 1: In Solution Explorer, right-click the project and click Publish (or use the Build > Publish menu item).



Step 2: If you have previously configured any publishing profiles, the Publish pane appears, in which case select Create new profile.

Step 3: In the Pick a publish target dialog box, select App Service.



Step 4: Select Publish. The Create App Service dialog box appears. Sign in with your Azure account, if necessary, and then the default app service settings populate the fields.

App Name

Subscription

Resource Group
 [New...](#)

Hosting Plan
 [New...](#)

Application Insights

Explore additional Azure services

[Create a SQL Database](#)

[Create a storage account](#)

Clicking the Create button will create the following Azure resources

Hosting Plan - ej2-documenteditor-server202005141...

App Service - ej2-documenteditor-server20200514102909

Export...

Create

Cancel

Step 5: Select Create. Visual Studio deploys the app to your Azure App Service, and the web app loads in your browser with the app name at `http://<app_name>.azurewebsites.net` (i.e. `http://ej2-documenteditor-server20200514102909.azurewebsites.net`).

Step 6: Navigate to Document Editor Web API control `http://ej2-documenteditor-server20200514102909.azurewebsites.net/api/documenteditor`. It returns the default get method response.

Append the app service running the URL `http://ej2-documenteditor-server20200514102909.azurewebsites.net/api/documenteditor` to the service URL in the client-side Document Editor control. For more information about how to get started with the Document Editor control, refer to this [getting started page](#).

For more information about the app container service, please look deeper into the [Microsoft Azure App Service](#) for a production-ready setup.

Accessibility in Angular Document editor component

The accessibility compliance for the Document editor component is outlined below.

| [Accessibility Criteria](#) | [Compatibility](#) |

| -- | -- |

| [WCAG 2.2 Support](#) | 

src="https://cdn.syncfusion.com/content/images/documentation/partial.png" alt="Intermediate"> |

| [Section 508 Support](#) | 

src="https://cdn.syncfusion.com/content/images/documentation/partial.png" alt="Intermediate"> |

| [Screen Reader Support](#) | |

| [Right-To-Left Support](#) | |

| [Color Contrast](#) | |

| [Mobile Device Support](#) | |

| [Keyboard Navigation Support](#) | |

| [Accessibility Checker Validation](#) | |

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

<style>

.post .post-content img {

display: inline-block;

margin: 0.5em 0;

}

</style>

<div> - All features of the component meet the requirement.</div>

<div> - Some features of the component do not meet the requirement.</div>

<div> - The component does not meet the requirement.</div>

Keyboard interaction

Document editor supports [keyboard shortcuts](#).

Ensuring accessibility

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

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

See also

- [Accessibility in Syncfusion Angular components](#)

Image in Angular Document editor component

Document Editor supports common raster format images like PNG, BMP, JPEG, SVG and GIF. You can insert an image file or online image in the document using the [insertImage\(\)](#) method. Refer to the following sample code.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { DocumentEditorAllModule } from '@syncfusion/ej2-angular-documenteditor'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import {
    DocumentEditorComponent, EditorService, SelectionService,
    SfdtExportService, EditorHistoryService, ImageResizerService
} from '@syncfusion/ej2-angular-documenteditor';
@Component({
    imports: [
        ButtonModule,
        DocumentEditorAllModule
    ],
    standalone: true,
    selector: 'app-container',
    //specifies the template string for the Document Editor component.
    template: `<div>
        <input id="insertImageButton" #insert_Image_Button
        style="position:fixed; left:-100em" type="file"
        (change)="onInsertImage($event)" accept=".jpeg,.jpg,.png,.gif,.bmp">
        <button ej2-button (click)="insertImageButtonClick()" >Insert
        Image</button>
        <ejs-documenteditor #document_editor id="container" height="330px"
        style="display:block" [enableSfdtExport]=true [enableWordExport]=true
        [enableSelection]=true [enableEditor]=true [isReadOnly]=false
        [enableImageResizer]=true> </ejs-documenteditor>
        </div>`,
    encapsulation: ViewEncapsulation.None,
    providers: [EditorService, SelectionService, SfdtExportService,
        ImageResizerService]
})
export class AppComponent {
    @ViewChild('document_editor')
    public documentEditor?: DocumentEditorComponent;
    public insertImageButtonClick(): void {
        let pictureUpload: HTMLInputElement =
        document.getElementById("insertImageButton") as HTMLInputElement;
        pictureUpload.value = '';
        //Open file picker.
        pictureUpload.click();
    }
    public onInsertImage(args: any): void {
        let documentEditor: DocumentEditorComponent = this.documentEditor as
        DocumentEditorComponent;
```

```

    if (navigator.userAgent.match('Chrome') ||
    navigator.userAgent.match('Firefox') || navigator.userAgent.match('Edge') ||
    navigator.userAgent.match('MSIE') || navigator.userAgent.match('.NET')) {
        if (args.target.files[0]) {
            //Get selected image.
            let path = args.target.files[0];
            let reader = new FileReader();
            reader.onload = function (frEvent: any) {
                let base64String = frEvent.target.result;
                let image = document.createElement('img');
                image.addEventListener('load', function () {
                    //Insert the image into Document Editor.
                    documentEditor.editor.insertImage(base64String,
this.width, this.height);
                })
                image.src = base64String;
            };
            //Convert the image in to base64 format.
            reader.readAsDataURL(path);
        }
        //Safari does not Support FileReader Class
    } else {
        let image = document.createElement('img');
        image.addEventListener('load', function () {
            //Insert the image into Document Editor.
            documentEditor.editor.insertImage(args.target.value);
        })
        image.src = args.target.value;
    }
}
}

```

MAIN.TS

```

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

```

Image files will be internally converted to base64 string. Whereas, online images are preserved as URL.

Note: EMF and WMF images can't be inserted, but these types of images will be preserved in Document Editor when using ASP.NET MVC Web API.

Image resizing

Document Editor provides built-in image resizer that can be injected into your application based on the requirements. This allows you to resize the image by dragging the resizing points using mouse or touch interactions. This resizer appears as follows.



Changing size

Document Editor expose API to get or set the size of the selected image. Refer to the following sample code.

```
`typescript
```

```
this.documentEditor.selection.imageFormat.width = 800;
```

```
this.documentEditor.selection.imageFormat.height = 800;
```

```
,
```

Note: Images are stored and processed(read/write) as base64 string in DocumentEditor. The online image URL is preserved as a URL in Document Editor upon saving.

Text wrapping style

Text wrapping refers to how images fit with surrounding text in a document. Please [refer to this page](#) for more information about text wrapping styles available in Word documents.

Positioning the image

DocumentEditor preserves the position properties of the image and displays the image based on position properties. It does not support modifying the position properties. Whereas the image will be automatically moved along with text edited if it is positioned relative to the line or paragraph.

See Also

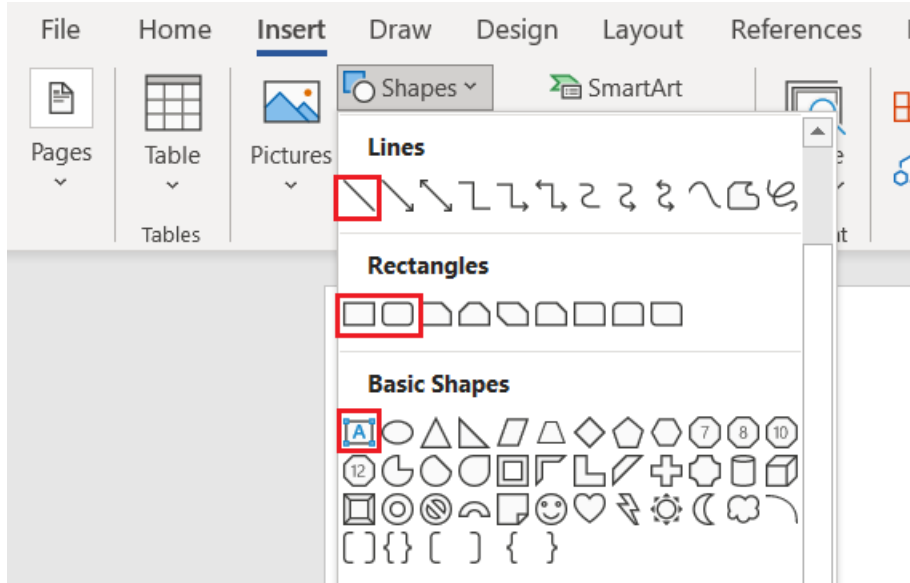
- [Feature modules](#)

Shapes in Angular Document editor component

Shapes are drawing objects that include a text box, rectangles, lines, curves, circles, etc. It can be preset or custom geometry. At present, Document Editor does not have support to insert shapes. however, if the document contains a shape while importing, it will be preserved properly.

Supported shapes

The Document Editor has preservation support for Text box, Rectangle, Rounded Rectangle and Line shapes.

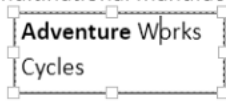


Note: When using ASP.NET MVC service, the unsupported shapes will be converted as image and preserved as image.

Text box Shape

A text box is a rectangular area on the document where you can enter text. When you click in a text box, a flashing cursor will display indicating that you can begin typing. It allows you to enter multiple lines of text with all text formatting.

Adventure Works Cycles, the fictitious company on which the Adventure Works sample databases are based, is a large, multinational manufacturing company. The company manufactures and sells metal and composite bicycles to North American, European and Asian commercial markets. While its base operation is located in Bothell, Washington with 290 employees, several regional sales teams are located throughout their market base.



Shape Resizer

The Document Editor also supports a built-in shape resizer to resize the shapes present in the document. The shape resizer accepts both touch and mouse interactions.



Text wrapping style

Text wrapping refers to how shapes fit with surrounding text in a document. Please [refer to this page](#) for more information about text wrapping styles available in Word documents.

Positioning the shape

DocumentEditor preserves the position properties of the shape and displays the shape based on position properties. It does not support modifying the position properties. Whereas the shape will be automatically moved along with text edited if it is positioned relative to the line or paragraph.

Text wrapping style in Angular Document editor component

Text wrapping refers to how images and shapes are fit with surrounding text in a document. Currently, Document Editor has only preservation support for image and textbox shape with below wrapping styles.

In-Line with Text

In this option, the image or shape is placed on the same line surrounding with text like any other word or letter. This image or shape will be automatically moved along with the text while editing, whereas the other options denote that the image or shape stays in a fixed position while the text shifts and wraps around it.

Adventure Works Cycles, the fictitious company on which the AdventureWorks sample databases are based, is a large, multinational manufacturing company. The company



manufactures and sells metal and composite bicycles to North American, European and Asian commercial markets. While its base operation is located in

In Front of Text

In this option, the image or shape is placed in front of the text. This can be used to place an image around some text or to add shape to highlight the part in a paragraph.

Adventure Works Cycles, the fictitious company on which the AdventureWorks sample databases are based, is a large manufacturing company. The company manufactures and sells metal and composite bicycles to North American, European and Asian commercial markets. While its base operation is located in Bothell, Washington with 290 employees, several regional sales centers are located and shipped in throughout their market base.



Note: Starting from v18.2.0.x, the in front of wrapping styles are supported.

Top and Bottom

In this option, Text wraps above and below the image or shape. No text is to the left or right of the image or shape. This can be used for larger images or shapes that occupy most of the width in a document.

Note: Starting from v19.1.0.x, the top and bottom wrapping style is supported.

Adventure Works Cycles, the fictitious company on which the AdventureWorks sample databases are based, is a large, multinational manufacturing company. The company



manufactures and sells metal and composite bicycles to North American, European and Asian commercial markets. While its base operation is located in Bothell, Washington with 290

Behind

In this option, the image or shape is placed behind the text. This can be used when you need to add a watermark or background image to a document.

Adventure Works Cycles, the fictitious company on which the AdventureWorks sample databases are based, is a large, multinational manufacturing company. The company manufactures and sells metal and composite bicycles to North American, European and Asian commercial markets. While its base operation is located in Bothell, Washington with 290 employees, several regional sales teams and manufacturer and located and shipped in throughout their market base.

Note: Starting from v19.2.0.x, behind text wrapping styles are supported.

Square

In this option, Text wraps around the image or text box in a square shape.

Note: Tight and Through styles will be preserved as square wrapping style in Document Editor which is supported from v19.2.0.x.

Adventure Works Cycles, the fictitious company on which the Adventure Works sample databases are based, is a large, multinational manufacturing company. The company manufactures and sells metal and composite bicycles to North American, European and Asian commercial markets. While its base operation is located in Bothell, Washington with 290 employees, several regional sales teams are located throughout their market base.

Adventure works cycles
company.

Bookmark in Angular Document editor component

Bookmark is a powerful tool that helps you to mark a place in the document to find again easily. You can enter many bookmarks in the document and give each one a unique name to identify easily.

Document Editor provides built-in dialog to add, delete, and navigate bookmarks within the document. To add a bookmark, select a portion of text in the document. After that, jump to the location or add links to it within the document using built-in hyperlink dialog. You can also delete bookmarks from a document.

Bookmark names need to begin with a letter. They can include both numbers and letters, but not spaces. To separate the words, use an underscore.

Bookmark names starting with an underscore are called hidden bookmarks. For example, bookmarks generated for table of contents.

Add bookmark

Using [insertBookmark](#) method, Bookmark can be added to the selected text.

```
`csharp
```

```
this.container.documentEditor.editor.insertBookmark("Bookmark1");
```

```
,
```

Select Bookmark

You can select the bookmark in the document using [selectBookmark](#) method by providing Bookmark name to select as shown in the following code snippet.

```
`csharp
```

```
container.documentEditor.selection.selectBookmark("Bookmark1", true);
```

```
,
```

Note: Second parameter is optional parameter and it denotes is exclude bookmark start and end from selection. If true, excludes bookmark start and end from selection.

Delete Bookmark

You can delete bookmark in the document using [deleteBookmark](#) method as shown in the following code snippet.

```
`csharp
this.container.documentEditor.editor.deleteBookmark("Bookmark1");
`
```

Get Bookmark from document

You can get all the bookmarks in the document using [getBookmarks](#) method as shown in the following code snippet.

```
`csharp
this.container.documentEditor.selection.getBookmarks(false);
`
```

Note: Parameter denotes is include hidden bookmarks. If false, ignore hidden bookmark.

Get Bookmark from selection

You can get bookmarks in current selection in the document using [getBookmarks](#) method as shown in the following code snippet.

```
`csharp
this.container.documentEditor.selection.getBookmarks(false);
`
```

Replace bookmark content

You can replace bookmark content without removing the bookmark start and end for backtracking the bookmark content.

```
`csharp
this.container.documentEditor.selection.selectBookmark("Bookmark1", true);
this.container.documentEditor.editor.insertText('Hello World')
`
```

You can replace content by removing the bookmark start and end, thus the bookmark content can't be tracked in future.

```
`csharp
this.container.documentEditor.selection.selectBookmark("Bookmark1");
this.container.documentEditor.editor.insertText('Hello World')
`
```


Show or Hide bookmark

You can show or hide the show square brackets around bookmarked items in Document editor component.

The following example code illustrates how to show or hide square brackets around bookmarked items.

```
`typescript
```

```
this.container.documentEditorSettings.showBookmarks = true;
```

```
,
```

Bookmark Dialog

The following example shows how to open bookmark dialog in document editor.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { DocumentEditorAllModule } from '@syncfusion/ej2-angular-documenteditor'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import {
    DocumentEditorComponent, EditorService, SelectionService,
    SfdtExportService, EditorHistoryService, BookmarkDialogService
} from '@syncfusion/ej2-angular-documenteditor';
@Component({
    imports: [

        ButtonModule,
        DocumentEditorAllModule
    ],
    standalone: true,
    selector: 'app-container',
    // specifies the template string for the Document Editor component
    template: `<div><button ej2-button (click)="showBookmarkDialog()" >
>Dialog</button>
    <ejs-documenteditor #document_editor id="container" height="330px"
    style="display:block" [isReadOnly]=false [enableEditor]=true
    [enableEditorHistory]=true [enableBookmarkDialog]=true>
    </ejs-documenteditor></div>`,
    encapsulation: ViewEncapsulation.None,
    providers: [EditorService, SelectionService, SfdtExportService,
    EditorHistoryService, BookmarkDialogService]
})
export class AppComponent {
    @ViewChild('document_editor')
    public documentEditor?: DocumentEditorComponent;
    public showBookmarkDialog(): void {
        //Open the bookmark dialog.
        (this.documentEditor as
        DocumentEditorComponent).showDialog('Bookmark');
    }
}
```

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

- [Feature modules](#)
- [Bookmark dialog](#)

Link in Angular Document editor component

Document Editor supports hyperlink field. You can link a part of the document content to Internet or file location, mail address, or any text within the document.

Navigate a hyperlink

Document Editor triggers `requestNavigate` event whenever user clicks Ctrl key or tap a hyperlink within the document. This event provides necessary details about link type, navigation URL, and local URL (if any) as arguments, and allows you to easily customize the hyperlink navigation functionality.

Add the requestNavigate event for DocumentEditor

The following example illustrates how to add requestNavigate event for DocumentEditor.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { DocumentEditorAllModule } from '@syncfusion/ej2-angular-documenteditor'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import {
    DocumentEditorComponent, SfdtExportService, SelectionService,
    RequestNavigateEventArgs
} from '@syncfusion/ej2-angular-documenteditor';
@Component({
    imports: [
        ButtonModule,
        DocumentEditorAllModule
    ],
    standalone: true,
    selector: 'app-container',
    //specifies the template string for the Document Editor component
    template: `<div>
        <ejs-documenteditor #document_editor height="330px"
        style="display:block" [isReadOnly]=false [enableSelection]=true
        [enableSfdtExport]=true [enableEditor]=true
        (requestNavigate)="onRequestNavigate($event)">
        </ejs-documenteditor>
        </div>`,
    encapsulation: ViewEncapsulation.None,
    providers: [SfdtExportService, SelectionService]
})
export class AppComponent {
```

```

    @ViewChild('document_editor')
    public documentEditor?: DocumentEditorComponent;
    public onRequestNavigate(args: RequestNavigateEventArgs): void {
        if (args.linkType !== 'Bookmark') {
            let link: string = args.navigationLink;
            if (args.localReference.length > 0) {
                link += '#' + args.localReference;
            }
            //Navigate to the specified URL.
            window.open(link);
            args.isHandled = 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 the requestNavigate event for DocumentEditorContainer component

The following example illustrates how to add requestNavigate event for DocumentEditorContainer component.

`typescript

```

import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import {
    DocumentEditorContainerComponent, ToolbarService
} from '@syncfusion/ej2-angular-documenteditor';
@Component({
    selector: 'app-container',
    // specifies the template string for the Document Editor container component
    template: `<div><button ej2-button (click)="insertText()" >Insert Text</button>
    <ejs-documenteditorcontainer #documenteditor_default
    serviceUrl="https://ej2services.syncfusion.com/production/web-services/api/documenteditor/"
    height="600px" style="display:block" [enableToolbar]=true (created)="onCreated()"> </ejs-
    documenteditorcontainer></div>`,
    encapsulation: ViewEncapsulation.None,
    providers: [ToolbarService]
})
export class AppComponent {
    @ViewChild('documenteditor_default')

```

```
public container: DocumentEditorContainerComponent;

public onCreate(): void {
  this.container.documentEditor.requestNavigate = (args) => {
    if (args.linkType !== 'Bookmark') {
      let link: string = args.navigationLink;
      if (args.localReference.length > 0) {
        link += '#' + args.localReference;
      }
      //Navigate to the specified URL.
      window.open(link);
      args.isHandled = true;
    }
  };
}
```

If the selection is in hyperlink, trigger this event by calling `navigateHyperlink` method of `Selection` instance. Refer to the following example.

```
`typescript
this.documentEditor.selection.navigateHyperlink();
`
```

Copy link

Document Editor copies link text of a hyperlink field to the clipboard if the selection is in hyperlink. Refer to the following example.

```
`typescript
this.documentEditor.selection.copyHyperlink();
`
```

Add hyperlink

To create a basic hyperlink in the document, press `ENTER` / `SPACEBAR` / `SHIFT + ENTER` / `TAB` key after typing the address, for instance <http://www.google.com>. Document Editor automatically converts this address to a hyperlink field. The text can be considered as a valid URL if it starts with any of the following.

```
`http://`<br>
```

```
`https://`<br>
```

```
file:///<br>
```

www.

mailto:

Refer to the following example.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { DocumentEditorAllModule } from '@syncfusion/ej2-angular-documenteditor'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import {
    DocumentEditorComponent, SfdtExportService, SelectionService,
    EditorService, RequestNavigateEventArgs
} from '@syncfusion/ej2-angular-documenteditor';
@Component({
    imports: [
        ButtonModule,
        DocumentEditorAllModule
    ],
    standalone: true,
    selector: 'app-container',
    //specifies the template string for the Document Editor component
    template: `<div>
        <ejs-documenteditor #document_editor height="330px"
        style="display:block" [isReadOnly]=false [enableSelection]=true
        [enableEditor]=true (requestNavigate)="onRequestNavigate($event)">
        </ejs-documenteditor>
        </div>`,
    encapsulation: ViewEncapsulation.None,
    providers: [SfdtExportService, SelectionService, EditorService]
})
export class AppComponent {
    @ViewChild('document_editor')
    public documentEditor?: DocumentEditorComponent;
    public onRequestNavigate(args: RequestNavigateEventArgs): void {
        if (args.linkType !== 'Bookmark') {
            let link: string = args.navigationLink;
            if (args.localReference.length > 0) {
                link += '#' + args.localReference;
            }
            window.open(link);
            args.isHandled = true;
        }
    }
}
```

MAIN.TS

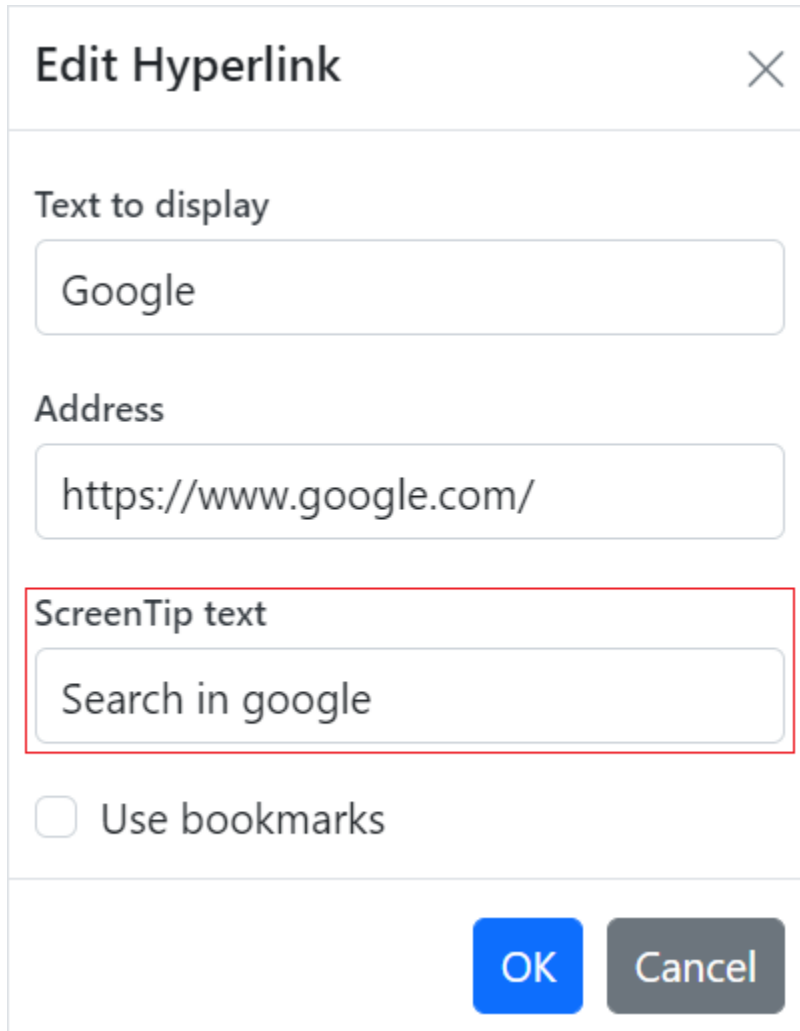
```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Customize screen tip

You can customize the screen tip text for the hyperlink by using below sample code.

```
`typescript
this.documentEditor.insertHyperlink('https://www.google.com', 'Google', '<<Screen tip text>>');
`
```

Screen tip text can be modified through UI by using the [Hyperlink dialog](#)



Edit Hyperlink [X]

Text to display

Google

Address

https://www.google.com/

ScreenTip text

Search in google

☐ Use bookmarks

OK Cancel

Remove hyperlink

To remove link from hyperlink in the document, press Backspace key at the end of a hyperlink. By removing the link, it will be converted as plain text. You can use 'removeHyperlink' method of 'Editor' instance if the selection is in hyperlink. Refer to the following example.

```
`typescript
this.documentEditor.editor.removeHyperlink();
`
```

Hyperlink dialog

Document Editor provides dialog support to insert or edit a hyperlink. Refer to the following example.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { DocumentEditorAllModule } from '@syncfusion/ej2-angular-documenteditor'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import {
    DocumentEditorComponent, EditorService, SelectionService,
    EditorHistoryService, HyperlinkDialogService, SfdtExportService
} from '@syncfusion/ej2-angular-documenteditor';
@Component({
    imports: [
        ButtonModule,
        DocumentEditorAllModule
    ],
    standalone: true,
    selector: 'app-container',
    //specifies the template string for the Document Editor component
    template: `<div><button ej2-button (click)="showHyperlinkDialog()" >Show
Dialog</button>
    <ejs-documenteditor #document_editor id="container" height="330px"
style="display:block" [isReadOnly]=false [enableEditor]=true
[enableSelection]=true [enableSfdtExport]=true [enableEditorHistory]=true
[enableHyperlinkDialog]=true> </ejs-documenteditor></div>`,
    encapsulation: ViewEncapsulation.None,
    providers: [EditorService, SelectionService, EditorHistoryService,
HyperlinkDialogService, SfdtExportService]
})
export class AppComponent {
    @ViewChild('document_editor')
    public documentEditor?: DocumentEditorComponent;
    public showHyperlinkDialog(): void {
        //Open hyperlink dialog.
        (this.documentEditor as
DocumentEditorComponent).showDialog('Hyperlink');
    }
}
```

MAIN.TS

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

You can use the following keyboard shortcut to open the hyperlink dialog if the selection is in hyperlink.

Key Combination	Description
----- -----	

|Ctrl + K | Open hyperlink dialog that allows you to create or edit hyperlink|

See Also

- [Feature modules](#)
- [Hyperlink dialog](#)

Table in Angular Document editor component

Tables are an efficient way to present information. Document Editor can display and edit the tables. You can select and edit tables through keyboard, mouse, or touch interactions. Document Editor exposes a rich set of APIs to perform these operations programmatically.

Create a table

You can create and insert a table at cursor position by specifying the required number of rows and columns.

Refer to the following sample code.

```
`typescript
this.documentEditor.editor.insertTable(3,3);
`
```

The maximum size of row and column is limited to 32767 and 63 respectively.

Insert rows

You can add a row (or several rows) above or below the row at cursor position by using the [insertRow](#) method. This method accepts the following parameters:

Parameter | Type | Description

left(optional) | boolean| This is optional and if omitted, it takes the value as false and inserts to the right of column at cursor position.

count(optional) | number | This is optional and if omitted, it takes the value as 1.

Refer to the following sample code.

```
`typescript
//Insert a column to the right of the column at cursor position.
this.documentEditor.editor.insertColumn();
//Insert a column to the left of the column at cursor position.
this.documentEditor.editor.insertColumn(false);
//Insert two columns to the left of the column at cursor position.
this.documentEditor.editor.insertColumn(false, 2);
`
```

Select an entire table

If the cursor position is inside a table, you can select the entire table by using the following sample code.

```
`typescript
```



```
this.documentEditor.selection.selectTable();
```

```
,
```

Select row

You can select the entire row at cursor position by using the following sample code.

```
`typescript
```

```
this.documentEditor.selection.selectRow();
```

```
,
```

If current selection spans across cells of different rows, all these rows will be selected.

Select column

You can select the entire column at cursor position by using the following sample code.

```
`typescript
```

```
this.documentEditor.selection.selectColumn();
```

```
,
```

If current selection spans across cells of different columns, all these columns will be selected.

Select cell

You can select the cell at cursor position by using the following sample code.

```
`typescript
```

```
this.documentEditor.selection.selectCell();
```

```
,
```

Delete table

Document Editor allows you to delete the entire table. You can use the [deleteTable\(\)](#) method of editor instance, if selection is in table. Refer to the following sample code.

```
`typescript
```

```
this.documentEditor.editor.deleteTable();
```

```
,
```

Delete row

Document Editor allows you to delete the selected number of rows. You can use the [deleteRow\(\)](#) method of editor instance to delete the selected number of rows, if selection is in table. Refer to the following sample code.

```
`typescript
```

```
this.documentEditor.editor.deleteRow();
```

```
,
```

Delete column

Document Editor allows you to delete the selected number of columns. You can use the [deleteColumn\(\)](#) method of editor instance to delete the selected number of columns, if selection is in table. Refer to the following sample code.

```
`typescript
```

```
this.documentEditor.editor.deleteColumn();
```

```
,
```

Merge cells

You can merge cells vertically, horizontally, or combination of both to a single cell. To vertically merge the cells, the columns within selection should be even in left and right directions. To horizontally merge the cells, the rows within selection should be even in top and bottom direction.

Refer to the following sample code.

```
`typescript
```

```
this.documentEditor.editor.mergeCells()
```

```
,
```

Positioning the table

Document Editor preserves the position properties of the table and displays the table based on position properties. It does not support modifying the position properties. Whereas the table will be automatically moved along with text edited if it is positioned relative to the paragraph.

How to work with tables

The following sample demonstrates how to delete the table row or columns, merge cells and how to bind the API with button.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { DocumentEditorAllModule } from '@syncfusion/ej2-angular-documenteditor'
import { ToolbarModule } from '@syncfusion/ej2-angular-navigations'
import { ComboBoxModule } from '@syncfusion/ej2-angular-dropdowns'
import { ColorPickerModule } from '@syncfusion/ej2-angular-inputs'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import {
    DocumentEditorComponent, EditorService, SelectionService,
    SfdtExportService, EditorHistoryService, TableDialogService,
    ContextMenuService
} from '@syncfusion/ej2-angular-documenteditor';
@Component({
    imports: [
        ButtonModule,
        ToolbarModule,
        DocumentEditorAllModule,
        ComboBoxModule,
        ColorPickerModule
    ],
    standalone: true,
    selector: 'app-container',
    styleUrls: ['./style.css'],
    template: `<div>
<div>
```

```

        <ejs-toolbar (clicked)='toolbarButtonClick($event) '>
            <e-items>
                <e-item prefixIcon="e-de-ctnr-table e-icons"
tooltipText="Insert Table" id="table"></e-item>
                <e-item type="Separator"></e-item>
                <e-item prefixIcon="e-de-ctnr-insertabove e-icons"
tooltipText="Insert new row above" id="insert_above"></e-item>
                <e-item prefixIcon="e-de-ctnr-insertbelow e-icons"
tooltipText="Insert new row below" id="insert_below"></e-item>
                <e-item type="Separator"></e-item>
                <e-item prefixIcon="e-de-ctnr-insertleft e-icons"
tooltipText="Insert new column to the left" id="insert_left"></e-item>
                <e-item prefixIcon="e-de-ctnr-insertright e-icons"
tooltipText="Insert new column to the right" id="insert_right"></e-item>
                <e-item type="Separator"></e-item>
                <e-item prefixIcon="e-de-delete-table e-icons"
tooltipText="Delete Entire table" id="delete_table"></e-item>
                <e-item prefixIcon="e-de-ctnr-deleterows e-icons"
tooltipText="Delete the selected row" id="delete_row"></e-item>
                <e-item prefixIcon="e-de-ctnr-deletecolumns e-icons"
tooltipText="Delete the selected column" id="delete_column"></e-item>
                <e-item type="Separator"></e-item>
                <e-item prefixIcon="e-de-ctnr-mergecell e-icons"
tooltipText="Merge the selected cells" id="merge_cell"></e-item>
                <e-item type="Separator"></e-item>
                <e-item text="Dialog" tooltipText="Open insert table
dialog" id="table_dialog"></e-item>
            </e-items>
        </ejs-toolbar>
    </div>
    <ejs-documenteditor #document_editor [isReadOnly]=false
[enableSelection]=true [enableEditorHistory]=true [enableEditor]=true
[enableTableDialog]=true [enableContextMenu]=true [enableSfdtExport]=true
height="330px" style="display:block" (created)="onCreated()"></ejs-
documenteditor>
</div>`,
    encapsulation: ViewEncapsulation.None,
    providers: [EditorService, SelectionService, SfdtExportService,
EditorHistoryService, TableDialogService, ContextMenuService]
})
export class AppComponent {
    @ViewChild('document_editor')
    public documentEditor?: DocumentEditorComponent;
    onCreated(): void {
        (this.documentEditor as
DocumentEditorComponent).editor.insertTable(2, 2);
    }
    public toolbarButtonClick(arg: any) {
        switch (arg.item.id) {
            case 'table':
                //Insert table API to add table
                (this.documentEditor as
DocumentEditorComponent).editor.insertTable(3, 2);
                break;
            case 'insert_above':
                //Insert the specified number of rows to the table above to
the row at cursor position

```

```

        (this.documentEditor as
DocumentEditorComponent).editor.insertRow(true, 2);
        break;
        case 'insert_below':
            //Insert the specified number of rows to the table below to
the row at cursor position
            (this.documentEditor as
DocumentEditorComponent).editor.insertRow();
            break;
        case 'insert_left':
            //Insert the specified number of columns to the table left
to the column at cursor position
            (this.documentEditor as
DocumentEditorComponent).editor.insertColumn(true, 2);
            break;
        case 'insert_right':
            //Insert the specified number of columns to the table right
to the column at cursor position
            (this.documentEditor as
DocumentEditorComponent).editor.insertColumn();
            break;
        case 'delete_table':
            //Delete the entire table
            (this.documentEditor as
DocumentEditorComponent).editor.deleteTable();
            break;
        case 'delete_row':
            //Delete the selected number of rows
            (this.documentEditor as
DocumentEditorComponent).editor.deleteRow();
            break;
        case 'delete_column':
            //Delete the selected number of columns
            (this.documentEditor as
DocumentEditorComponent).editor.deleteColumn();
            break;
        case 'merge_cell':
            //Merge the selected cells into one (both vertically and
horizontally)
            (this.documentEditor as
DocumentEditorComponent).editor.mergeCells();
            break;
        case 'table_dialog':
            //Opens insert table dialog
            (this.documentEditor as
DocumentEditorComponent).showDialog('Table');
            break;
    }
}
}

```

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

- [Feature modules](#)
- [Insert table dialog](#)

Table of contents in Angular Document editor component

The table of contents in a document is same as the list of chapters at the beginning of a book. It lists each heading in the document and the page number, where that heading starts with various options to customize the appearance.

Inserting table of contents

Document Editor exposes an API to insert table of contents at cursor position programmatically. You can specify the settings for table of contents explicitly. Otherwise, the default settings will be applied.

[TableOfContentsSettings](#) contain the following properties:

- **startLevel:** Specifies the start level for constructing table of contents.
- **endLevel:** Specifies the end level for constructing table of contents.
- **includeHyperlink:** Specifies whether the link for headings is included.
- **includePageNumber:** Specified whether the page number of the headings is included.
- **rightAlign:** Specifies whether the page number is right aligned.
- **tabLeader:** Specifies the tab leader styles such as none, dot, hyphen, and underscore.
- **includeOutlineLevels:** Specifies whether the outline levels are included.

The following code illustrates how to insert table of content in document editor.

```
`typescript
```

```
let tocSettings: TableOfContentsSettings =
```

```
{
```

```
startLevel: 1, endLevel: 3, includeHyperlink: true, includePageNumber: true, rightAlign: true
```

```
};
```

```
this.documentEditor.editor.insertTableOfContents(tocSettings);
```

```
,
```

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { DocumentEditorAllModule } from '@syncfusion/ej2-angular-documenteditor'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import {
    DocumentEditorComponent, EditorService, SelectionService
} from '@syncfusion/ej2-angular-documenteditor';
@Component({
    imports: [
```

```

        ButtonModule,
        DocumentEditorAllModule
    ],
    standalone: true,
    selector: 'app-container',
    template: `<div>
        <ejs-documenteditor #document_editor [isReadOnly]=true
        (created)="onCreated()" [enableSelection]=true [enableEditor]=true
        height="330px" style="display:block"></ejs-documenteditor>
    </div>`,
    encapsulation: ViewEncapsulation.None,
    providers: [EditorService, SelectionService]
})
export class AppComponent {
    @ViewChild('document_editor')
    public documentEditor?: DocumentEditorComponent;
    onCreated(): void {
        if ((this.documentEditor as
DocumentEditorComponent).isDocumentLoaded) {
            let sfdt: string =
'{"sections":[{"blocks":[{"paragraphFormat":{"styleName":"Heading
1"},"inlines":[{"text":"Headin"}, {"name": "_GoBack", "bookmarkType": 0}, {"name":
": _GoBack", "bookmarkType": 1}, {"text": "g1"}]}], {"paragraphFormat":{"styleName":
": "Heading
2"}, {"inlines": [{"text": "Heading2"}]}], {"paragraphFormat":{"styleName": "Headin
g
3"}, {"inlines": [{"text": "Heading3"}]}], {"paragraphFormat":{"styleName": "Headin
g
4"}, {"inlines": [{"text": "Heading4"}]}], {"paragraphFormat":{"styleName": "Headin
g
5"}, {"inlines": [{"text": "Heading5"}]}], {"paragraphFormat":{"styleName": "Headin
g
6"}, {"inlines": [{"text": "Heading6"}]}], {"paragraphFormat":{"styleName": "Normal
"}, {"inlines": [{"text": "Normal"}]}], "headersFooters": {}, "sectionFormat": {"hea
derDistance": 36.0, "footerDistance": 36.0, "pageWidth": 612.0, "pageHeight": 792.0
, "leftMargin": 72.0, "rightMargin": 72.0, "topMargin": 72.0, "bottomMargin": 72.0, "
differentFirstPage": false, "differentOddAndEvenPages": false}], "characterForm
at": {"fontSize": 11.0, "fontFamily": "Calibri"}, "paragraphFormat": {"afterSpacin
g": 8.0, "lineSpacing": 1.0791666507720947, "lineSpacingType": "Multiple"}, "backg
round": {"color": "#FFFFFF"}, "styles": [{"type": "Paragraph", "name": "Normal", "
next": "Normal"}, {"type": "Paragraph", "name": "Heading
1", "basedOn": "Normal", "next": "Normal", "link": "Heading 1
Char", "characterFormat": {"fontSize": 16.0, "fontFamily": "Calibri
Light", "fontColor": "#2F5496FF"}, "paragraphFormat": {"beforeSpacing": 12.0, "aft
erSpacing": 0.0, "outlineLevel": "Level1"}}, {"type": "Paragraph", "name": "Heading
2", "basedOn": "Normal", "next": "Normal", "link": "Heading 2
Char", "characterFormat": {"fontSize": 13.0, "fontFamily": "Calibri
Light", "fontColor": "#2F5496FF"}, "paragraphFormat": {"beforeSpacing": 2.0, "afte
rSpacing": 0.0, "outlineLevel": "Level2"}}, {"type": "Paragraph", "name": "Heading
3", "basedOn": "Normal", "next": "Normal", "link": "Heading 3
Char", "characterFormat": {"fontSize": 12.0, "fontFamily": "Calibri
Light", "fontColor": "#1F3763FF"}, "paragraphFormat": {"beforeSpacing": 2.0, "afte
rSpacing": 0.0, "outlineLevel": "Level3"}}, {"type": "Paragraph", "name": "Heading
4", "basedOn": "Normal", "next": "Normal", "link": "Heading 4
Char", "characterFormat": {"italic": true, "fontFamily": "Calibri
Light", "fontColor": "#2F5496FF"}, "paragraphFormat": {"beforeSpacing": 2.0, "afte

```

```

rSpacing":0.0,"outlineLevel":"Level4"}},{ "type":"Paragraph","name":"Heading
5","basedOn":"Normal","next":"Normal","link":"Heading 5
Char","characterFormat":{"fontFamily":"Calibri
Light","fontColor":"#2F5496FF"},"paragraphFormat":{"beforeSpacing":2.0,"afte
rSpacing":0.0,"outlineLevel":"Level5"}},{ "type":"Paragraph","name":"Heading
6","basedOn":"Normal","next":"Normal","link":"Heading 6
Char","characterFormat":{"fontFamily":"Calibri
Light","fontColor":"#1F3763FF"},"paragraphFormat":{"beforeSpacing":2.0,"afte
rSpacing":0.0,"outlineLevel":"Level6"}},{ "type":"Character","name":"Default
Paragraph Font"}},{ "type":"Character","name":"Heading 1
Char","basedOn":"Default Paragraph
Font","characterFormat":{"fontSize":16.0,"fontFamily":"Calibri
Light","fontColor":"#2F5496FF"}},{ "type":"Character","name":"Heading 2
Char","basedOn":"Default Paragraph
Font","characterFormat":{"fontSize":13.0,"fontFamily":"Calibri
Light","fontColor":"#2F5496FF"}},{ "type":"Character","name":"Heading 3
Char","basedOn":"Default Paragraph
Font","characterFormat":{"fontSize":12.0,"fontFamily":"Calibri
Light","fontColor":"#1F3763FF"}},{ "type":"Character","name":"Heading 4
Char","basedOn":"Default Paragraph
Font","characterFormat":{"italic":true,"fontFamily":"Calibri
Light","fontColor":"#2F5496FF"}},{ "type":"Character","name":"Heading 5
Char","basedOn":"Default Paragraph
Font","characterFormat":{"fontFamily":"Calibri
Light","fontColor":"#2F5496FF"}},{ "type":"Character","name":"Heading 6
Char","basedOn":"Default Paragraph
Font","characterFormat":{"fontFamily":"Calibri
Light","fontColor":"#1F3763FF"}]]}';
        //Open the document in Document Editor.
        (this.documentEditor as DocumentEditorComponent).open(sfdd);
    }
}
}

```

MAIN.TS

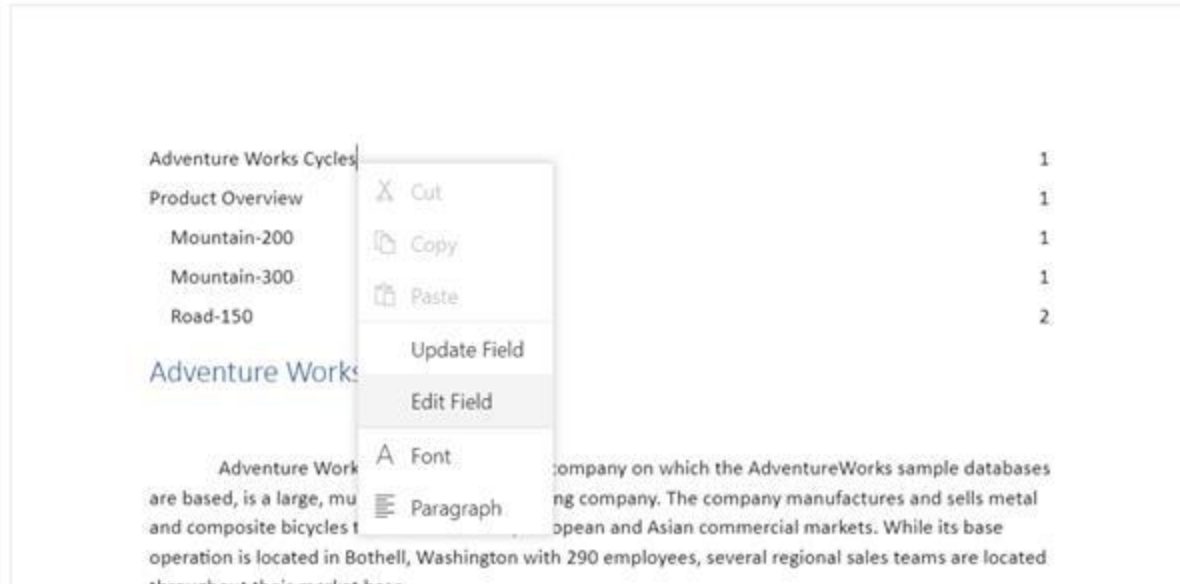
```

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

```

Update or edit table of contents

You can update or edit the table of contents using the built-in context menu shown up by right-clicking it. Refer to the following screenshot.



- **Update Field:** Updates the headings in table of contents with same settings by searching the entire document.
- **Edit Field:** Opens the built-in table of contents dialog and allows you to modify its settings.

You can also do it programmatically by using the exposed API. Refer to the following sample code.

`typescript

```
let tocSettings: TableOfContentsSettings =
{
  startLevel: 1, endLevel: 3, includeHyperlink: true, includePageNumber: true, rightAlign: true
};
this.documentEditor.editor.insertTableOfContents(tocSettings);
`
```

Same method is used for inserting, updating, and editing table of contents. This will work based on the current element at cursor position and the optional settings parameter. If table of contents is present at cursor position, the update operation will be done based on the optional settings parameter. Otherwise, the insert operation will be done.

See Also

- [Table of contents dialog](#)

Header footer in Angular Document editor component

Document Editor supports headers and footers in its document. Each section in the document can have the following types of headers and footers:

- **First page:** Used only on the first page of the section.
- **Even pages:** Used on all even numbered pages in the section.

- Default: Used on all pages of the section, where first or even pages are not applicable or not specified.

You can define this by setting format properties of the corresponding section using the following sample code.

```
`typescript
//Defines whether different header footer is required for first page of the section
this.documentEditor.selection.sectionFormat.differentFirstPage = true;
//Defines whether different header footer is required for odd and even pages in the section
this.documentEditor.selection.sectionFormat.differentOddAndEvenPages = true;
`
```

[Go to header footer region](#)

Double click in header or footer region to move the selection into it. You can also do this by using the following code.

```
`typescript
this.documentEditor.selection.goToHeader();
`
```

```
`typescript
this.documentEditor.selection.goToFooter();
`
```

[Link to previous](#)

Link to previous is enabled by default when document has more than one section. If you're using different headers and footers such as different first page or different odd and even pages, they can't be linked together because they're all separate.

Before setting or getting the link to previous value, use the ['goToHeader'](#) or ['goToFooter'](#) API to move the current selection to the header or footer region.

You can get or set the default header footer link to previous value of a section at cursor position by using the following sample code.

```
`typescript
this.container.documentEditor.selection.sectionFormat.oddPageHeader.linkToPrevious = false;
this.container.documentEditor.selection.sectionFormat.oddPageFooter.linkToPrevious = false;
`
```

In case the document has different header and footer types, such as different first page, odd, and even pages.

```
`typescript
// Different first page
this.container.documentEditor.selection.sectionFormat.firstPageHeader.linkToPrevious = false;
```

```
this.container.documentEditor.selection.sectionFormat.firstPageFooter.linkToPrevious = false;
//Even page
this.container.documentEditor.selection.sectionFormat.firstPageHeader.linkToPrevious = false;
this.container.documentEditor.selection.sectionFormat.firstPageFooter.linkToPrevious = false;
`
```

Note: When there is more than one section in the document, the Link to Previous option becomes available. By default, this feature is disabled state in UI and set to return false for the first section.

Header and footer distance

You can define the distance of header region content from the top of the page. Refer to the following sample code.

```
`typescript
this.documentEditor.selection.sectionFormat.headerDistance = 36;
`
```

Same way, you can define the distance of footer region content from the bottom of the page. Refer to the following sample code.

```
`typescript
this.documentEditor.selection.sectionFormat.footerDistance = 36;
`
```

Close header footer region

Move the selection to the document body from header or footer region by double clicking or tapping the document area. You can also perform this by using the following sample code.

```
`typescript
this.documentEditor.selection.closeHeaderFooter()
`
```

See Also

- [Working with Section Formatting](#)

Text format in Angular Document editor component

Document Editor supports several formatting options for text like bold, italic, font color, highlight color, and more. This section describes how to modify the formatting for selected text in detail.

Bold

The bold formatting for selected text can be get or set by using the following sample code.

```
`typescript
//Gets the value for bold formatting of selected text.
let bold : boolean = documenteditor.selection.characterFormat.bold;
//Sets bold formatting for selected text.
```

```
documenteditor.selection.characterFormat.bold = true;
```

```
,
```

You can toggle the bold formatting based on existing value at selection. Refer to the following sample code.

```
`typescript
```

```
documenteditor.editor.toggleBold();
```

```
,
```

Italic

The Italic formatting for selected text can be get or set by using the following sample code.

```
`typescript
```

```
//Gets the value for italic formatting of selected text.
```

```
let italic : boolean = documenteditor.selection.characterFormat.italic;
```

```
//Sets italic formatting for selected text.
```

```
documenteditor.selection.characterFormat.italic= true|false;
```

```
,
```

You can toggle the Italic formatting based on existing value at selection. Refer to the following sample code.

```
`typescript
```

```
documenteditor.editor.toggleItalic();
```

```
,
```

Underline property

The underline style for selected text can be get or set by using the following sample code.

```
`typescript
```

```
//Gets the value for underline formatting of selected text.
```

```
let underline : Underline = documenteditor.selection.characterFormat.underline;
```

```
//Sets underline formatting for selected text.
```

```
documenteditor.selection.characterFormat.underline='Single' | 'None';
```

```
,
```

You can toggle the underline style of selected text based on existing value at selection by specifying a value. Refer to the following sample code.

```
`typescript
```

```
documenteditor.editor.toggleUnderline('Single');
```

```
,
```

Strikethrough property

The strikethrough style for selected text can be get or set by using the following sample code.

```
`typescript
```

```
//Gets the value for strikethrough formatting of selected text.
```

```
let strikethrough : Strikethrough = documenteditor.selection.characterFormat.strikethrough;
```

```
//Sets strikethrough formatting for selected text.
```

```
documenteditor.selection.characterFormat.strikethrough='Single' | 'Normal';
```

```
,
```

You can toggle the strikethrough style of selected text based on existing value at selection by specifying a value. Refer to the following sample code.

```
`typescript
```

```
documenteditor.editor.toggleStrikethrough();
```

```
,
```

Superscript property

The selected text can be made superscript by using the following sample code.

```
`typescript
```

```
//Gets the value for baselineAlignment formatting of selected text.
```

```
let baselineAlignment : BaselineAlignment =  
documenteditor.selection.characterFormat.baselineAlignment;
```

```
//Sets baselineAlignment formatting for selected text.
```

```
documenteditor.selection.characterFormat.baselineAlignment='Superscript';
```

```
,
```

Toggle the selected text as superscript or normal using the following sample code.

```
`typescript
```

```
documenteditor.editor.toggleSuperscript();
```

```
,
```

Subscript property

The selected text can be made subscript by using the following sample code.

```
`typescript
```

```
//Gets the value for baselineAlignment formatting of selected text.
```

```
let baselineAlignment : BaselineAlignment =  
documenteditor.selection.characterFormat.baselineAlignment;
```

```
//Sets baselineAlignment formatting for selected text.
```

```
documenteditor.selection.characterFormat.baselineAlignment='Subscript';
```

```
,
```

Toggle the selected text as subscript or normal using the following sample code.

```
`typescript
```

```
documenteditor.editor.toggleSubscript();  
、
```

You can make a subscript or superscript text as normal using the following code.

```
`typescript  
documenteditor.selection.characterFormat.baselineAlignment='Normal';  
、
```

Size

The size of selected text can be get or set using the following code.

```
`typescript  
//Gets the value for fontSize formatting of selected text.  
let fontSize : number = documenteditor.selection.characterFormat.fontSize;  
//Sets fontSize formatting for selected text.  
documenteditor.selection.characterFormat.fontSize= 32;  
、
```

Color

The color of selected text can be get or set using the following code.

```
`typescript  
//Gets the value for fontColor formatting of selected text.  
let fontColor : string = documenteditor.selection.characterFormat.fontColor;  
//Sets fontColor formatting for selected text.  
documenteditor.selection.characterFormat.fontColor= 'Pink';  
documenteditor.selection.characterFormat.fontColor= '#FFC0CB';  
、
```

Font

The font style of selected text can be get or set using the following sample code.

```
`typescript  
//Gets the value for fontFamily formatting of selected text.  
let baselineAlignment : string = documenteditor.selection.characterFormat.fontFamily;  
//Sets fontFamily formatting for selected text.  
documenteditor.selection.characterFormat.fontFamily= 'Arial';  
、
```

Highlight color

The highlight color of the selected text can be get or set using the following sample code.

```
`typescript
```

```
//Gets the value for highlightColor formatting of selected text.
let highlightColor : HighlightColor = documenteditor.selection.characterFormat.highlightColor;
//Sets highlightColor formatting for selected text.
documenteditor.selection.characterFormat.highlightColor= 'Pink';
documenteditor.selection.characterFormat.highlightColor= '#FFC0CB';
`
```

Toolbar with options for text formatting

Refer to the following example.

```
`typescript
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import {
  DocumentEditorComponent, EditorService, SelectionService, EditorHistoryService, SfdtExportService,
  ContextMenuService
} from '@syncfusion/ej2-angular-documenteditor';
@Component({
  selector: 'app-container',
  styleUrls: ['styles.css'],
  template: `<div>
<div>
<ejs-toolbar (clicked)='toolbarClickHandler($event)'>
<e-items>
<e-item prefixIcon="e-de-ctnr-bold e-icons" tooltipText="Bold" id="bold"></e-item>
<e-item prefixIcon="e-de-ctnr-italic e-icons" tooltipText="Italic" id="italic"></e-item>
<e-item prefixIcon="e-de-ctnr-underline e-icons" tooltipText="Underline" id="underline"></e-item>
<e-item prefixIcon="e-de-ctnr-strikethrough e-icons" tooltipText="Strikethrough"
id="strikethrough"></e-item>
<e-item prefixIcon="e-de-ctnr-subscript e-icons" tooltipText="Subscript" id="subscript"></e-item>
<e-item prefixIcon="e-de-ctnr-superscript e-icons" tooltipText="Superscript" id="superscript"></e-item>
<e-item type="Seperator"></e-item>
<e-item type="Input">
<ng-template #template>
<ejs-colorpicker [value]='#000000' [showButtons]=true (change)='onFontColorChange()' ></ejs-
colorpicker>
</ng-template>
```

```

</e-item>
<e-item type="Seperator"></e-item>
<e-item type="Input">
<ng-template #template>
<ejs-combobox [dataSource]='fontStyle' [width]='120' [index]='2' [allowCustom]=true
(change)='onFontFamilyChange()' [showClearButton]=false></ejs-combobox>
</ng-template>
</e-item>
<e-item type="Input">
<ng-template #template>
<ejs-combobox [dataSource]='fontSize' [width]='80' [index]='2' [allowCustom]='true'
(change)='onFontSizeChange()' [showClearButton]='false'></ejs-combobox>
</ng-template>
</e-item>
</e-items>
</ejs-toolbar>
</div>

<ejs-documenteditor #document_editor (selectionChange)='onSelectionChange()'
[enableSelection]='true' [isReadOnly]='false' [enableEditor]=true [enableEditorHistory]=true
[enableSfdtExport]=true [enableContextMenu]=true height="330px" style="display:block"></ejs-
documenteditor>

</div>`,
encapsulation: ViewEncapsulation.None,
providers: [EditorService, SelectionService, EditorHistoryService, SfdtExportService,
ContextMenuService]
})
export class AppComponent {
@ViewChild('document_editor')
public documentEditor: DocumentEditorComponent;

public fontStyle: string[] = ['Algerian', 'Arial', 'Calibri', 'Cambria', 'Cambria Math', 'Candara', 'Courier
New', 'Georgia', 'Impact', 'Segoe Print', 'Segoe Script', 'Segoe UI', 'Symbol', 'Times New Roman',
'Verdana', 'Windings'];

public fontSize: string[] = ['8', '9', '10', '11', '12', '14', '16', '18', '20', '22', '24', '26', '28', '36', '48', '72', '96'];

public toolbarButtonClick(arg: any) {
switch (arg.item.id) {
case 'bold':

```

```
//Toggles the bold of selected content
this.documentEditor.editor.toggleBold();
break;
case 'italic':
//Toggles the Italic of selected content
this.documentEditor.editor.toggleItalic();
break;
case 'underline':
//Toggles the underline of selected content
this.documentEditor.editor.toggleUnderline('Single');
break;
case 'strikethrough':
//Toggles the strikethrough of selected content
this.documentEditor.editor.toggleStrikethrough();
break;
case 'subscript':
//Toggles the subscript of selected content
this.documentEditor.editor.toggleSubscript();
break;
case 'superscript':
//Toggles the superscript of selected content
this.documentEditor.editor.toggleSuperscript();
break;
}
}
public onFontFamilyChange(args) {
this.documentEditor.ej2Instances.selection.characterFormat.fontFamily = args.value;
this.documentEditor.focusIn();
}
public onFontSizeChange(args) {
this.documentEditor.ej2Instances.selection.characterFormat.fontSize = args.value;
this.documentEditor.focusIn();
}
```



```

public onFontColorChange(args) {
this.documentEditor.ej2Instances.selection.characterFormat.fontColor = args.currentValue.hex;
this.documentEditor.focusIn();
}

public onSelectionChange() {
var characterformat = this.documentEditor.ej2Instances.selection.characterFormat;
var properties = [characterformat.bold, characterformat.italic, characterformat.underline,
characterformat.strikeThrough];
var toggleBtnId = ["bold", "italic", "underline", "strikethrough"];
for (var i = 0; i < properties.length; i++) {
let toggleBtn: HTMLElement = document.getElementById(toggleBtnId[i]);
if ((typeof (properties[i]) == 'boolean' && properties[i] == true) || (typeof (properties[i]) == 'string' &&
properties[i] != 'None'))
toggleBtn.classList.add("e-btn-toggle");
else {
if (toggleBtn.classList.contains("e-btn-toggle"))
toggleBtn.classList.remove("e-btn-toggle");
}
}
}
}
`

```

See Also

- [Feature modules](#)
- [Font dialog](#)
- [Keyboard shortcuts](#)

Paragraph format in Angular Document editor component

Document Editor supports various paragraph formatting options such as text alignment, indentation, paragraph spacing, and more.

Indentation

You can modify the left or right indentation of selected paragraphs using the following sample code.

```

`typescript
this.documentEditor.selection.paragraphFormat.leftIndent= 24;
this.documentEditor.selection.paragraphFormat.rightIndent= 24;

```

`

Special indentation

You can define special indent for first line of the paragraph using the following sample code.

`typescript

```
this.documentEditor.selection.paragraphFormat.firstLineIndent= 24;
```

`

Increase indent

You can increase the left indent of selected paragraphs by a factor of 36 points using the following sample code.

`typescript

```
this.documentEditor.editor.increaseIndent()
```

`

Decrease indent

You can decrease the left indent of selected paragraphs by a factor of 36 points using the following sample code.

`typescript

```
this.documentEditor.editor.decreaseIndent()
```

`

Text alignment

You can get or set the text alignment of selected paragraphs using the following sample code.

`typescript

```
this.documentEditor.selection.paragraphFormat.textAlignment= 'Center' | 'Left' | 'Right' | 'Justify';
```

`

You can toggle the text alignment of selected paragraphs by specifying a value using the following sample code.

`typescript

```
this.documentEditor.editor.toggleTextAlignment('Center' | 'Left' | 'Right' | 'Justify');
```

`

Line spacing and its type

You can define the line spacing and its type for selected paragraphs using the following sample code.

`typescript

```
this.documentEditor.selection.paragraphFormat.lineSpacingType='AtLeast';
```

```
this.documentEditor.selection.paragraphFormat.lineSpacing= 6;
```

`

Paragraph spacing

You can define the spacing before or after the paragraph by using the following sample code.

```
`typescript
```

```
this.documentEditor.selection.paragraphFormat.beforeSpacing= 24;
```

```
this.documentEditor.selection.paragraphFormat.afterSpacing= 24;
```

```
,
```

You can also set automatic spacing before and after the paragraph by using the following sample code.

```
`typescript
```

```
this.documentEditor.selection.paragraphFormat.spaceBeforeAuto = true;
```

```
this.documentEditor.selection.paragraphFormat.spaceAfterAuto = true;
```

```
,
```

Note: If auto spacing property is enabled, then value defined in the `beforeSpacing` and `afterSpacing` property will not be considered.

Pagination properties

You can enable or disable the following pagination properties for the paragraphs in a Word document.

- Widow/Orphan control - whether the first and last lines of the paragraph are to remain on the same page as the rest of the paragraph when paginating the document.
- Keep with next - whether the specified paragraph remains on the same page as the paragraph that follows it while paginating the document.
- Keep lines together - whether all lines in the specified paragraphs remain on the same page while paginating the document.

The following example code illustrates how to enable or disable these pagination properties for the selected paragraphs.

```
`typescript
```

```
this.documenteditor.selection.paragraphFormat.widowControl = false;
```

```
this.documenteditor.selection.paragraphFormat.keepWithNext = true;
```

```
this.documenteditor.selection.paragraphFormat.keepLinesTogether = true;
```

```
,
```

Paragraph Border

You can apply borders to the paragraphs in a Word document. Using borders, decorate the paragraphs to set them apart from other paragraphs in the document.

The following example code illustrates how to apply box border for the selected paragraphs.

```
`typescript
```

```
// left
```

```
this.documenteditor.selection.paragraphFormat.borders.left.lineStyle = 'Single';
```

```
this.documenteditor.selection.paragraphFormat.borders.left.lineWidth = 3;
```

```
this.documenteditor.selection.paragraphFormat.borders.left.color = "#000000";
```

```
//right
this.documenteditor.selection.paragraphFormat.borders.right.lineStyle = 'Single';
this.documenteditor.selection.paragraphFormat.borders.right.lineWidth = 3;
this.documenteditor.selection.paragraphFormat.borders.right.color = "#000000";

//top
this.documenteditor.selection.paragraphFormat.borders.top.lineStyle = 'Single';
this.documenteditor.selection.paragraphFormat.borders.top.lineWidth = 3;
this.documenteditor.selection.paragraphFormat.borders.top.color = "#000000";

//bottom
this.documenteditor.selection.paragraphFormat.borders.bottom.lineStyle = 'Single';
this.documenteditor.selection.paragraphFormat.borders.bottom.lineWidth = 3;
this.documenteditor.selection.paragraphFormat.borders.bottom.color = "#000000";
,
```

Note: At present, the Document editor component displays all the border styles as single line. But you can apply any border style and get the proper display in Microsoft Word app when opening the exported Word document.

Show or Hide Paragraph marks

You can show or hide the hidden formatting symbols like spaces, tab, paragraph marks, and breaks in Document editor component. These marks help identify the start and end of a paragraph and all the hidden formatting symbols in a Word document.

The following example code illustrates how to show or hide paragraph marks.

```
`typescript
this.documenteditor.documentEditorSettings.showHiddenMarks = true;
,
```

Toolbar with paragraph formatting options

The following sample demonstrates the paragraph formatting options using a toolbar.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { DocumentEditorAllModule } from '@syncfusion/ej2-angular-documenteditor'
import { ToolbarModule } from '@syncfusion/ej2-angular-navigations'
import { ComboBoxModule } from '@syncfusion/ej2-angular-dropdowns'
import { ColorPickerModule } from '@syncfusion/ej2-angular-inputs'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import {
    DocumentEditorComponent, EditorService, SelectionService,
    EditorHistoryService, SfdtExportService, ContextMenuService
} from '@syncfusion/ej2-angular-documenteditor';
```

```

@Component({
  imports: [

    ButtonModule,
    ToolbarModule,
    DocumentEditorAllModule,
    ComboBoxModule,
    ColorPickerModule
  ],
  standalone: true,
  selector: 'app-container',
  styleUrls: ['./style.css'],
  //specifies the template string for the Document Editor component
  template: `<div>
    <div>
      <ejs-toolbar (clicked)='toolbarButtonClick($event)'>
        <e-items>
          <e-item prefixIcon='e-de-ctnr-alignleft e-icons'
id='AlignLeft' tooltipText='Align Left'></e-item>
          <e-item prefixIcon='e-de-ctnr-aligncenter e-icons'
id='AlignCenter' tooltipText='AlignCenter'></e-item>
          <e-item prefixIcon='e-de-ctnr-alignright e-icons'
id='AlignRight' tooltipText='AlignRight'></e-item>
          <e-item prefixIcon='e-de-ctnr-justify e-icons' id='Justify'
tooltipText='Justify'></e-item>
          <e-item prefixIcon='e-de-ctnr-increaseindent e-icons'
id='IncreaseIndent' tooltipText='Increase Indent'></e-item>
          <e-item prefixIcon='e-de-ctnr-decreaseindent e-icons'
id='DecreaseIndent' tooltipText='Decrease Indent'></e-item>
          <e-item type='Separator'></e-item>
          <e-item prefixIcon='e-de-ctnr-clearall e-icons'
id='ClearFormat' tooltipText='ClearFormatting'></e-item>
          <e-item type='Separator'></e-item>
          <e-item prefixIcon='e-de-e-paragraph-mark e-icons'
id='ShowParagraphMark' tooltipText='Show the hidden characters like spaces,
tab, paragraph marks, and breaks.(Ctrl + *)'></e-item>
        </e-items>
      </ejs-toolbar>
    </div>
    <ejs-documenteditor #document_editor
(selectionChange)='onSelectionChange()' [enableSelection]='true'
[isReadOnly]='false' [enableEditor]=true [enableEditorHistory]=true
[enableSfdtExport]=true [enableContextMenu]=true height="330px"
style="display:block"></ejs-documenteditor>
  </div>`,
  encapsulation: ViewEncapsulation.None,
  providers: [EditorService, SelectionService, EditorHistoryService,
SfdtExportService, ContextMenuService]
})
export class AppComponent {
  @ViewChild('document_editor')
  public documentEditor?: DocumentEditorComponent;
  public toolbarButtonClick(arg: any) {
    switch (arg.item.id) {
      case 'AlignLeft':
        //Toggle the Left alignment for selected or current
paragraph

```

```

        (this.documentEditor as
DocumentEditorComponent).editor.toggleTextAlignment('Left');
        break;
        case 'AlignRight':
            //Toggle the Right alignment for selected or current
paragraph
            (this.documentEditor as
DocumentEditorComponent).editor.toggleTextAlignment('Right');
            break;
        case 'AlignCenter':
            //Toggle the Center alignment for selected or current
paragraph
            (this.documentEditor as
DocumentEditorComponent).editor.toggleTextAlignment('Center');
            break;
        case 'Justify':
            //Toggle the Justify alignment for selected or current
paragraph
            (this.documentEditor as
DocumentEditorComponent).editor.toggleTextAlignment('Justify');
            break;
        case 'IncreaseIndent':
            //Increase the left indent of selected or current paragraph
            (this.documentEditor as
DocumentEditorComponent).editor.increaseIndent();
            break;
        case 'DecreaseIndent':
            //Decrease the left indent of selected or current paragraph
            (this.documentEditor as
DocumentEditorComponent).editor.decreaseIndent();
            break;
        case 'ClearFormat':
            //Clear formatting for selected paragraph or content.
            (this.documentEditor as
DocumentEditorComponent).editor.clearFormatting();
            break;
        case 'ShowParagraphMark':
            //Show or hide the hidden characters like spaces, tab,
paragraph marks, and breaks.
            (this.documentEditor as
DocumentEditorComponent).documentEditorSettings.showHiddenMarks =
!(this.documentEditor as
DocumentEditorComponent).documentEditorSettings.showHiddenMarks;
            break;
    }
}
// Selection change to retrieve formatting
public onSelectionChange() {
    if ((this.documentEditor as DocumentEditorComponent).selection) {
        var paragraphFormat = (this.documentEditor as
DocumentEditorComponent).selection.paragraphFormat;
        var toggleBtnId = ['AlignLeft', 'AlignCenter', 'AlignRight',
'Justify', 'ShowParagraphMark'];
        //Remove toggle state.
        if (document.getElementById('AlignLeft')) {
            for (var i = 0; i < toggleBtnId.length; i++) {

```

```

        let toggleBtn: HTMLElement =
document.getElementById(toggleBtnId[i]) as HTMLElement;
        toggleBtn.classList.remove('e-btn-toggle');
    }
    //Add toggle state based on selection paragraph format.
    if (paragraphFormat.textAlignment === 'Left') {
        (document.getElementById('AlignLeft') as
HTMLElement).classList.add('e-btn-toggle');
    } else if (paragraphFormat.textAlignment === 'Right') {
        (document.getElementById('AlignRight') as
HTMLElement).classList.add('e-btn-toggle');
    } else if (paragraphFormat.textAlignment === 'Center') {
        (document.getElementById('AlignCenter') as
HTMLElement).classList.add('e-btn-toggle');
    } else {
        (document.getElementById('Justify') as
HTMLElement).classList.add('e-btn-toggle');
    }
    if ((this.documentEditor as
DocumentEditorComponent).documentEditorSettings.showHiddenMarks) {
        (document.getElementById('ShowParagraphMark') as
HTMLElement).classList.add('e-btn-toggle');
    }
    }
    // #endregion
}
}
}

```

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

- [Feature modules](#)
- [Paragraph dialog](#)
- [Keyboard shortcuts](#)

Styles in Angular Document editor component

Styles are useful for applying a set of formatting consistently throughout the document. In document editor, styles are created and added to a document programmatically or via the built-in Styles dialog.

Styles definition overview

A Style in document editor should have the following properties:

- **name:** Name of the style. All styles in a document have a unique name, which is used as an identifier when applying the style.

- **type**: Specifies the document elements that the style will target. For example, paragraph or character.
- **next**: Specifies that the current style inherits the style set to this property. This is how hierarchical styles are defined.
- **link**: Provides a relation between the paragraph and character style.
- **characterFormat**: Specifies the properties of paragraph and character style.
- **paragraphFormat**: Specifies the properties of paragraph style.
- **basedOn**: Specifies that the current style inherits the style set to this property. This is how hierarchical styles are defined. It can be optional.

The style type should match the inherited style type. For example, it is not possible to have a character style inherit a paragraph style.

Default style

The default style for span and paragraph properties is normal. It internally inherits the default style of the document loaded or document editor component.

Style hierarchy

Each style initially checks its local value for the property that is being evaluated and turns to the style it is based on. If no local value is found, it turns to its default style.

Style inheritance of different styles are listed as follows:

Character style

Character styles are based only on other character styles.

The inheritance is: Character properties are inherited from the base character style.

Paragraph style

Paragraph styles are based on other paragraph styles or on linked styles.

When a paragraph style is based on another paragraph style, the inheritance of the properties is as follows:

- Paragraph properties are inherited from the base paragraph style.
- Span properties are inherited from the base paragraph style.

When a paragraph style is based on a linked style, the inheritance of the properties is as follows:

- Paragraph properties are inherited from the paragraph style part in its base linked style.
- Span properties are inherited from the span style part in its base linked style.

Linked style

Linked styles are composite styles and their components are paragraph and character styles with link between them. To apply paragraph properties, take the properties from the linked paragraph style. Similarly, to apply character properties, take the properties from linked character style.

Linked styles are based on other linked styles or on paragraph styles.

When a linked style is based on a paragraph style, the hierarchy of the properties is as follows:

- Paragraph properties are inherited from the 'basedOn' paragraph style.

- Character properties are inherited from the 'basedOn' paragraph style.

When a linked style is based on another linked style, the hierarchy of the properties is as follows:

- Paragraph properties are inherited from the paragraph style part in its base linked style.
- Span properties are inherited from the span style part in its base linked style.

Defining new styles

New Styles are defined and added to the style collection of the document. In this way, they will be discovered by the default UI and applied to the parts of a document.

Defining a character style

The following example shows how to programmatically create a character style.

```
`typescript
let styleJson: any = {
  "type": "Character",
  "name": "New CharacterStyle",
  "basedOn": "Default Paragraph Font",
  "characterFormat": {
    "fontSize": 16.0,
    "fontFamily": "Calibri Light",
    "fontColor": "#2F5496",
    "bold": true,
    "italic": true,
    "underline": "Single"
  }
};
this.documentEditor.editor.createStyle(JSON.stringify(styleJson));
`
```

Defining a paragraph style

The following example shows how to programmatically create a paragraph style.

```
`typescript
let styleJson: any = {
  "type": "Paragraph",
  "name": "New ParagraphStyle",
  "basedOn": "Normal",
  "characterFormat": {
```

```
"fontSize": 16.0,
"fontFamily": "Calibri Light",
"fontColor": "#2F5496",
"bold": true,
"italic": true,
"underline": "Single"
},
"paragraphFormat": {
"leftIndent": 0.0,
"rightIndent": 0.0,
"firstLineIndent": 0.0,
"beforeSpacing": 12.0,
"afterSpacing": 0.0,
"lineSpacing": 1.0791666507720947,
"lineSpacingType": "Multiple",
"textAlignment": "Left",
"outlineLevel": "Level1"
}
};
this.documentEditor.editor.createStyle(JSON.stringify(styleJson));
`
```

Defining a linked style

The following example shows how to programmatically create linked style.

```
`typescript
let styleJson: any = {
"type": "Paragraph",
"name": "New Linked",
"basedOn": "Normal",
"next": "Normal",
"link": "New Linked Char",
"characterFormat": {
"fontSize": 16.0,
"fontFamily": "Calibri Light",
```

```
"fontColor": "#2F5496"
},
"paragraphFormat": {
  "leftIndent": 0.0,
  "rightIndent": 0.0,
  "firstLineIndent": 0.0,
  "beforeSpacing": 12.0,
  "afterSpacing": 0.0,
  "lineSpacing": 1.0791666507720947,
  "lineSpacingType": "Multiple",
  "textAlignment": "Left",
  "outlineLevel": "Level1"
}
};
this.documentEditor.editor.createStyle(JSON.stringify(styleJson));
`
```

Applying a style

The styles are applied using the **applyStyle** method of **Editor**, the parameter should be passed is the **Name** of the Style.

The styles of the **Character** type is applied to the currently selected part of the document. If there is no selection, the values that will be applied to the word at caret position. The styles of **Paragraph** type follow the same logic and are applied to all paragraphs in the selection or the current paragraph.

When there is no selection, styles of **Linked** type will change the values of the paragraph, and apply both the Paragraph and Character properties. When there is selection, Linked Style changes only the character properties of the selected text.

For example, the following line will apply the "New Linked" to the current paragraph.

```
`typescript
this.documentEditor.editor.applyStyle('New Linked');
//Clear direct formatting and apply the specified style
this.documentEditor.editor.applyStyle('New Linked', true);
`
```

Get Styles

You can get the styles in the document using the below code snippet.

```
`typescript
//Get paragraph styles
```

```
let paragraphStyles = this.documentEditor.getStyles('Paragraph');
//Get character styles
let paragraphStyles = this.documentEditor.getStyles('Character');
`
```

Modify an existing style

You can modify a existing style with the specified style properties using [createStyle](#) method. If modifyExistingStyle parameter is set to `true` the style properties is updated to the existing style.

The following illustrate to modify an existing style.

```
`typescript
let styleJson: any = {
  "type": "Paragraph",
  "name": "Heading 1",
  "characterFormat": {
    "fontSize": 32,
    "fontFamily": "Calibri"
  }
};
this.documentEditor.editor.createStyle(styleName, true);
`
```

If modifyExistingStyle parameter is set to true and a style already exists with same name, it modifies the specified properties in the existing style.

If modifyExistingStyle parameter is set to false and a style already exists with same name, it creates a new style with unique name by appending ‘_1’. Hence, the newly style will not have the specified name.

If no style exists with same name, it creates a new style.

List format in Angular Document editor component

Document Editor supports both the single-level and multilevel lists. Lists are used to organize data as step-by-step instructions in documents for easy understanding of key points. You can apply list to the paragraph either using supported APIs.

Create bullet list

Bullets are usually used for unordered lists. To apply bulleted list for selected paragraphs, use the following method of ‘Editor’ instance.

```
applyBullet(bullet, fontFamily);
```

Parameter	Type	Description
bullet	string	Bullet character.
fontFamily	string	Bullet font family.

Refer to the following sample code.

```
`typescript
this.documentEditor.editor.applyBullet('\uf0b7', 'Symbol');
`
```

Create numbered list

Numbered lists are usually used for ordered lists. To apply numbered list for selected paragraphs, use the following method of 'Editor' instance.

applyNumbering(numberFormat,listLevelPattern)

Parameter	Type	Description
numberFormat	string	"%n" representations in 'numberFormat' parameter will be replaced by respective list level's value. "%1" will be displayed as "1"
listLevelPattern(optional)	string	Default value is 'Arabic'.

Refer to the following example.

```
`typescript
this.documentEditor.editor.applyNumbering('%1', 'UpRoman');
`
```

Clear list

You can also clear the list formatting applied for selected paragraphs. Refer to the following sample code.

```
`typescript
this.documentEditor.editor.clearList();
`
```

Working with lists

The following sample demonstrates how to create bullet and numbering lists in document editor.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { DocumentEditorAllModule } from '@syncfusion/ej2-angular-documenteditor'
import { ToolbarModule } from '@syncfusion/ej2-angular-navigations'
import { ComboBoxModule } from '@syncfusion/ej2-angular-dropdowns'
import { ColorPickerModule } from '@syncfusion/ej2-angular-inputs'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import {
    DocumentEditorComponent, EditorService, SelectionService,
    EditorHistoryService, SfdtExportService
} from '@syncfusion/ej2-angular-documenteditor';
@Component({
    imports: [
```

```

        ButtonModule,
        ToolbarModule,
        DocumentEditorAllModule,
        ComboBoxModule,
        ColorPickerModule
    ],
    standalone: true,
    selector: 'app-container',
    styleUrls: ['./style.css'],
    //specifies the template string for the Document Editor component
    template: `<div>
        <div>
            <ejs-toolbar (clicked)='toolbarButtonClick($event)'>
                <e-items>
                    <e-item prefixIcon="e-de-ctnr-bullets e-icons"
tooltipText="Bullets" id="Bullets"></e-item>
                    <e-item prefixIcon="e-de-ctnr-numbering e-icons"
tooltipText="Numbering" id="Numbering"></e-item>
                    <e-item text="Clear" id="clearlist" tooltipText="Clear
List"></e-item>
                </e-items>
            </ejs-toolbar>
        </div>
        <ejs-documenteditor #document_editor [enableSelection]='true'
[isReadOnly]='false' [enableEditor]=true [enableEditorHistory]=true
[enableSfdtExport]=true height="330px" style="display:block"></ejs-
documenteditor>
    </div>`,
    encapsulation: ViewEncapsulation.None,
    //Provide necessary service.
    providers: [EditorService, SelectionService, EditorHistoryService,
SfdtExportService]
})
export class AppComponent {
    @ViewChild('document_editor')
    public documentEditor?: DocumentEditorComponent;
    public toolbarButtonClick(args: any) {
        switch (args.item.id) {
            case 'Bullets':
                //To create bullet list
                (this.documentEditor as
DocumentEditorComponent).editor.applyBullet('\uf0b7', 'Symbol');
                break;
            case 'Numbering':
                //To create numbering list
                (this.documentEditor as
DocumentEditorComponent).editor.applyNumbering('%1', 'UpRoman');
                break;
            case 'clearlist':
                //To clear list
                (this.documentEditor as
DocumentEditorComponent).editor.clearList();
                break;
        }
    }
}

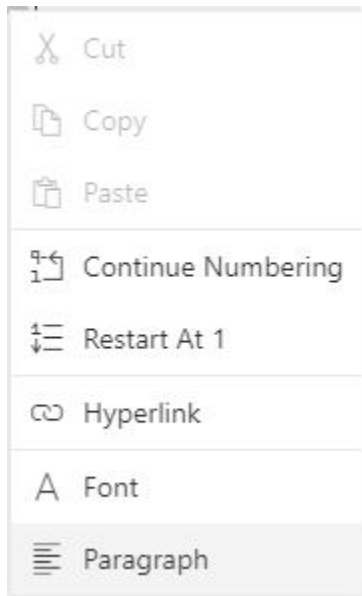
```

MAIN.TS

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

Editing numbered list

Document Editor restarts the numbering or continue numbering for a numbered list. These options are found in the built-in context menu, if the list value is selected. Refer to the following screenshot.



See Also

- [List dialog](#)

Table format in Angular Document editor component

Document Editor customizes the formatting of table, or table cells such as table width, cell margins, cell spacing, background color, and table alignment. This section describes how to customize these formatting for selected cells, rows, or table in detail.

Cell margins

You can customize the cell margins by using the following sample code.

```
`typescript
//To change the left margin
this.documentEditor.selection.cellFormat.leftMargin=5.4;
//To change the right margin
this.documentEditor.selection.cellFormat.rightMargin=5.4;
//To change the top margin
```

```
this.documentEditor.selection.cellFormat.topMargin=5.4;  
//To change the bottom margin  
this.documentEditor.selection.cellFormat.bottomMargin=5.4;  
`
```

You can also define the default cell margins for a table. If the specific cell margin value is not defined explicitly in the cell formatting, the corresponding value will be retrieved from default cells margin of the table. Refer to the following sample code.

```
`typescript  
//To change the left margin  
this.documentEditor.selection.tableFormat.leftMargin=5.4;  
//To change the right margin  
this.documentEditor.selection.tableFormat.rightMargin=5.4;  
//To change the top margin  
this.documentEditor.selection.tableFormat.topMargin=5.4;  
//To change the bottom margin  
this.documentEditor.selection.tableFormat.bottomMargin=5.4;  
`
```

Background color

You can explicitly set the background color of selected cells using the following sample code.

```
`typescript  
this.documentEditor.selection.cellFormat.background='#E0E0E0';  
`
```

Refer to the following sample code to customize the background color of the table.

```
`typescript  
this.documentEditor.selection.tableFormat.background='#E0E0E0';  
`
```

Cell spacing

Refer to the following sample code to customize the spacing between each cell in a table.

```
`typescript  
this.documentEditor.selection.tableFormat.cellSpacing=2;  
`
```

Cell vertical alignment

The content is aligned within a table cell to 'Top', 'Center', or 'Bottom'. You can customize this property of selected cells. Refer to the following sample code.

```
`typescript
```



```
this.documentEditor.selection.cellFormat.verticalAlignment='Bottom';  
`
```

Table alignment

The tables are aligned in document editor to 'Left', 'Right', or 'Center'. Refer to the following sample code.

```
`typescript  
this.documentEditor.selection.tableFormat.tableAlignment='Center';  
`
```

Cell width

Set the desired width of table cells that will be considered when the table is layouted. Refer to the following sample code.

```
`typescript  
this.documentEditor.selection.cellFormat.preferredWidthType='Point';  
this.documentEditor.selection.cellFormat.preferredWidth=100;  
`
```

Table width

You can set the desired width of a table in **Point** or **Percent** type. Refer to the following sample code.

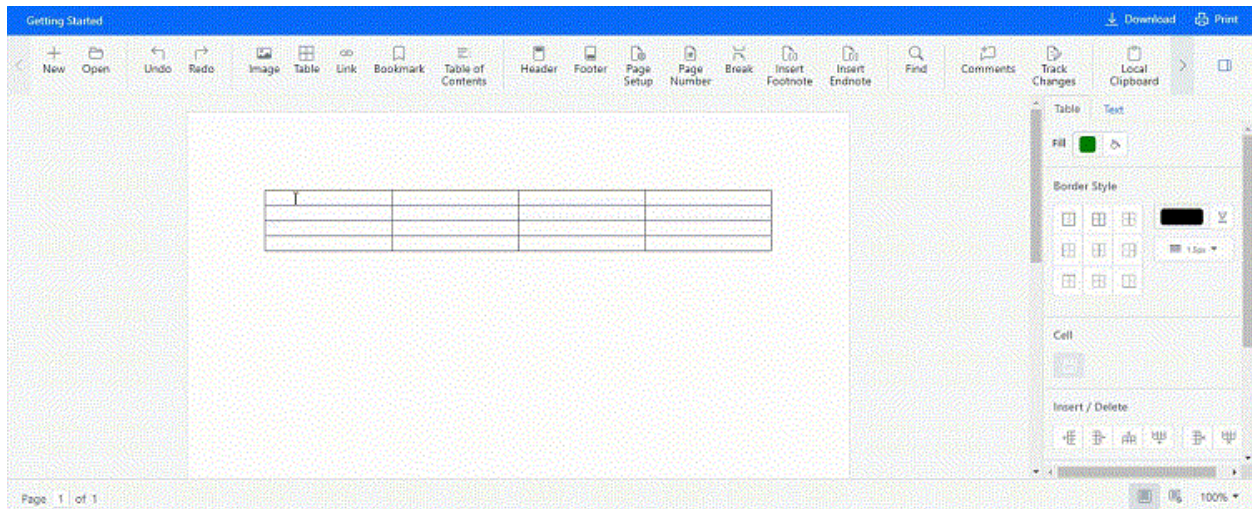
```
`typescript  
this.documentEditor.selection.tableFormat.preferredWidthType='Point';  
this.documentEditor.selection.tableFormat.preferredWidth=300;  
`
```

Apply borders

Document Editor exposes API to customize the borders for table cells by specifying the settings. Refer to the following sample code.

```
`typescript  
import { BorderSettings } from '@syncfusion/ej2-documenteditor';  
//To apply border  
let borderSettings: BorderSettings = {  
  type: 'AllBorders',  
  lineWidth: 12  
};  
this.documentEditor.editor.applyBorders(borderSettings);  
`
```

Please check below gif which illustrates how to apply border for selected cells through properties pane options - border color, line size and no border:



Working with row formatting

Document Editor allows various row formatting such as height and repeat header.

Row height

You can customize the height of a table row as 'Auto', 'AtLeast', or 'Exactly'. Refer to the following sample code.

```
`typescript
```

```
this.documentEditor.selection.rowFormat.heightType='Exactly';
```

```
this.documentEditor.selection.rowFormat.height=20;
```

```
`
```

Header row

The header row describes the content of a table. A table can optionally have a header row. Only the first row of a table can be the header row. If the cursor position is at first row of the table, then you can define whether it as header row or not, using the following sample code.

```
`typescript
```

```
this.documentEditor.selection.rowFormat.isHeader=true;
```

```
`
```

Allow row break across pages

This property is valid if a table row does not fit in the current page during table layout. It defines whether a table row can be allowed to break. If the value is false, the entire row will be moved to the start of next page. You can modify this property for selected rows using the following sample code.

```
`typescript
```

```
this.documentEditor.selection.rowFormat.allowRowBreakAcrossPages=false;
```

```
`
```

Title

Document Editor expose API to get or set the table title of the selected table. Refer to the following sample code to set title.

```
`typescript
```

```
this.documenteditor.selection.tableFormat.title = 'Shipping Details';
```

```
,
```

Description

Document Editor expose API to get or set the table description of the selected image. Refer to the following sample code to set description.

```
`typescript
```

```
this.documenteditor.selection.tableFormat.description = 'Freight cost and shipping details';
```

```
,
```

See Also

- [Table properties dialog](#)

Section format in Angular Document editor component

Document Editor supports various section formatting such as page size, page margins, and more.

Page size

You can get or set the size of a section at cursor position by using the following sample code.

```
`typescript
```

```
this.documentEditor.selection.sectionFormat.pageWidth = 500;
```

```
this.documentEditor.selection.sectionFormat.pageHeight = 600;
```

```
,
```

You can change the orientation of the page by swapping the values of page width and height respectively.

Page margins

Left and right page margin defines the gap between the document content from left and right side of the page respectively. Top and bottom page margins defines the gap between the document content from header and footer of the page respectively.

Refer to the following sample code.

```
`typescript
```

```
this.documentEditor.selection.sectionFormat.leftMargin = 10;
```

```
this.documentEditor.selection.sectionFormat.rightMargin = 10;
```

```
this.documentEditor.selection.sectionFormat.bottomMargin = 10;
```

```
this.documentEditor.selection.sectionFormat.topMargin = 10;
```

```
,
```

Header distance

You can define the distance of header content from the top of the page by using the following sample code.

```
`typescript
```

```
this.documentEditor.selection.sectionFormat.headerDistance = 72;
```

```
,
```

Footer distance

You can define the distance of footer content from the bottom of the page by using the following sample code.

```
`typescript
```

```
this.documentEditor.selection.sectionFormat.footerDistance = 72;
```

```
,
```

Columns

You can define the number of columns, column width, and space between columns for the pages in a section.

The following code example illustrates how to define the two columns layout for the pages in a section.

```
`typescript
```

```
let column: SelectionColumnFormat = new
```

```
SelectionColumnFormat(this.container.documentEditor.selection);
```

```
column.width = 216;
```

```
column.space = 36;
```

```
this.container.documentEditor.selection.sectionFormat.columns = [column, column];
```

```
this.container.documentEditor.selection.sectionFormat.lineBetweenColumns = true;
```

```
,
```

Breaks

You can insert Column break

The following code indicate that the text following the column break will begin in the next column

```
`typescript
```

```
this.container.documentEditor.editor.insertColumnBreak();
```

```
,
```

You can insert next page section break to start the new section on the next page

The following code example illustrates how to insert a next page section break

```
`typescript
```

```
this.container.documentEditor.editor.insertSectionBreak(SectionBreakType.NewPage);
```

```
,
```

You can insert continuous section break to start the new section on the same page

The following code example illustrates how to insert a continuous section break

```
`typescript
```

```
this.container.documentEditor.editor.insertSectionBreak(SectionBreakType.Continuous);  
`
```

See Also

*[Page setup dialog](#)

Comments in Angular Document editor component

Document Editor allows you to add comments to documents. You can add, navigate and remove comments in code and from the UI.

Add a new comment

Comments can be inserted to the selected text.

```
`typescript  
this.documentEditor.editor.insertComment("Test comment");  
`
```

Comment navigation

Next and previous comments can be navigated using the below code snippet.

```
`typescript  
//Navigate to next comment  
this.documentEditor.selection.navigateNextComment();  
//Navigate to previous comment  
this.documentEditor.selection.navigatePreviousComment();  
`
```

Delete comment

Current comment can be deleted using the below code snippet.

```
`typescript  
this.documentEditor.editor.deleteComment();  
`
```

Delete all comment

All the comments in the document can be deleted using the below code snippet.

```
`typescript  
this.documentEditor.editor.deleteAllComments();  
`
```

Protect the document in comments only mode

Document Editor provides support for protecting the document with **CommentsOnly** protection. In this protection, user allowed to add or edit comments alone in the document.

Document editor provides an option to protect and unprotect document using [enforceProtection](#) and [stopProtection](#) API.

The following example code illustrates how to enforce and stop protection in Document editor container.

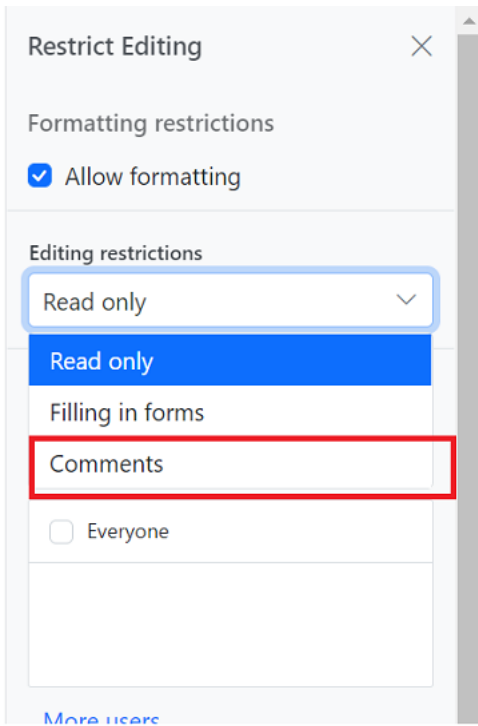
```
`typescript
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import { DocumentEditorContainerComponent, ToolbarService } from '@syncfusion/ej2-angular-documenteditor';

@Component({
  selector: 'app-container',
  // specifies the template string for the Document Editor component
  template: `<div><button ej2-button (click)="protectDocument()" >Protect</button>
<button ej2-button (click)="unProtectDocument()" >Unprotect</button>
<ejs-documenteditorcontainer #document_editor
serviceUrl="https://ej2services.syncfusion.com/production/web-services/api/documenteditor/"
height="600px" style="display:block" [enableToolbar]=true> </ejs-documenteditorcontainer></div>`,
  encapsulation: ViewEncapsulation.None,
  providers: [ToolbarService]
})
export class AppComponent {
  @ViewChild('document_editor')
  public container: DocumentEditorContainerComponent;

  public protectDocument(): void {
    //enforce protection
    container.documentEditor.editor.enforceProtection('123', 'CommentsOnly');
  }

  public unProtectDocument(): void {
    //stop the document protection
    container.documentEditor.editor.stopProtection('123');
  }
}
```

Comment only protection can be enabled in UI by using [Restrict Editing pane](#)



Note: In enforce Protection method, first parameter denotes password and second parameter denotes protection type. Possible values of protection type are NoProtection | ReadOnly | FormFieldsOnly | CommentsOnly. In stop protection method, parameter denotes the password.

Mention support in Comments

Mention support displays a list of items that users can select or tag from the suggested list. To use this feature, type the @ character in the comment box and select or tag the user from the suggestion list.

The following example illustrates how to enable mention support in the Document Editor

```
`typescript
import { Component, OnInit } from '@angular/core';
import { ToolbarService, DocumentEditorSettingsModel } from '@syncfusion/ej2-angular-documenteditor';

@Component({
  selector: 'app-root',
  // specifies the template string for the DocumentEditorContainer component
  template: <ejs-documenteditorcontainer
    serviceUrl="https://ej2services.syncfusion.com/production/web-
    services/api/documenteditor/" height="600px" style="display:block"
    [documentEditorSettings]= "settings" [enableToolbar]=true> </ejs-documenteditorcontainer>,
  providers: [ToolbarService]
})
export class AppComponent implements OnInit {
```

```

public mentionData: any = [
  { "Name": "Selma Rose", "Eimg": "3", "EmailId": "selma@mycompany.com" },
  { "Name": "Russo Kay", "Eimg": "8", "EmailId": "russo@mycompany.com" },
  { "Name": "Camden Kate", "Eimg": "9", "EmailId": "camden@mycompany.com" },
  { "Name": "Mary Kate", "Eimg": "4", "EmailId": "marry@mycompany.com" },
  { "Name": "Ursula Ann", "Eimg": "2", "EmailId": "ursula@mycompany.com" },
  { "Name": "Margaret", "Eimg": "5", "EmailId": "margaret@mycompany.com" },
  { "Name": "Laura Grace", "Eimg": "6", "EmailId": "laura@mycompany.com" },
  { "Name": "Robert", "Eimg": "8", "EmailId": "robert@mycompany.com" },
  { "Name": "Albert", "Eimg": "9", "EmailId": "albert@mycompany.com" },
  { "Name": "Michale", "Eimg": "10", "EmailId": "michale@mycompany.com" },
  { "Name": "Andrew James", "Eimg": "7", "EmailId": "james@mycompany.com" },
  { "Name": "Rosalie", "Eimg": "4", "EmailId": "rosalie@mycompany.com" },
  { "Name": "Stella Ruth", "Eimg": "2", "EmailId": "stella@mycompany.com" },
  { "Name": "Richard Rose", "Eimg": "10", "EmailId": "richard@mycompany.com" },
  { "Name": "Gabrielle", "Eimg": "3", "EmailId": "gabrielle@mycompany.com" },
  { "Name": "Thomas", "Eimg": "7", "EmailId": "thomas@mycompany.com" },
  { "Name": "Charles Danny", "Eimg": "8", "EmailId": "charles@mycompany.com" },
  { "Name": "Daniel", "Eimg": "10", "EmailId": "daniel@mycompany.com" },
  { "Name": "Matthew", "Eimg": "7", "EmailId": "matthew@mycompany.com" },
  { "Name": "Donald Krish", "Eimg": "9", "EmailId": "donald@mycompany.com" },
  { "Name": "Yohana", "Eimg": "1", "EmailId": "yohana@mycompany.com" },
  { "Name": "Kevin Paul", "Eimg": "10", "EmailId": "kevin@mycompany.com" },
  { "Name": "Andrew Fuller", "Eimg": "3", "EmailId": "andrew@mycompany.com" }
];

public settings: DocumentEditorSettingsModel = { mentionSettings : { dataSource: this.mentionData,
fields: { text: 'Name' } } };

ngOnInit(): void {
}

}
,

```

Fields in Angular Document editor component

Document Editor has preservation support for all types of fields in an existing word document without any data loss.

Adding Fields

You can add a field to the document by using [insertField](#) method in [Editor](#) module.

The following example code illustrates how to insert merge field programmatically by providing the field code and field result.

```
`typescript
let fieldCode: string = 'MERGEFIELD First Name \* MERGEFORMAT ';
let fieldResult: string = '«First Name»';
this.documentEditor.editor.insertField(fieldCode, fieldResult);
`
```

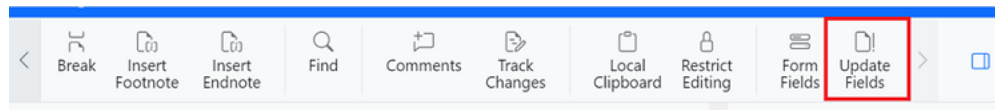
Note: Document editor does not validate/process the field code/field result. it simply inserts the field with specified field information.

Update fields

Document Editor provides support for updating bookmark cross reference field. The following example code illustrates how to update bookmark cross reference field.

```
`typescript
//Update all the bookmark cross reference field in the document.
this.documentEditor.updateFields();
`
```

Bookmark cross reference fields can be updated through UI by using update fields option in **Toolbar**.



The following type of fields are automatically updated in Document Editor.

- Numpages
- Section
- Page

Get field info

You can get field code and field result of the current selected field by using [getFieldInfo](#) method in the [Selection](#) module.

```
`typescript
//Gets the field information of the selected field.
let fieldInfo: FieldInfo = this.documentEditor.selection.getFieldInfo();
`
```

Note: For nested fields, this method returns combined field code and result.

Set field info

You can modify the field code and field result of the current selected field by using [setFieldInfo](#) method in the [Editor](#) module.

```
`typescript
//Gets the field information for the selected field.
let fieldInfo: FieldInfo = this.documentEditor.selection.getFieldInfo();
//Modify field code
fieldInfo.code = 'MERGEFIELD First Name \* MERGEFORMAT ';
//Modify field result
fieldInfo.result = '«First Name»';
//Modify field code and result of the current selected field.
this.documentEditor.editor.setFieldInfo(fieldInfo);
`
```

Note: For nested field, entire field gets replaced completely with the specified field information.

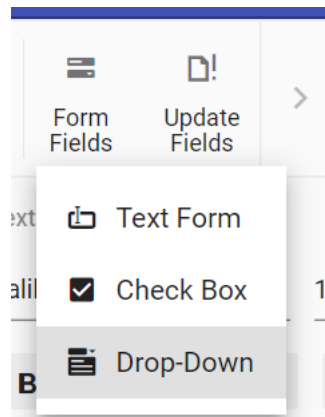
See Also

[Mail merge using Word library \(DocIO\)](#)

[Mail merge demo](#)

Form fields in Angular Document editor component

DocumentEditorContainer component provide support for inserting Text, CheckBox, DropDown form fields through in-built toolbar.



Insert form field

Form fields can be inserted using [insertFormField](#) method in editor module.

```
`typescript
//Insert Text form field
this.documentEditor.editor.insertFormField('Text');
//Insert Checkbox form field
```

```
this.documentEditor.editor.insertFormField('CheckBox');
```

```
//Insert Drop down form field
```

```
this.documentEditor.editor.insertFormField('Dropdown');
```

```
,
```

Get form field names

All the form fields names form current document can be retrieved using [getFormFieldNames\(\)](#).

```
`typescript
```

```
let formFieldsNames: string[] = this.documentEditor.getFormFieldNames();
```

```
,
```

Get form field properties

Form field properties can be retrieved using [getFormFieldInfo\(\)](#).

```
`typescript
```

```
//Text form field
```

```
let textfieldInfo: TextFormFieldInfo = this.documentEditor.getFormFieldInfo('Text1') as  
TextFormFieldInfo;
```

```
//Checkbox form field
```

```
let checkboxfieldInfo: CheckBoxFormFieldInfo = this.documentEditor.getFormFieldInfo('Check1') as  
CheckBoxFormFieldInfo;
```

```
//Dropdown form field
```

```
let dropdownfieldInfo: DropDownFormFieldInfo = this.documentEditor.getFormFieldInfo('Drop1') as  
DropDownFormFieldInfo;
```

```
,
```

Set form field properties

Form field properties can be modified using [setFormFieldInfo\(\)](#).

```
`typescript
```

```
// Set text form field properties
```

```
let textfieldInfo: TextFormFieldInfo = this.documentEditor.getFormFieldInfo('Text1') as  
TextFormFieldInfo;
```

```
textfieldInfo.defaultValue = "Hello";
```

```
textfieldInfo.format = "Uppercase";
```

```
textfieldInfo.type = "Text";
```

```
textfieldInfo.name = "Text2";
```

```
this.documentEditor.setFormFieldInfo('Text1',textfieldInfo);
```

```
// Set checkbox form field properties
```

```
let checkboxfieldInfo: CheckBoxFormFieldInfo = this.documentEditor.getFormFieldInfo('Check1') as
CheckBoxFormFieldInfo;
checkboxfieldInfo.defaultValue = true;
checkboxfieldInfo.name = "Check2";
this.documentEditor.setFormFieldInfo('Check1',checkboxfieldInfo);
// Set checkbox form field properties
let dropdownfieldInfo: DropDownFormFieldInfo = this.documentEditor.getFormFieldInfo('Drop1') as
DropDownFormFieldInfo;
dropdownfieldInfo.dropDownItems = ['One','Two', 'Three'];
dropdownfieldInfo.name = "Drop2";
this.documentEditor.setFormFieldInfo('Drop1',dropdownfieldInfo);
`
```

Note:If a form field already exists in the document with the new name specified, the old form field name property will be cleared and it will not be accessible. Ensure the new name is unique.

Export form field data

Data of the all the Form fields in the document can be exported using [exportFormData](#).

`typescript

```
let formFieldDate: FormFieldData[] = this.documentEditor.exportFormData();
`
```

Import form field data

Form fields can be prefilled with data using [importFormData](#).

`typescript

```
let textformField: FormFieldData = {fieldName: 'Text1', value: 'Hello World'};
let checkformField: FormFieldData = {fieldName: 'Check1', value: true};
let dropdownformField: FormFieldData = {fieldName: 'Drop1', value: 1};
//Import form field data
this.documentEditor.importFormData([textformField,checkformField,dropdownformField]);
`
```

Reset form fields

Reset all the form fields in current document to default value using [resetFormFields](#).

`typescript

```
this.documentEditor.resetFormFields();
`
```

Protect the document in form filling mode

Document Editor provides support for protecting the document with **FormFieldsOnly** protection. In this protection, user can only fill form fields in the document.

Document editor provides an option to protect and unprotect document using [enforceProtection](#) and [stopProtection](#) API.

The following example code illustrates how to enforce and stop protection in Document editor container.

```
`typescript
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import { DocumentEditorContainerComponent, ToolbarService } from '@syncfusion/ej2-angular-documenteditor';

@Component({
  selector: 'app-container',
  // specifies the template string for the Document Editor component
  template: `<div><button ej2-button (click)="protectDocument()" >Protect</button>
<button ej2-button (click)="unProtectDocument()" >Unprotect</button>
<ejs-documenteditorcontainer #document_editor
serviceUrl="https://ej2services.syncfusion.com/production/web-services/api/documenteditor/"
height="600px" style="display:block" [enableToolbar]=true> </ejs-documenteditorcontainer></div>`,
  encapsulation: ViewEncapsulation.None,
  providers: [ToolbarService]
})
export class AppComponent {
  @ViewChild('document_editor')
  public container: DocumentEditorContainerComponent;

  public protectDocument(): void {
    //enforce protection
    container.documentEditor.editor.enforceProtection('123', 'FormFieldsOnly');
  }

  public unProtectDocument(): void {
    //stop the document protection
    container.documentEditor.editor.stopProtection('123');
  }
}
```

Note: In enforce Protection method, first parameter denotes password and second parameter denotes protection type. Possible values of protection type are `NoProtection` | `ReadOnly` | `FormFieldsOnly` | `CommentsOnly`. In stop protection method, parameter denotes the password.

Clipboard in Angular Document editor component

Document Editor takes advantage of system clipboard and allows you to copy or move a portion of the document into it in HTML format, so that it can be pasted in any application that supports clipboard.

Copy

Copy a portion of document to system clipboard using built-in context menu of document editor. You can also do it programmatically using the following sample code.

```
`typescript
this.documentEditor.selection.copy();
`
```

Cut

Cut a portion of document to system clipboard using built-in context menu of document editor. You can also do it programmatically using the following sample code.

```
`typescript
this.documentEditor.editor.cut();
`
```

Paste

Due to limitations, you can paste contents from system clipboard in Document Editor only using the 'CTRL + V' keyboard shortcut.

Note: Due to browser limitation of getting content from system clipboard, paste using API and context menu option doesn't work.

Local paste (copy/paste within control)

Document Editor expose API to enable local paste within the control. On enabling this, the following is performed:

- Selected contents will be stored to an internal clipboard in addition to system clipboard.
- Clipboard paste will be overridden, and internally stored data (SFDt data) that has formatted text will be pasted using paste() API in Document editor.

Refer to the following sample code.

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

import {
  DocumentEditorComponent, EditorService, SelectionService, SfdtExportService, EditorHistoryService,
  BookmarkDialogService
} from '@syncfusion/ej2-angular-documenteditor';

@Component({
```

```

selector: 'app-container',
//specifies the template string for the Document Editor component
template: `<div><button ejb-button (click)="pasteLocal()" >Paste local</button>
<ejs-documenteditor #document_editor id="container" height="330px" style="display:block"
[isReadOnly]=false [enableEditor]=true [enableLocalPaste]=true>
</ejs-documenteditor></div>`,
encapsulation: ViewEncapsulation.None,
providers: [EditorService, SelectionService, SfdtExportService]
})
export class AppComponent {
@ViewChild('document_editor')
public documentEditor: DocumentEditorComponent;
public pasteLocal(): void {
//paste copied content.
this.documentEditor.editor.paste();
}
}
`

```

By default, **enableLocalPaste** is false.

When local paste is enabled for a document editor instance, you can paste contents programmatically if the internal clipboard has stored data during last copy operation. Refer to the following sample code.

```

`typescript
this.documentEditor.editor.paste();
`

```

Paste options in context menu

In Document editor, paste options in context menu will be in disabled state if you were try to copy/paste content from outside of Document editor. It gets enabled when **enableLocalPaste** is true and trying to copy/paste content inside Document editor.

Note: Due to browser limitation of getting content from system clipboard, paste using API and context menu option doesn't work. Hence, the paste option is disabled in context menu.

Alternatively, you can use the keyboard shortcuts,

- Cut: Ctrl + X
- Copy: Ctrl + C
- Paste: Ctrl + V

EnableLocalPaste behaviour

|EnableLocalPaste |Paste behavior details|

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

|True |Allows to paste content that is copied from the same Document Editor component alone and prevents pasting content from system clipboard. Hence the content copied from outside Document Editor component can't be pasted.
Browser limitation of pasting from system clipboard using API and context menu options, will be resolved. So, you can copy and paste content within the Document Editor component using API and context menu options too. |

|False |Allows to paste content from system clipboard. Hence the content copied from both the Document Editor component and outside can be pasted.
Browser limitation of pasting from system clipboard using API and context menu options, will remain as a limitation. |

Note:

- Keyboard shortcut for pasting will work properly in both cases.
- Copying content from Document Editor component and pasting outside will work properly in both cases.

Paste with formatting

Document Editor provides support to paste the system clipboard data with formatting. To enable clipboard paste with formatting options, set the `enableLocalPaste` property in Document Editor to false and use this .NET Standard library [Syncfusion.EJ2.WordEditor.AspNet.Core](#) by the web API service implementation. This library helps you to paste the system clipboard data with formatting.

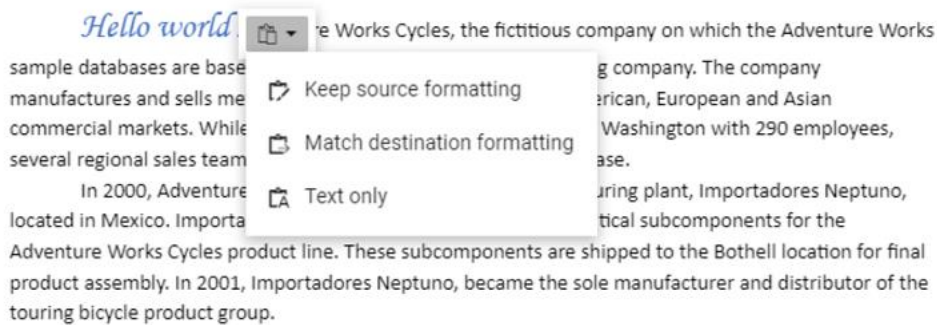
Refer this [page](#) for more details.

You can paste your system clipboard data in the following ways:

- **Keep Source Formatting** This option retains the character styles and direct formatting applied to the copied text. Direct formatting includes characteristics such as font size, italics, or other formatting that is not included in the paragraph style.
- **Match Destination Formatting** This option discards most of the formatting applied directly to the copied text, but it retains the formatting applied for emphasis, such as bold and italic when it is applied to only a portion of the selection. The text takes on the style characteristics of the paragraph where it is pasted. The text also takes on any direct formatting or character style properties of text that immediately precedes the cursor when the text is pasted.
- **Text Only** This option discards all formatting and non-text elements such as pictures or tables. The text takes on the style characteristics of the paragraph where it is pasted and takes on any direct formatting or character style properties of text that immediately precedes the cursor when the text is pasted. Graphical elements are discarded and tables are converted to a series of paragraphs.

This paste option appears as follows.

Adventure Works Cycles



See Also

- Feature modules
- Keyboard shortcuts

History in Angular Document editor component

Document Editor tracks the history of all editing actions done in the document, which allows undo and redo functionality.

Enable or disable history

Inject the `EditorHistory` module in your application to provide history preservation functionality for `DocumentEditor`. Refer to the following code example.

```
`typescript
```

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

import { DocumentEditorComponent, SfdtExportService, SelectionService, EditorService,
EditorHistoryService } from '@syncfusion/ej2-angular-documenteditor';

@Component({
  selector: 'app-container',
  //specifies the template string for the Document Editor component
  template: `<ejs-documenteditor #document_editor id="container" style="width: 100%;height:
100%;display:block" [isReadOnly]=false [enableSelection]=true [enableEditor]=true
[enableEditorHistory]=true >
</ejs-documenteditor>`,
  encapsulation: ViewEncapsulation.None,
  //Inject require modules.
  providers: [SfdtExportService, SelectionService, EditorService, EditorHistoryService]
})

export class AppComponent {
}
```

`

You can enable or disable history preservation for a document editor instance any time using the 'enableEditorHistory' property. Refer to the following sample code.

```
`typescript
this.documentEditor.enableEditorHistory = false;
```

`

Undo and redo

You can perform undo and redo by **CTRL+Z** and **CTRL+Y** keyboard shortcuts. Document Editor exposes API to do it programmatically.

To undo the last editing operation in document editor, refer to the following sample code.

```
`typescript
this.documentEditor.editorHistory.undo();
```

`

To redo the last undone action, refer to the following code example.

```
`typescript
this.documentEditor.editorHistory.redo();
```

`

Stack size

History of editing actions will be maintained in stack, so that the last item will be reverted first. By default, document editor limits the size of undo and redo stacks to 500 each respectively. However, you can customize this limit. Refer to the following sample code.

```
`typescript
//Set undo limit.
this.documentEditor.editorHistory.undoLimit = 400;
//Set redo limit.
this.documentEditor.editorHistory.redoLimit = 400;
```

`

See Also

- [Feature modules](#)
- [Keyboard shortcuts](#)

Find and replace in Angular Document editor component

The document editor component searches a portion of text in the document through a built-in interface called **OptionsPane** or rich APIs. When used in combination with selection performs various operations on the search results like replacing it with some other text, highlighting it, making it bolder, and more.

Options pane

This provides the options to search for a portion of text in the document. After search operation is completed, the search results will be displayed in a list and options to navigate between them. The current occurrence of matched text or all occurrences with another text can be replaced by switching to **Replace** tab. This pane is opened using the keyboard shortcut **CTRL+F**. You can also open it programmatically using the following sample code.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { DocumentEditorAllModule } from '@syncfusion/ej2-angular-documenteditor'
import { ToolbarModule } from '@syncfusion/ej2-angular-navigations'
import { ComboBoxModule } from '@syncfusion/ej2-angular-dropdowns'
import { ColorPickerModule } from '@syncfusion/ej2-angular-inputs'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import {
    DocumentEditorComponent, SelectionService, EditorService, SearchService,
    OptionsPaneService
} from '@syncfusion/ej2-angular-documenteditor';
@Component({
    imports: [

        ButtonModule,
        ToolbarModule,
        DocumentEditorAllModule,
        ComboBoxModule,
        ColorPickerModule
    ],
    standalone: true,
    selector: 'app-container',
    //specifies the template string for the Document Editor component
    template: `<div>
        <button ej2-button (click)="showOptionsPane()" >Show</button>
        <ejs-documenteditor #document_editor height="330px"
style="display:block" id="container" height="330px" style="display:block"
[enableSelection]=true [enableSearch]=true [enableEditor]=true
[isReadOnly]=false [enableOptionsPane]=true (created)="onCreated()"></ejs-
documenteditor>
        </div>`,
    encapsulation: ViewEncapsulation.None,
    providers: [SelectionService, EditorService, SearchService,
OptionsPaneService]
})
export class AppComponent {
    @ViewChild('document_editor')
    public documentEditor?: DocumentEditorComponent;
    onCreated(): void {
        if ((this.documentEditor as
DocumentEditorComponent).isDocumentLoaded) {
            let sfdt: string = `{
                "sections": [
                    {
                        "blocks": [
```

```

        {
            "inlines": [
                {
                    "characterFormat": {
                        "bold": true,
                        "italic": true
                    },
                    "text": "Adventure Works Cycles, the
fictitious company on which the AdventureWorks sample databases are based,
is a large, multinational manufacturing company. The company manufactures
and sells metal and composite bicycles to North American, European and Asian
commercial markets. While its base operation is located in Bothell,
Washington with 290 employees, several regional sales teams are located
throughout their market base."
                }
            ]
        }
    ]
}
};
//Open the document in Document Editor
(this.documentEditor as DocumentEditorComponent).open(sfdd);
}
}
public showOptionsPane(): void {
    //Open options pane.
    (this.documentEditor as DocumentEditorComponent).showOptionsPane();
}
}

```

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 close the options pane by pressing **Esc** key.

Search

The [Search](#) module of Document Editor exposes the following APIs:

API Name	Type	Description
getWeather	GET	Retrieve weather information for a specific location.
getForecast	GET	Retrieve weather forecast for a specific location.
getHistory	GET	Retrieve weather history for a specific location.
getAlerts	GET	Retrieve weather alerts for a specific location.
getRadar	GET	Retrieve weather radar data for a specific location.
getSatellite	GET	Retrieve weather satellite data for a specific location.
getAirQuality	GET	Retrieve air quality data for a specific location.
getPollution	GET	Retrieve pollution data for a specific location.
getTraffic	GET	Retrieve traffic data for a specific location.
getEvents	GET	Retrieve events data for a specific location.
getNews	GET	Retrieve news data for a specific location.
getSports	GET	Retrieve sports data for a specific location.
getFinance	GET	Retrieve finance data for a specific location.
getHealth	GET	Retrieve health data for a specific location.
getEducation	GET	Retrieve education data for a specific location.
getRealEstate	GET	Retrieve real estate data for a specific location.
getJobs	GET	Retrieve jobs data for a specific location.
getTravel	GET	Retrieve travel data for a specific location.
getFood	GET	Retrieve food data for a specific location.
getDrinks	GET	Retrieve drinks data for a specific location.
getGyms	GET	Retrieve gyms data for a specific location.
getParks	GET	Retrieve parks data for a specific location.
getMuseums	GET	Retrieve museums data for a specific location.
getTheaters	GET	Retrieve theaters data for a specific location.
getConcerts	GET	Retrieve concerts data for a specific location.
getFestivals	GET	Retrieve festivals data for a specific location.
getSportsEvents	GET	Retrieve sports events data for a specific location.
getConferences	GET	Retrieve conferences data for a specific location.
getMeetings	GET	Retrieve meetings data for a specific location.
getWebinars	GET	Retrieve webinars data for a specific location.
getPodcasts	GET	Retrieve podcasts data for a specific location.
getBooks	GET	Retrieve books data for a specific location.
getMovies	GET	Retrieve movies data for a specific location.
getTVShows	GET	Retrieve TV shows data for a specific location.
getMusic	GET	Retrieve music data for a specific location.
getArt	GET	Retrieve art data for a specific location.
getFashion	GET	Retrieve fashion data for a specific location.
getBeauty	GET	Retrieve beauty data for a specific location.
getHealthcare	GET	Retrieve healthcare data for a specific location.
getEducation2	GET	Retrieve education data for a specific location.
getRealEstate2	GET	Retrieve real estate data for a specific location.
getJobs2	GET	Retrieve jobs data for a specific location.
getTravel2	GET	Retrieve travel data for a specific location.
getFood2	GET	Retrieve food data for a specific location.
getDrinks2	GET	Retrieve drinks data for a specific location.
getGyms2	GET	Retrieve gyms data for a specific location.
getParks2	GET	Retrieve parks data for a specific location.
getMuseums2	GET	Retrieve museums data for a specific location.
getTheaters2	GET	Retrieve theaters data for a specific location.
getConcerts2	GET	Retrieve concerts data for a specific location.
getFestivals2	GET	Retrieve festivals data for a specific location.
getSportsEvents2	GET	Retrieve sports events data for a specific location.
getConferences2	GET	Retrieve conferences data for a specific location.
getMeetings2	GET	Retrieve meetings data for a specific location.
getWebinars2	GET	Retrieve webinars data for a specific location.
getPodcasts2	GET	Retrieve podcasts data for a specific location.
getBooks2	GET	Retrieve books data for a specific location.
getMovies2	GET	Retrieve movies data for a specific location.
getTVShows2	GET	Retrieve TV shows data for a specific location.
getMusic2	GET	Retrieve music data for a specific location.
getArt2	GET	Retrieve art data for a specific location.
getFashion2	GET	Retrieve fashion data for a specific location.
getBeauty2	GET	Retrieve beauty data for a specific location.
getHealthcare2	GET	Retrieve healthcare data for a specific location.
getEducation3	GET	Retrieve education data for a specific location.
getRealEstate3	GET	Retrieve real estate data for a specific location.
getJobs3	GET	Retrieve jobs data for a specific location.
getTravel3	GET	Retrieve travel data for a specific location.
getFood3	GET	Retrieve food data for a specific location.
getDrinks3	GET	Retrieve drinks data for a specific location.
getGyms3	GET	Retrieve gyms data for a specific location.
getParks3	GET	Retrieve parks data for a specific location.
getMuseums3	GET	Retrieve museums data for a specific location.
getTheaters3	GET	Retrieve theaters data for a specific location.
getConcerts3	GET	Retrieve concerts data for a specific location.
getFestivals3	GET	Retrieve festivals data for a specific location.
getSportsEvents3	GET	Retrieve sports events data for a specific location.
getConferences3	GET	Retrieve conferences data for a specific location.
getMeetings3	GET	Retrieve meetings data for a specific location.
getWebinars3	GET	Retrieve webinars data for a specific location.
getPodcasts3	GET	Retrieve podcasts data for a specific location.
getBooks3	GET	Retrieve books data for a specific location.
getMovies3	GET	Retrieve movies data for a specific location.
getTVShows3	GET	Retrieve TV shows data for a specific location.
getMusic3	GET	Retrieve music data for a specific location.
getArt3	GET	Retrieve art data for a specific location.
getFashion3	GET	Retrieve fashion data for a specific location.
getBeauty3	GET	Retrieve beauty data for a specific location.
getHealthcare3	GET	Retrieve healthcare data for a specific location.
getEducation4	GET	Retrieve education data for a specific location.
getRealEstate4	GET	Retrieve real estate data for a specific location.
getJobs4	GET	Retrieve jobs data for a specific location.
getTravel4	GET	Retrieve travel data for a specific location.
getFood4	GET	Retrieve food data for a specific location.
getDrinks4	GET	Retrieve drinks data for a specific location.
getGyms4	GET	Retrieve gyms data for a specific location.
getParks4	GET	Retrieve parks data for a specific location.
getMuseums4	GET	Retrieve museums data for a specific location.
getTheaters4	GET	Retrieve theaters data for a specific location.
getConcerts4	GET	Retrieve concerts data for a specific location.
getFestivals4	GET	Retrieve festivals data for a specific location.
getSportsEvents4	GET	Retrieve sports events data for a specific location.
getConferences4	GET	Retrieve conferences data for a specific location.
getMeetings4	GET	Retrieve meetings data for a specific location.
getWebinars4	GET	Retrieve webinars data for a specific location.
getPodcasts4	GET	Retrieve podcasts data for a specific location.
getBooks4	GET	Retrieve books data for a specific location.
getMovies4	GET	Retrieve movies data for a specific location.
getTVShows4	GET	Retrieve TV shows data for a specific location.
getMusic4	GET	Retrieve music data for a specific location.
getArt4	GET	Retrieve art data for a specific location.

<code>findAll()</code>	Method	Searches for specified text in the whole document and highlights it with yellow.
------------------------	--------	----------------------------------------------------------------------------------

searchResults	Property	This is an instance of SearchResults.
---------------	----------	---------------------------------------

[find\(\)](#) | Method | Find immediate occurrence of specified text from cursor position in the document and highlights it with yellow.

Find the immediate occurrence in the document

Using [find\(\)](#) method, you can find the immediate occurrence of specified text from current cursor position in the document.

The following example code illustrates how to use find in Document editor.

```
`typescript
this.documenteditor.search.find('Some text', 'None');
`
```

Note: Second parameter is optional parameter and it denotes find Options. Possible values of find options are 'None' | 'WholeWord' | 'CaseSensitive' | 'CaseSensitiveWholeWord'.

Find all the occurrences in the document

Using [findAll\(\)](#) method, you can find all the occurrences of specified text in the whole document and highlight it with yellow.

The following example code illustrates how to find All the text in the document.

```
`typescript
this.documenteditor.search.findAll('Some text', 'None');
`
```

Note: Second parameter is optional parameter and it denotes find Options. Possible values of find options are 'None' | 'WholeWord' | 'CaseSensitive' | 'CaseSensitiveWholeWord'.

Search results

The [SearchResults](#) class provides information about the search results after a search operation is completed that can be identified using the [searchResultsChange](#) event. This will expose the following APIs:

API Name	Type	Description
----------	------	-------------

---	---	---
-----	-----	-----

length	Property	Returns the total number of results found on the search.
------------------------	----------	----------------------------------------------------------

index	Property	Returns the index of selected search result. You can change the value for this property to move the selection.
-----------------------	----------	----------------------------------------------------------------------------------------------------------------

replaceAll()	Method	Replaces all the occurrences with specified text.
------------------------------	--------	---------------------------------------------------

clear()	Method	Clears the search result.
-------------------------	--------	---------------------------

Replace all the occurrences

Using [replaceAll](#), you can replace all the occurrences with specified text.

The following example code illustrates how to use replace All in Document editor.

```
`typescript
this.documentEditor.search.findAll ('Some text');
// Replace all the searched text with word 'Mike'
this.documentEditor.search.searchResults.replaceAll("Mike");
`
```

Replace

Using [insertText](#), you can replace the current searched text with specified text and it replace single occurrence.

Note: This [insertText](#) API accepts following control characters

- * New line characters ("r", "r\n", "n") - Inserts a new paragraph and appends the remaining text to the new paragraph.
 - * Line break character ("v") - Moves the remaining text to start in new line.
 - * Tab character ("t") - Allocates a tab space and continue the next character.
-

The following example code illustrates how to find a text in the document and replace each occurrence of the text one by one programmatically.

```
`typescript
this.container.documentEditor.search.findAll('works');
let searchLength: number = this.container.documentEditor.search.searchResults.length;
for (let i = 0; i < searchLength; i++) {
  // It will move selection to specific searched index,move to each occurrence one by one
  this.container.documentEditor.search.searchResults.index = i;
  // Replace it with some text
  this.container.documentEditor.editor.insertText('Hello');
}
this.container.documentEditor.search.searchResults.clear();
```

SearchResultsChange event

[DocumentEditor](#) exposes the [searchResultsChange](#) event that will be triggered whenever search results are changed. Consider the following scenarios:

- A search operation is completed with some results.
- The results are replaced with some other text, since it will be cleared automatically.
- The results are cleared explicitly.

Refer to the following code example.

```
`typescript
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import {
  DocumentEditorComponent, EditorService, SelectionService, SfdtExportService, EditorHistoryService,
  BookmarkDialogService
} from '@syncfusion/ej2-angular-documenteditor';
```

```

@Component({
  selector: 'app-container',
  //specifies the template string for the Document Editor component
  template: <ejs-documenteditor #document_editor id="container" height="330px"
  style="display:block" [isReadOnly]=false [enableSelection]=true [enableSearch]=true
  (searchResultsChange)="onSearchResultChange()" > </ejs-documenteditor>,
  encapsulation: ViewEncapsulation.None,
  providers: [EditorService, SelectionService, SfdtExportService]
})
export class AppComponent {
  @ViewChild('document_editor')
  public documentEditor: DocumentEditorComponent;
  // search result change event handler
  public onSearchResultChange(): void {
  }
}

```

Customize find and replace

Using the exposed APIs, you can customize the find and replace functionality in your application. Refer to the following sample code.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { DocumentEditorAllModule } from '@syncfusion/ej2-angular-documenteditor'
import { ToolbarModule } from '@syncfusion/ej2-angular-navigations'
import { ComboBoxModule } from '@syncfusion/ej2-angular-dropdowns'
import { ColorPickerModule } from '@syncfusion/ej2-angular-inputs'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import {
  DocumentEditorComponent, SelectionService, EditorService, SearchService,
  OptionsPaneService
} from '@syncfusion/ej2-angular-documenteditor';
@Component({
  imports: [
    ButtonModule,
    ToolbarModule,
    DocumentEditorAllModule,
    ComboBoxModule,
    ColorPickerModule
  ],

```

```

standalone: true,
selector: 'app-container',
//specifies the template string for the Document Editor component
template: `<div style="width:100%;">
<table>
  <tr>
    <td>
      <label>Text to find:</label>
    </td>
    <td>
      <input type="text" id="find_text" />
    </td>
  </tr>
  <tr>
    <td>
      <label>Text to replace:</label>
    </td>
    <td>
      <input type="text" id="replace_text" />
    </td>
  </tr>
  <tr>
    <td colspan="2">
      <button ejb-button (click)="onReplaceButtonClick()"
>Replace all</button>
    </td>
  </tr>
</table>
  <ejb-documenteditor #document_editor id="container" height="330px"
style="display:block" [enableSelection]=true [enableSearch]=true
[enableEditor]=true [isReadOnly]=false (created)="onCreated()"></ejb-
documenteditor>
  </div>`,
encapsulation: ViewEncapsulation.None,
providers: [SelectionService, EditorService, SearchService]
})
export class AppComponent {
  @ViewChild('document_editor')
  public documentEditor?: DocumentEditorComponent;
  onCreated(): void {
    if ((this.documentEditor as
DocumentEditorComponent).isDocumentLoaded) {
      let sfdt: string = `{
        "sections": [
          {
            "blocks": [
              {
                "inlines": [
                  {
                    "characterFormat": {
                      "bold": true,
                      "italic": true
                    },
                    "text": "Adventure Works Cycles, the
fictitious company on which the AdventureWorks sample databases are based,
is a large, multinational manufacturing company. The company manufactures
and sells metal and composite bicycles to North American, European and Asian

```



```
commercial markets. While its base operation is located in Bothell,
Washington with 290 employees, several regional sales teams are located
throughout their market base."
    }
    ]
    }
    ]
    }
    ];
    (this.documentEditor as DocumentEditorComponent).open(sfdd);
    }
    public onReplaceButtonClick(): void {
        let textToFind: string = (document.getElementById('find_text') as
HTMLInputElement).value;
        let textToReplace: string = (document.getElementById('replace_text')
as HTMLInputElement).value;
        if (textToFind !== '') {
            // Find all the occurrences of given text
            (this.documentEditor as
DocumentEditorComponent).searchModule.findAll(textToFind);
            if ((this.documentEditor as
DocumentEditorComponent).searchModule.searchResults.length > 0) {
                // Replace all the occurrences of given text
                (this.documentEditor as
DocumentEditorComponent).searchModule.searchResults.replaceAll(textToReplace
);
            }
        }
    }
}
```

MAIN.TS

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

Keyboard shortcut in Angular Document editor component

Text formatting

The following table lists the default keyboard shortcuts in document editor for formatting text:

Key combination	Description
----- -----	
Ctrl + B	Toggles the bold property of selected text.
Ctrl + I	Toggles the italic property of selected text.
Ctrl + U	Toggles the underline property of selected text.
Ctrl + +	Toggles the subscript formatting of selected text.

|Ctrl + Shift + + | Toggles the superscript formatting of selected contents. |

| Ctrl + } | Increases the actual font size of selected text by one point. |

| Ctrl + { | Decreases the actual font size of selected text by one point. |

Paragraph formatting

The following table lists the default keyboard shortcuts for formatting the paragraph:

Key combination	Description
-----------------	-------------

-----	-----
-------	-------

Ctrl + E	Selected paragraphs are center aligned.
----------	-----------------------------------------

Ctrl + J	Selected paragraphs are justified.
----------	------------------------------------

Ctrl + L	Selected paragraphs are left aligned.
----------	---------------------------------------

Ctrl + R	Selected paragraphs are right aligned.
----------	----------------------------------------

Ctrl + 1	Single line spacing is applied for selected paragraphs.
----------	---------------------------------------------------------

Ctrl + 5	1.5 line spacing is applied for selected paragraphs.
----------	------------------------------------------------------

Ctrl + 2	Double spacing is applied for selected paragraphs.
----------	----------------------------------------------------

Ctrl + 0	No spacing is applied before the selected paragraphs.
----------	-------------------------------------------------------

Ctrl + M	Increases the left indent of selected paragraphs by a factor of 36 points.
----------	----------------------------------------------------------------------------

Ctrl + Shift + M	Decreases the left indent of selected paragraphs by a factor of 36 points.
------------------	----------------------------------------------------------------------------

Ctrl + *	Show/Hide the hidden characters like spaces, tab, paragraph marks, and breaks.
----------	--------------------------------------------------------------------------------

Clipboard

Key Combination	Description
-----------------	-------------

-----	-----
-------	-------

Ctrl + C	Copies selected contents to the clipboard.
----------	--------------------------------------------

Ctrl + V	Pastes plain text content from the clipboard.
----------	-----------------------------------------------

Ctrl + X	Moves selected content to the clipboard.
----------	------------------------------------------

Keyboard shortcut to navigate around the document

Key Combination	Description
-----------------	-------------

-----	-----
-------	-------

Left arrow	Moves the cursor position one character to the left.
------------	------------------------------------------------------

Right arrow	Moves the cursor position one character to the right.
-------------	-------------------------------------------------------

Down arrow	Moves the cursor position down one line.
------------	------------------------------------------

Up arrow	Moves the cursor position up one line.
----------	----------------------------------------

Ctrl + Left arrow	Moves the cursor position one word to the left.
-------------------	-------------------------------------------------

Ctrl + Right arrow	Moves the cursor position one word to the right.
--------------------	--------------------------------------------------

Ctrl + Up arrow	Moves the cursor position one paragraph up.
-----------------	---------------------------------------------

|Ctrl + Down arrow| Moves the cursor position one paragraph down. |

|Tab (in table)| Moves the cursor position one cell to the right. |

|Shift + Tab (in table)| Moves the cursor position one cell to the left. |

|Home| Moves the cursor position to the start of a line. |

|End| Moves the cursor position to the end of a line. |

|Page up| Moves the cursor position one screen up. |

|Page down| Moves the cursor position one screen down. |

|Ctrl + Home| Moves the cursor position to the start of a document. |

|Ctrl + End| Moves the cursor position to the end of a document. |

Keyboard shortcut to extend selection

Key Combination	Description
-----	-----
Shift + Left arrow	Extends selection one character to the left.
Shift + Right arrow	Extends selection one character to the right.
Shift + Down arrow	Extends selection one line downward.
Shift + Up arrow	Extends selection one line upward.
Shift + Home	Extends selection to the start of a line.
Shift + End	Extends Selection to the end of a line.
Ctrl + A	Extends selection to the entire document.
Ctrl + Shift + Left arrow	Extends selection one word to the left.
Ctrl + Shift + Right arrow	Extends selection one word to the right.
Ctrl + Shift + Down arrow	Extends selection to the end of a paragraph.
Ctrl + Shift + Up arrow	Extends selection to the start of a paragraph.
Ctrl + Shift + Home	Extends selection to the start of a document.
Ctrl + Shift + End	Extends selection to the end of a document.

Find and Replace

Create, Save and Print document

|Ctrl + S| Saves the document in SFDt format.|

|Ctrl + P| Prints the document.|

Edit Operation

Key Combination	Description
----- -----	
Backspace	Deletes one character to the left.
Delete	Deletes one character to the right.
Ctrl + Z	Undo last performed action.
Ctrl + Y	Redo last undo action.

Insert special characters

Key Combination	Description
----- -----	
Ctrl + Enter	Inserts page break.
Shift + Enter	Inserts line break.

Dialog

Key Combination	Description
----- -----	
Ctrl + F	Opens options pane.
Ctrl + D	Opens font dialog.
Ctrl + K	Opens hyperlink dialog.

See Also

- [How to override the keyboard shortcuts](#)

Scrolling zooming in Angular Document editor component

The Document Editor renders the document as page by page. You can scroll through the pages by mouse wheel or touch interactions. You can also scroll through the page by using 'scrollToPage()' method of document editor instance. Refer to the following code example.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { DocumentEditorAllModule } from '@syncfusion/ej2-angular-documenteditor'
import { ToolbarModule } from '@syncfusion/ej2-angular-navigations'
import { ComboBoxModule } from '@syncfusion/ej2-angular-dropdowns'
import { ColorPickerModule } from '@syncfusion/ej2-angular-inputs'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import {
    DocumentEditorComponent
```

```

} from '@syncfusion/ej2-angular-documenteditor';
@Component({
  imports: [

    ButtonModule,
    ToolbarModule,
    DocumentEditorAllModule,
    ComboBoxModule,
    ColorPickerModule

  ],
  standalone: true,
  selector: 'app-container',
  template: `<div>
    <ejs-documenteditor #document_editor height="330px"
style="display:block" (created)="onCreated()" "></ejs-documenteditor>
    </div>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild('document_editor')
  public documentEditor?: DocumentEditorComponent;
  onCreated(): void {
    if ((this.documentEditor as
DocumentEditorComponent).isDocumentLoaded) {
      let sfdt: any = {
        "sections": [
          {
            "blocks": [
              {
                "paragraphFormat": {
                  "styleName": "Normal"
                },
                "inlines": [
                  {
                    "text": "First page"
                  }
                ]
              }
            ],
            "headersFooters": {},
          },
          {
            "blocks": [
              {
                "paragraphFormat": {
                  "styleName": "Normal"
                },
                "inlines": [
                  {
                    "text": "Second page"
                  }
                ]
              }
            ],
            "headersFooters": {},
          }
        ],
      }
    }
  },
}

```

```

        "characterFormat": {},
        "paragraphFormat": {},
        "background": {
            "color": "#FFFFFF"
        },
        "styles": [
            {
                "type": "Paragraph",
                "name": "Normal",
                "next": "Normal"
            },
            {
                "type": "Character",
                "name": "Default Paragraph Font"
            }
        ]
    };
    //Open the default document in Document Editor.
    (this.documentEditor as
DocumentEditorComponent).open(JSON.stringify(sfdt));
    //Scroll to specified page number.
    (this.documentEditor as
DocumentEditorComponent).scrollToPage(2);
    }
}
}

```

MAIN.TS

```

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

```

Calling this method brings the specified page into view but doesn't move selection. Hence this method will work by default. That is, it works even if selection is not enabled.

In case, if you wish to move the selection to any page in document editor and bring it into view, you can use 'goToPage()' method of selection instance. Refer to the following code example.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { DocumentEditorAllModule } from '@syncfusion/ej2-angular-documenteditor'
import { ToolbarModule } from '@syncfusion/ej2-angular-navigations'
import { ComboBoxModule } from '@syncfusion/ej2-angular-dropdowns'
import { ColorPickerModule } from '@syncfusion/ej2-angular-inputs'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import {
    DocumentEditorComponent, SelectionService
} from '@syncfusion/ej2-angular-documenteditor';
@Component({
    imports: [

```

```

        ButtonModule,
        ToolbarModule,
        DocumentEditorAllModule,
        ComboBoxModule,
        ColorPickerModule
    ],
    standalone: true,
    selector: 'app-container',
    template: `<div>
        <ejs-documenteditor #document_editor height="330px"
style="display:block" [enableSelection]=true (created)="onCreated()"></ejs-
documenteditor>
        </div>`,
    encapsulation: ViewEncapsulation.None,
    providers: [SelectionService]
})
export class AppComponent {
    @ViewChild('document_editor')
    public documentEditor?: DocumentEditorComponent;
    onCreated(): void {
        if ((this.documentEditor as
DocumentEditorComponent).isDocumentLoaded) {
            let sfdd: string = `{
                "sections": [
                    {
                        "blocks": [
                            {
                                "paragraphFormat": {
                                    "styleName": "Normal"
                                },
                                "inlines": [
                                    {
                                        "text": "First page"
                                    }
                                ]
                            }
                        ],
                        "headersFooters": {},
                    },
                    {
                        "blocks": [
                            {
                                "paragraphFormat": {
                                    "styleName": "Normal"
                                },
                                "inlines": [
                                    {
                                        "text": "Second page"
                                    }
                                ]
                            }
                        ],
                        "headersFooters": {},
                    }
                ],
                "characterFormat": {},
            `;
        }
    }
}

```

```

        "paragraphFormat": {},
        "background": {
            "color": "#FFFFFF"
        },
        "styles": [
            {
                "type": "Paragraph",
                "name": "Normal",
                "next": "Normal"
            },
            {
                "type": "Character",
                "name": "Default Paragraph Font"
            }
        ]
    };
    //Open the document in Document Editor.
    (this.documentEditor as
    DocumentEditorComponent).open(JSON.stringify(sfdt));
    //Navigate to specified page number.
    (this.documentEditor as
    DocumentEditorComponent).selection.goToPage(2);
    }
}

```

MAIN.TS

```

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

```

Zooming

You can scale the contents in document editor ranging from 10% to 500% of the actual size. You can achieve this using mouse or touch interactions. You can also use `zoomFactor` property of document editor instance. The value can be specified in a range from 0.1 to 5. Refer to the following code example.

`typescript

```
this.documentEditor.zoomFactor = 3;
```

,

Page Fit Type

Apart from specifying the zoom factor as value, the Document Editor provides option to specify page fit options such as fit to full page or fit to page width. You can set this option using 'fitPage' method of document editor instance. Refer to the following code example.

`typescript

```
this.documentEditor.fitPage('Fit page width');
```

,

Zoom option using UI

The following code example shows how to provide zoom options in document editor.

```
`typescript
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import {
  DocumentEditorComponent
} from '@syncfusion/ej2-angular-documenteditor';
@Component({
  selector: 'app-container',
  template: `<div>
    <ejs-documenteditor #documenteditor_editor (selectionChange)='onSelectionChange()'
    (viewChange)='onViewChange($event)' (documentChange)='onDocumentChange()'
    (created)='onCreated()' [enableSelection]=true [isReadOnly]=false height="330px"
    style="display:block"></ejs-documenteditor>
  </div>
  <div id="documenteditor_statusbar">
    <label style="margin-top: 6px;margin-right: 2px">Page </label>
    <div class="single-line e-de-pagenumbers-text" (keydown)='pageKeydownEvent($event)'
    (click)='pagerClickEvent($event)' id="editablePageNumber" style="font-size:12px;font-weight:
    700;display: inline-flex;height: 17px;padding: 0px 4px;" contenteditable="false">
    <label id="documenteditorpageno" style="text-transform:capitalize;white-
    space:pre;overflow:hidden;user-select:none;cursor:text;height:17px;max-
    width:150px">{{currentPage}}</label>
  </div>
  <label id="documenteditorpagecountseparator" style="margin-left:2px;letter-spacing:
  1.05px">of</label>
  <label id="documenteditor_pagecount" style="margin-left:6px;letter-spacing:
  1.05px">{{pageCount}}</label>
  <button ej2-dropdownbutton class="e-de-statusbar-zoom" [items]="zoomItems"
  [content]="zoomContent" (select)='onZoom($event)' iconCss="e-ddb-icons e-profile"></button>
</div>
`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild('document_editor')
```

```
public documentEditor: DocumentEditorComponent;
public zoomContent: string = "100%";
public zoomItems = [
  {
    text: '150%',
  },
  {
    text: '125%',
  },
  {
    text: '100%',
  },
  {
    text: '75%',
  },
  {
    text: '50%',
  },
  {
    separator: true
  },
  {
    text: 'Fit one page'
  },
  {
    text: 'Fit page width',
  }
];
public pageCount: number = 1;
public currentPage: number = 1;
onCreated(): void {
  if (this.documentEditor.isDocumentLoaded) {
    let sfdt: string = `{
  "sections": [
```

```
{
  "blocks": [
    {
      "paragraphFormat": {
        "styleName": "Normal"
      },
      "inlines": [
        {
          "text": "First page"
        }
      ]
    },
    {
      "headersFooters": {},
    },
    {
      "blocks": [
        {
          "paragraphFormat": {
            "styleName": "Normal"
          },
          "inlines": [
            {
              "text": "Second page"
            }
          ]
        }
      ],
      "headersFooters": {},
    },
    {
      "blocks": [
        {
```

```
"paragraphFormat": {
  "styleName": "Normal"
},
"inlines": [
  {
    "text": "Third page"
  }
]
},
"headersFooters": {},
},
"characterFormat": {},
"paragraphFormat": {},
"background": {
  "color": "#FFFFFF"
},
"styles": [
  {
    "type": "Paragraph",
    "name": "Normal",
    "next": "Normal"
  },
  {
    "type": "Character",
    "name": "Default Paragraph Font"
  }
]
}';
//Open the default document.
this.documentEditor.open(JSON.stringify(sfdd));
}
```

```
}  
public onViewChange(args: any) {  
  //Set current page number in status bar.  
  this.currentPage = args.startPage;  
}  
public onSelectionChange(args: any) {  
  ///Set current page number on selection change.  
  this.currentPage = this.documentEditor.selection.startPage;  
}  
public onDocumentChange() {  
  //Get page count.  
  this.pageCount = this.documentEditor.pageCount  
  this.zoomContent = Math.round(this.documentEditor.zoomFactor * 100) + '%';  
}  
public onZoom(args: any) {  
  let zoomValue = args.item.text;  
  if (zoomValue.match('Fit one page')) {  
    this.documentEditor.fitPage('FitOnePage');  
  } else if (zoomValue.match('Fit page width')) {  
    this.documentEditor.fitPage('FitPageWidth');  
  } else {  
    //Set zoom factor.  
    this.documentEditor.zoomFactor = parseInt(zoomValue, 0) / 100;  
  }  
  this.zoomContent = Math.round(this.documentEditor.zoomFactor * 100) + '%';  
}  
public pageKeydownEvent(e: any) {  
  if (e.which === 13) {  
    e.preventDefault();  
    let pageNumber = parseInt(document.getElementById("editablePageNumber").textContent, 0);  
    if (pageNumber > this.documentEditor.pageCount) {  
      this.updatePageNumber();  
    } else {
```

```
//Navigate to specified page number.
this.documentEditor.selection.goToPage(parseInt(document.getElementById("editablePageNumber").textContent, 0));
}
document.getElementById("editablePageNumber").contentEditable = 'false';
if (document.getElementById("editablePageNumber").textContent === "") {
this.updatePageNumber();
}
}
if (e.which > 64) {
e.preventDefault();
}
}
public pageBlurEvent() {
if (document.getElementById("editablePageNumber").textContent === "" ||
parseInt(document.getElementById("editablePageNumber").textContent, 0) >
this.documentEditor.pageCountt) {
this.updatePageNumber();
}
document.getElementById("editablePageNumber").contentEditable = 'false';
}
public pagerClickEvent() {
document.getElementById("editablePageNumber").contentEditable = 'true';
document.getElementById("editablePageNumber").focus();
window.getSelection().selectAllChildren(document.getElementById("editablePageNumber"));
}
public updatePageNumber() {
//Update current page number.
this.currentPage = this.documentEditor.selection.startPage.toString();
}
}
`
```

Print in Angular Document editor component

To print the document, use the [print](#) method from document editor instance.

Refer to the following example for showing a document and print it.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { DocumentEditorAllModule } from '@syncfusion/ej2-angular-documenteditor'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import {
    DocumentEditorComponent, PrintService
} from '@syncfusion/ej2-angular-documenteditor';
@Component({
    imports: [

        ButtonModule,
        DocumentEditorAllModule
    ],
    standalone: true,
    selector: 'app-container',
    //specifies the template string for the Document Editor component
    template: `<div>
        <button ej2-button (click)="onPrint()" >Print</button>
        <ejs-documenteditor #document_editor height="330px"
style="display:block" [enablePrint]=true (created)="onCreated()"></ejs-
documenteditor>
        </div>`,
    encapsulation: ViewEncapsulation.None,
    providers: [PrintService]
})
export class AppComponent {
    @ViewChild('document_editor')
    public documentEditor?: DocumentEditorComponent;
    onCreated(): void {
        if ((this.documentEditor as
DocumentEditorComponent).isDocumentLoaded) {
            let sfdt: string = `{
                "sections": [
                    {
                        "blocks": [
                            {
                                "inlines": [
                                    {
                                        "characterFormat": {
                                            "bold": true,
                                            "italic": true
                                        },
                                        "text": "Hello World"
                                    }
                                ]
                            }
                        ]
                    },
                    "headersFooters": {
                    }
                }
            }`
        }
    }
}
```

```

        }`;
        //Open the default document.
        (this.documentEditor as DocumentEditorComponent).open(sfdt);
    }
}
public onPrint(): void {
    //Print the document content.
    (this.documentEditor as DocumentEditorComponent).print();
}
}

```

MAIN.TS

```

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

```

Refer to the following example for creating a document and print it.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { DocumentEditorAllModule } from '@syncfusion/ej2-angular-documenteditor'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import {
    DocumentEditorComponent, PrintService, SelectionService, EditorService,
    EditorHistoryService
} from '@syncfusion/ej2-angular-documenteditor';
@Component({
    imports: [
        ButtonModule,
        DocumentEditorAllModule
    ],
    standalone: true,
    selector: 'app-container',
    template: `<div>
        <button ejs-button (click)="onPrint()" >Print</button>
        <ejs-documenteditor #document_editor height="330px"
style="display:block" [isReadOnly]=true [enableSelection]=true
[enableEditor]=true [enablePrint]=true (created)="onCreated()"></ejs-
documenteditor>
        </div>`,
    encapsulation: ViewEncapsulation.None,
    providers: [PrintService, SelectionService, EditorService,
    EditorHistoryService]
})
export class AppComponent {
    @ViewChild('document_editor')
    public documentEditor?: DocumentEditorComponent;
    onCreated(): void {

```



```

        if ((this.documentEditor as
DocumentEditorComponent).isDocumentLoaded) {
            let sfdt: string = `{
                "sections": [
                    {
                        "blocks": [
                            {
                                "inlines": [
                                    {
                                        "characterFormat": {
                                            "bold": true,
                                            "italic": true
                                        },
                                        "text": "Hello World"
                                    }
                                ]
                            }
                        ]
                    },
                    "headersFooters": {
                    }
                ]
            }`;
            //Open the document content.
            (this.documentEditor as DocumentEditorComponent).open(sfdt);
        }
        public onPrint(): void {
            //Print the document content.
            (this.documentEditor as DocumentEditorComponent).print();
        }
    }
}

```

MAIN.TS

```

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

```

Improve print quality

Document editor provides an option to improve the print quality using [printDevicePixelRatio](#) in Document editor settings. Document editor using canvas approach to render content. Then, canvas are converted to image and it process for print. Using printDevicePixelRatio API, you can increase the image quality based on your requirement.

The following example code illustrates how to improve the print quality in Document editor container.

`typescript

```

import { Component, OnInit } from '@angular/core';
import { ToolbarService } from '@syncfusion/ej2-angular-documenteditor';
@Component({

```

```

selector: 'app-root',
// specifies the template string for the DocumentEditorContainer component
template: <ejs-documenteditorcontainer id="container"
serviceUrl="https://ej2services.syncfusion.com/production/web-
services/api/documenteditor/" height="600px" style="display:block"
[documentEditorSettings]= "settings" [enableToolbar]=true> </ejs-documenteditorcontainer>,
providers: [ToolbarService]
})
export class AppComponent implements OnInit {
// Add required font families to list it in font drop down
public settings={printDevicePixelRatio :2};
ngOnInit(): void {
}
}
`

```

Note: By default, printDevicePixelRatio value is 1.

Print using window object

You can print the document in document editor by passing the window instance. This is useful to implement print in third party frameworks such as electron, where the window instance will not be available. Refer to the following example.

```

`typescript
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import {
  DocumentEditorComponent, PrintService, SelectionService, EditorService, EditorHistoryService
} from '@syncfusion/ej2-angular-documenteditor';
@Component({
  selector: 'app-container',
  template: `<div>
<button ej2-button (click)="onPrint()" >Print</button>
<ejs-documenteditor #document_editor height="330px" style="display:block" [enablePrint]=true></ejs-
documenteditor>
</div>`,
  encapsulation: ViewEncapsulation.None,
  providers: [PrintService]
})

```

```
export class AppComponent {
  @ViewChild('document_editor')
  public documentEditor: DocumentEditorComponent;
  public onPrint(): void {
    //Print the document content.
    this.documentEditor.print(window);
  }
}
```

Page setup

Some of the print options cannot be configured using JavaScript. Refer to the following links to learn more about the browser page setup:

- [Chrome](#)
- [Firefox](#)

However, you can customize margins, paper, and layout options by modifying the section format properties using page setup dialog

```
`typescript
this.documentEditor.showPageSetupDialog();
`
```

By customizing margins, papers, and layouts, the layout of the document will be changed in document editor. To modify these options during print operation, serialize the document as SFDT using the [serialize](#) method in document editor instance and open the SFDT data in another instance of document editor in separate window.

The following example shows how to customize layout options only for printing.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { DocumentEditorAllModule } from '@syncfusion/ej2-angular-documenteditor'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import {
  DocumentEditorComponent, PrintService, SelectionService, EditorService,
  EditorHistoryService, SfdtExportService
} from '@syncfusion/ej2-angular-documenteditor';
@Component({
  imports: [
    ButtonModule,
    DocumentEditorAllModule
  ],
```

```

standalone: true,
  selector: 'app-container',
  //specifies the template string for the Document Editor component
  template: `<div>
    <button ejs-button (click)="onPrint()" >Print</button>
    <ejs-documenteditor #document_editor height="330px"
style="display:block" [isReadOnly]=false [enableSelection]=true
[enableEditor]=true [enablePrint]=true [enableSfdtExport]=true></ejs-
documenteditor>
    <ejs-documenteditor #document_editor_preview height="330px"
style="display:block" [isReadOnly]=false [enableSelection]=true
[enableEditor]=true [enablePrint]=true [enableSfdtExport]=true></ejs-
documenteditor>
  </div>`,
  encapsulation: ViewEncapsulation.None,
  providers: [PrintService, SelectionService, EditorService,
EditorHistoryService, SfdtExportService]
})
export class AppComponent {
  @ViewChild('document_editor')
  public documentEditor?: DocumentEditorComponent;
  @ViewChild('document_editor_preview')
  public documentEditorPreview?: DocumentEditorComponent;
  public onPrint(): void {
    let sfdtData = (this.documentEditor as
DocumentEditorComponent).serialize();
    //Open the document in preview document editor.
    (this.documentEditorPreview as
DocumentEditorComponent).open(sfdtData);
    //Set A5 paper size
    (this.documentEditorPreview as
DocumentEditorComponent).selection.sectionFormat.pageWidth = 419.55;
    (this.documentEditorPreview as
DocumentEditorComponent).selection.sectionFormat.pageHeight = 595.30;
    //Print the document content.
    (this.documentEditorPreview as DocumentEditorComponent).print();
  }
}

```

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 Document editor component

Document Editor provides dialog support to major operations such as insert or edit hyperlink, formatting text, paragraph, style, list and table properties.

Font Dialog

Font dialog allows you to modify all text properties for selected contents at once such as bold, italic, underline, font size, font color, strikethrough, subscript and superscript.

Refer to the following example.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { DocumentEditorAllModule } from '@syncfusion/ej2-angular-documenteditor'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import {
    DocumentEditorComponent, SfdtExportService, SelectionService,
    FontDialogService, EditorService, ContextMenuService
} from '@syncfusion/ej2-angular-documenteditor';
@Component({
    imports: [

        ButtonModule,
        DocumentEditorAllModule
    ],
    standalone: true,
    selector: 'app-container',
    //specifies the template string for the Document Editor component
    template: `<div>
        <button ej2-button (click)="btnClick()" >Show Dialog</button>
        <ejs-documenteditor #document_editor id="container" height="330px"
        style="display:block" [isReadOnly]=false [enableSelection]=true
        [enableSfdtExport]=true [enableContextMenu]=true
        [enableFontDialog]=true [enableEditor]=true>
        </ejs-documenteditor>
        </div>`,
    encapsulation: ViewEncapsulation.None,
    providers: [SfdtExportService, SelectionService, FontDialogService,
    EditorService, ContextMenuService]
})
export class AppComponent {
    @ViewChild('document_editor')
    public documentEditor?: DocumentEditorComponent;
    public btnClick(): void {
        //Open font dialog
        (this.documentEditor as DocumentEditorComponent).showDialog('Font');
    }
}

```

MAIN.TS

```

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

```

Paragraph dialog

This dialog allows modifying the paragraph formatting for selection at once such as text alignment, indentation, and spacing.

To open this dialog, refer to the following example.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { DocumentEditorAllModule } from '@syncfusion/ej2-angular-documenteditor'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import {
    DocumentEditorComponent, SfdtExportService, SelectionService,
    ParagraphDialogService, EditorService
} from '@syncfusion/ej2-angular-documenteditor';
@Component({
    imports: [

        ButtonModule,
        DocumentEditorAllModule
    ],
    standalone: true,
    selector: 'app-container',
    //specifies the template string for the Document Editor component
    template: `<div>
        <button ej2-button (click)="btnClick()" >Show Dialog</button>
        <ejs-documenteditor #document_editor id="container" height="330px"
style="display:block" [isReadOnly]=false [enableSfdtExport]=true
        [enableParagraphDialog]=true [enableEditor]=true>
        </ejs-documenteditor>
    </div>`,
    encapsulation: ViewEncapsulation.None,
    providers: [SfdtExportService, SelectionService,
    ParagraphDialogService, EditorService]
})
export class AppComponent {
    @ViewChild('document_editor')
    public documentEditor?: DocumentEditorComponent;
    public btnClick(): void {
        //Open paragraph dialog.
        (this.documentEditor as
DocumentEditorComponent).showDialog('Paragraph');
    }
}

```

MAIN.TS

```

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

```

Table dialog

This dialog allows creating and inserting a table at cursor position by specifying the required number of rows and columns.

To open this dialog, refer to the following example.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'

```

```
import { BrowserModule } from '@angular/platform-browser'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { DocumentEditorAllModule } from '@syncfusion/ej2-angular-documenteditor'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import {
    DocumentEditorComponent, SfdtExportService, SelectionService,
    TableDialogService, EditorService
} from '@syncfusion/ej2-angular-documenteditor';
@Component({
    imports: [

        ButtonModule,
        DocumentEditorAllModule
    ],
    standalone: true,
    selector: 'app-container',
    template: `<div>
        <button ej2-button (click)="btnClick()" >Show Dialog</button>
        <ejs-documenteditor #document_editor id="container" height="330px"
        style="display:block" [isReadOnly]=false [enableSfdtExport]=true
        [enableTableDialog]=true [enableEditor]=true>
        </ejs-documenteditor>
    </div>`,
    encapsulation: ViewEncapsulation.None,
    providers: [SfdtExportService, SelectionService, TableDialogService,
    EditorService]
})
export class AppComponent {
    @ViewChild('document_editor')
    public documentEditor?: DocumentEditorComponent;
    public btnClick(): void {
        //Open table dialog.
        (this.documentEditor as
        DocumentEditorComponent).showDialog('Table');
    }
}
```

MAIN.TS

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

Bookmark dialog

This dialog allows you to perform the following operations:

- View all bookmarks.
- Navigate to a bookmark.
- Create a bookmark at current selection.
- Delete an existing bookmark.

To open this dialog, refer to the following example.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { DocumentEditorAllModule } from '@syncfusion/ej2-angular-documenteditor'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import {
    DocumentEditorComponent, SfddtExportService, SelectionService,
    BookmarkDialogService, EditorService
} from '@syncfusion/ej2-angular-documenteditor';
@Component({
    imports: [

        ButtonModule,
        DocumentEditorAllModule
    ],
    standalone: true,
    selector: 'app-container',
    //specifies the template string for the Document Editor component
    template: `<div>
        <button ej2-button (click)="btnClick()" >Show Dialog</button>
        <ejs-documenteditor #document_editor id="container" height="330px"
        style="display:block" [isReadOnly]=false [enableSfddtExport]=true
        [enableBookmarkDialog]=true [enableEditor]=true>
        </ejs-documenteditor>
        </div>`,
    encapsulation: ViewEncapsulation.None,
    providers: [SfddtExportService, SelectionService,
    BookmarkDialogService, EditorService]
})
export class AppComponent {
    @ViewChild('document_editor')
    public documentEditor?: DocumentEditorComponent;
    public btnClick(): void {
        //Open bookmark dialog.
        (this.documentEditor as
        DocumentEditorComponent).showDialog('Bookmark');
    }
}

```

MAIN.TS

```

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

```

Hyperlink dialog

This dialog allows editing or inserting a hyperlink at cursor position.

To open this dialog, refer to the following example.

APP.COMPONENT.TS


```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { DocumentEditorAllModule } from '@syncfusion/ej2-angular-documenteditor'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import {
    DocumentEditorComponent, SfdtExportService, SelectionService,
    HyperlinkDialogService, EditorService
} from '@syncfusion/ej2-angular-documenteditor';
@Component({
    imports: [

        ButtonModule,
        DocumentEditorAllModule
    ],
    standalone: true,
    selector: 'app-container',
    //specifies the template string for the Document Editor component
    template: `<div>
        <button ej2-button (click)="btnClick()" >Show Dialog</button>
        <ejs-documenteditor #document_editor id="container" height="330px"
style="display:block" [isReadOnly]=false [enableSfdtExport]=true
[enableHyperlinkDialog]=true [enableEditor]=true>
        </ejs-documenteditor>
    </div>`,
    encapsulation: ViewEncapsulation.None,
    providers: [SfdtExportService, SelectionService,
HyperlinkDialogService, EditorService]
})
export class AppComponent {
    @ViewChild('document_editor')
    public documentEditor?: DocumentEditorComponent;
    public btnClick(): void {
        //Open hyperlink dialog.
        (this.documentEditor as
DocumentEditorComponent).showDialog('Hyperlink');
    }
}

```

MAIN.TS

```

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

```

Table of contents dialog

This dialog allows creating and inserting table of contents at cursor position. If the table of contents already exists at cursor position, you can customize its properties.

To open this dialog, refer to the following example.

`typescript

```
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
```

```
import {
  DocumentEditorComponent, SfdtExportService, SelectionService, TableOfContentsDialogService,
  EditorService
} from '@syncfusion/ej2-angular-documenteditor';
@Component({
  selector: 'app-container',
  template: `<div">
    <button ej2-button (click)="btnClick()" >Show Dialog</button>
    <ejs-documenteditor #document_editor id="container" height="330px" style="display:block"
    [isReadOnly]=false [enableSfdtExport]=true
    [enableTableOfContentsDialog]=true [enableEditor]=true>
  </ejs-documenteditor>
</div>`,
  encapsulation: ViewEncapsulation.None,
  providers: [SfdtExportService, SelectionService, TableOfContentsDialogService, EditorService]
})
export class AppComponent {
  @ViewChild('document_editor')
  public documentEditor: DocumentEditorComponent;
  public btnClick(): void {
    //Open table of contents dialog.
    this.documentEditor.showDialog('TableOfContents');
  }
}
```

Styles Dialog

This dialog allows managing the styles in a document. It will display all the styles in the document with options to modify the properties of the existing style or create new style with the help of 'Style dialog'. Refer to the following example.

```
`typescript
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import {
  DocumentEditorComponent, SfdtExportService, SelectionService, StyleDialogService,
  StylesDialogService, EditorService
} from '@syncfusion/ej2-angular-documenteditor';
```

```

@Component({
  selector: 'app-container',
  template: `<div style="height:330px">
    <button ej-button (click)="btnClick()" >Show Dialog</button>

    <ejs-documenteditor #document_editor id="container" style="width: 100%;height: 100%;display:block"
    [isReadOnly]=false [enableSfdtExport]=true
    [enableStyleDialog]=true [enableStylesDialog]=true [enableEditor]=true>
  </ejs-documenteditor>
</div>`,
  encapsulation: ViewEncapsulation.None,
  providers: [SfdtExportService, SelectionService, StyleDialogService, StylesDialogService, EditorService]
})
export class AppComponent {
  @ViewChild('document_editor')
  public documentEditor: DocumentEditorComponent;
  public btnClick(): void {
    //Open styles dialog.
    this.documentEditor.showDialog('Styles');
  }
}
`

```

Style dialog

You can directly use this dialog for modifying any existing style or add new style by providing the style name.

To open this dialog, refer to the following example.

```

`typescript
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import {
  DocumentEditorComponent, SfdtExportService, SelectionService, StyleDialogService, EditorService
} from '@syncfusion/ej2-angular-documenteditor';
@Component({
  selector: 'app-container',
  template: `<div style="height:330px">
    <button ej-button (click)="btnClick()" >Show Dialog</button>

```

```

<ejs-documenteditor #document_editor id="container" style="width: 100%;height: 100%;display:block"
[isReadOnly]=false [enableSfdtExport]=true
[enableStyleDialog]=true [enableEditor]=true>
</ejs-documenteditor>
</div>`,
encapsulation: ViewEncapsulation.None,
providers: [SfdtExportService, SelectionService, StyleDialogService, EditorService]
})
export class AppComponent {
  @ViewChild('document_editor')
  public documentEditor: DocumentEditorComponent;
  public btnClick(): void {
    //Open style dialog.
    this.documentEditor.showDialog('Style');
  }
}
`

```

List dialog

This dialog allows creating a new list or modifying existing lists in the document.

To open this dialog, refer to the following example.

```

`typescript
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import {
  DocumentEditorComponent, SfdtExportService, SelectionService, ListDialogService, EditorService
} from '@syncfusion/ej2-angular-documenteditor';
@Component({
  selector: 'app-container',
  //specifies the template string for the Document Editor component
  template: `<div style="height:330px">
<button ej2-button (click)="btnClick()" >Show Dialog</button>
<ejs-documenteditor #document_editor id="container" style="width: 100%;height: 100%;display:block"
[isReadOnly]=false [enableSfdtExport]=true
[enableListDialog]=true [enableEditor]=true>
</ejs-documenteditor>
`

```

```

</div>`,
encapsulation: ViewEncapsulation.None,
providers: [SfdtExportService, SelectionService, ListDialogService, EditorService]
})
export class AppComponent {
  @ViewChild('document_editor')
  public documentEditor: DocumentEditorComponent;
  public btnClick(): void {
    //Open list dialog.
    this.documentEditor.showDialog('List');
  }
}
`

```

Borders and shading dialog

This dialog allows customizing the border style, border width, and background color of the table or selected cells.

To open this dialog, refer to the following example.

```

`typescript
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import {
  DocumentEditorComponent, SfdtExportService, SelectionService, BordersAndShadingDialogService,
  EditorService
} from '@syncfusion/ej2-angular-documenteditor';
@Component({
  selector: 'app-container',
  //specifies the template string for the Document Editor component
  template: `<div style="height:330px">
    <button ej-button (click)="btnClick()" >Show Dialog</button>
    <ejs-documenteditor #document_editor id="container" style="width: 100%;height: 100%;display:block"
    [isReadOnly]=false [enableSfdtExport]=true
    [enableBordersAndShadingDialog]=true [enableEditor]=true>
  </ejs-documenteditor>
</div>`,
  encapsulation: ViewEncapsulation.None,

```

```

providers: [SfdtExportService, SelectionService, BordersAndShadingDialogService, EditorService]
})
export class AppComponent {
  @ViewChild('document_editor')
  public documentEditor: DocumentEditorComponent;
  public btnClick(): void {
    //Open borders and shading dialog.
    this.documentEditor.showDialog('BordersAndShading');
  }
}
`

```

Table options dialog

This dialog allows customizing the default cell margins and spacing between each cells of the selected table.

To open this dialog, refer to the following example.

```

`typescript
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import {
  DocumentEditorComponent, SfdtExportService, SelectionService, TableOptionsDialogService,
  EditorService
} from '@syncfusion/ej2-angular-documenteditor';
@Component({
  selector: 'app-container',
  //specifies the template string for the Document Editor component
  template: `<div style="height:330px">
    <button ej2-button (click)="btnClick()" >Show Dialog</button>
    <ejs-documenteditor #document_editor id="container" style="width: 100%;height: 100%;display:block"
    [isReadOnly]=false [enableSfdtExport]=true
    [enableTableOptionsDialog]=true [enableEditor]=true>
  </ejs-documenteditor>
</div>`,
  encapsulation: ViewEncapsulation.None,
  providers: [SfdtExportService, SelectionService, TableOptionsDialogService, EditorService]
})
`

```

```
export class AppComponent {
  @ViewChild('document_editor')
  public documentEditor: DocumentEditorComponent;
  public btnClick(): void {
    //Open table options dialog.
    this.documentEditor.showDialog('TableOptions');
  }
}
```

Table properties dialog

This dialog allows customizing the table, row, and cell properties of the selected table.

To open this dialog, refer to the following example.

`typescript

```
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import {
  DocumentEditorComponent, SfdtExportService, SelectionService, TableOptionsDialogService,
  TablePropertiesDialogService, BordersAndShadingDialogService, EditorService
} from '@syncfusion/ej2-angular-documenteditor';
@Component({
  selector: 'app-container',
  //specifies the template string for the Document Editor component
  template: `<div style="height:330px">
    <button ej-button (click)="btnClick()" >Show Dialog</button>
    <ejs-documenteditor #document_editor id="container" style="width: 100%;height: 100%;display:block"
    [isReadOnly]=false [enableSfdtExport]=true
    [enableTableOptionsDialog]=true [enableTablePropertiesDialog]=true
    [enableBordersAndShadingDialog]=true [enableEditor]=true>
  </ejs-documenteditor>
</div>`,
  encapsulation: ViewEncapsulation.None,
  providers: [SfdtExportService, SelectionService, TableOptionsDialogService,
    TablePropertiesDialogService, BordersAndShadingDialogService, EditorService]
})
export class AppComponent {
  @ViewChild('document_editor')
```

```
public documentEditor: DocumentEditorComponent;

public btnClick(): void {
  //open table properties dialog.
  this.documentEditor.showDialog('TableProperties');
}
}
`
`
```

Page setup dialog

This dialog allows customizing margins, size, and layout options for pages of the section.

To open this dialog, refer to the following example.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { DocumentEditorAllModule } from '@syncfusion/ej2-angular-documenteditor'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import {
  DocumentEditorComponent, SfdtExportService, SelectionService,
  PageSetupDialogService, EditorService
} from '@syncfusion/ej2-angular-documenteditor';
@Component({
  imports: [
    ButtonModule,
    DocumentEditorAllModule
  ],
  standalone: true,
  selector: 'app-container',
  //specifies the template string for the Document Editor component
  template: `<div>
    <button ej2-button (click)="btnClick()" >Show Dialog</button>
    <ejs-documenteditor #document_editor id="container" height="330px"
    style="display:block" [isReadOnly]=false [enableSfdtExport]=true
    [enablePageSetupDialog]=true [enableEditor]=true>
    </ejs-documenteditor>
  </div>`,
  encapsulation: ViewEncapsulation.None,
  providers: [SfdtExportService, SelectionService,
  PageSetupDialogService, EditorService]
})
export class AppComponent {
  @ViewChild('document_editor')
  public documentEditor?: DocumentEditorComponent;
  public btnClick(): void {
    //Open page setup dialog.
    (this.documentEditor as
    DocumentEditorComponent).showDialog('PageSetup');
  }
}
```



```
}

```

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

- [Feature module](#)

Right to left in Angular Document editor component

Document Editor provides RTL (right-to-left) support. This can be enabled using the “enableRtl” property.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { DocumentEditorAllModule } from '@syncfusion/ej2-angular-documenteditor'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import { L10n } from '@syncfusion/ej2-base';
import {
    DocumentEditorComponent, PrintService, SfdtExportService,
    WordExportService, TextExportService, SelectionService,
    SearchService, EditorService, ImageResizerService, EditorHistoryService,
    ContextMenuService,
    OptionsPaneService, HyperlinkDialogService, TableDialogService,
    BookmarkDialogService, TableOfContentsDialogService,
    PageSetupDialogService, StyleDialogService, ListDialogService,
    ParagraphDialogService, BulletsAndNumberingDialogService,
    FontDialogService, TablePropertiesDialogService,
    BordersAndShadingDialogService, TableOptionsDialogService,
    CellOptionsDialogService, StylesDialogService
} from '@syncfusion/ej2-angular-documenteditor';
//Set locale content.
L10n.load({
    'ar-AE': {
        'documenteditor': {
            'Table': 'الجدول',
            'Row': 'الصف',
            'Cell': 'الخلية',
            'Ok': 'موافق',
            'Cancel': 'إلغاء الأمر',
            'Size': 'حجم',
            'Preferred Width': 'العرض المفضل',
            'Points': 'نقاط',
            'Percent': 'المائه',
            'Measure in': 'القياس في',
            'Alignment': 'محاذاة',

```

```

'Left': 'ليسار',
'Center': 'مركز',
'Right': 'الحق',
'Justify': 'تبرير',
'Indent from left': 'مسافة بادئه من اليسار',
'Borders and Shading': 'الحدود والتظليل',
'Options': 'خيارات',
'Specify height': 'تحديد الارتفاع',
'At least': 'الاقل',
'Exactly': 'تماما',
'Row height is': 'ارتفاع الصف هو',
'Allow row to break across pages': 'السماح بفصل الصف عبر',
الصفحات',
'Repeat as header row at the top of each page': 'تكرار كصف راس',
في اعلي كل صفحه',
'Vertical alignment': 'محاذاة عمودي',
'Top': 'أعلى',
'Bottom': 'اسفل',
'Default cell margins': 'هوامش الخلية الافتراضية',
'Default cell spacing': 'تباعد الخلايا الافتراضي',
'Allow spacing between cells': 'السماح بالتباعد بين الخلايا',
'Cell margins': 'هوامش الخلية',
'Same as the whole table': 'نفس الجدول بأكمله',
'Borders': 'الحدود',
'None': 'اي',
'Single': 'واحد',
'Dot': 'نقطه',
'DashSmallGap': 'داشسمجاب',
'DashLargeGap': 'الاتحاد الخاص',
'DashDot': 'داشدوت',
'DashDotDot': 'ددهدودوت',
'Double': 'انقر نقرا مزدوجا',
'Triple': 'الثلاثي',
'ThinThickSmallGap': 'فجوه صغيره سميكه رقيق',
'ThickThinSmallGap': 'الفجوة الصغيرة رقيقه سميكه',
'ThinThickThinSmallGap': 'فجوه صغيره سميكه رقيقه الفجوة الصغيرة',
'ThinThickMediumGap': 'فجوه متوسطه سميك',
'ThickThinMediumGap': 'سميكه الفجوة متوسطه رقيقه',
'ThinThickThinMediumGap': 'رقيقه سميكه متوسطه الفجوة',
'ThinThickLargeGap': 'الفجوة الكبيرة رقيقه سميكه',
'ThickThinLargeGap': 'فجوه كبيره رقيقه سميك',
'ThinThickThinLargeGap': 'رقيقه سميكه الفجوة الكبيرة',
'SingleWavy': 'واحد مائج',
'DoubleWavy': 'مزدوج مائج',
'DashDotStroked': 'اندفاعه نقطه القوية',
'Emboss3D': 'مزخرفD3',
'Engrave3D': 'نقشD3',
'Outset': 'البدايه',
'Inset': 'الداخلي',
'Thick': 'سميكه',
'Style': 'نمط',
'Width': 'عرض',
'Height': 'ارتفاع',
'Letter': 'رساله',
'Tabloid': 'التابلويد',
'Legal': 'القانونيه',
'Statement': 'بيان',

```

```

'Executive': 'التنفيذي',
'A3': 'A3',
'A4': 'A4',
'A5': 'A5',
'B4': 'B4',
'B5': 'B5',
'Custom Size': 'حجم مخصص',
'Different odd and even': 'مختلفه غريبه وحتى',
'Different first page': 'الصفحة الاولى مختلفه',
'From edge': 'من الحافة',
'Header': 'رأس',
'Footer': 'تذييل الصفحة',
'Margin': 'الهوامش',
'Paper': 'الورق',
'Layout': 'تخطيط',
'Orientation': 'التوجه',
'Landscape': 'المناظر الطبيعيه',
'Portrait': 'صوره',
'Table Of Contents': 'جدول المحتويات',
'Show page numbers': 'إظهار أرقام الصفحات',
'Right align page numbers': 'محاذاة أرقام الصفحات إلى اليمين',
'Nothing': 'شيء',
'Tab leader': 'قائد علامة التبويب',
'Show levels': 'إظهار المستويات',
'Use hyperlinks instead of page numbers': 'استخدام الارتباطات',
'التشعبية بدلا من أرقام الصفحات',
'Build table of contents from': 'بناء جدول محتويات من',
'Styles': 'انماط',
'Available styles': 'الأنماط المتوفرة',
'TOC level': 'مستوي جدول المحتويات',
'Heading': 'عنوان',
'Heading 1': 'عنوان 1',
'Heading 2': 'عنوان 2',
'Heading 3': 'عنوان 3',
'Heading 4': 'عنوان 4',
'Heading 5': 'عنوان 5',
'Heading 6': 'عنوان 6',
'List Paragraph': 'فقره القائمة',
'Normal': 'العادي',
'Outline levels': 'مستويات المخطط التفصيلي',
'Table entry fields': 'حقول إدخال الجدول',
'Modify': 'تعديل',
'Color': 'لون',
'Setting': 'اعداد',
'Box': 'مربع',
'All': 'جميع',
'Custom': 'المخصصه',
'Preview': 'معاينه',
'Shading': 'التظليل',
'Fill': 'ملء',
'Apply To': 'تنطبق علي',
'Table Properties': 'خصائص الجدول',
'Cell Options': 'خيارات الخلية',
'Table Options': 'خيارات الجدول',
'Insert Table': 'ادراج جدول',
'Number of columns': 'عدد الاعمده',
'Number of rows': 'عدد الصفوف',

```

```

'Text to display': 'النص الذي سيتم عرضه',
'Address': 'عنوان',
'Insert Hyperlink': 'ادراج ارتباط تشعبي',
'Edit Hyperlink': 'تحرير ارتباط تشعبي',
'Insert': 'ادراج',
'General': 'العامه',
'Indentation': 'المسافه البادئه',
'Before text': 'قبل النص',
'Special': 'الخاصه',
'First line': 'السطر الأول',
'Hanging': 'معلقه',
'After text': 'بعد النص',
'By': 'من',
'Before': 'قبل',
'Line Spacing': 'تباعد الأسطر',
'After': 'بعد',
'At': 'في',
'Multiple': 'متعدده',
'Spacing': 'تباعد',
'Define new Multilevel list': 'تحديد قائمه متعددة الاصعده جديده',
'List level': 'مستوي القائمة',
'Choose level to modify': 'اختر المستوي الذي تريد تعديله',
'Level': 'مستوي',
'Number format': 'تنسيق الأرقام',
'Number style for this level': 'نمط الرقم لهذا المستوي',
'Enter formatting for number': 'إدخال تنسيق لرقم',
'Start at': 'بداية من',
'Restart list after': 'أعاده تشغيل قائمه بعد',
'Position': 'موقف',
'Text indent at': 'المسافة البادئة للنص في',
'Aligned at': 'محاذاة في',
'Follow number with': 'اتبع الرقم مع',
'Tab character': 'حرف علامة التبويب',
'Space': 'الفضاء',
'Arabic': 'العربية',
'UpRoman': 'حتى الروماني',
'LowRoman': 'الرومانية منخفضه',
'UpLetter': '',
'LowLetter': '',
'Number': 'عدد',
'Leading zero': 'يؤدي صفر',
'Bullet': 'رصاصه',
'Ordinal': 'الترتيبيه',
'Ordinal Text': 'النص الترتيبي',
'For East': 'للشرق',
'No Restart': 'لا أعاده تشغيل',
'Font': 'الخط',
'Font style': 'نمط الخط',
'Underline style': 'نمط التسطير',
'Font color': 'لون الخط',
'Effects': 'الاثار',
'Strikethrough': 'يتوسطه',
'Superscript': 'مرتفع',
'Subscript': 'منخفض',
'Double strikethrough': 'خط مزدوج يتوسطه خط',
'Regular': 'العاديه',
'Bold': 'جريئه',

```

```

'Italic': 'مائل',
'Cut': 'قطع',
'Copy': 'نسخ',
'Paste': 'الصق',
'Hyperlink': 'الارتباط التشعبي',
'Open Hyperlink': 'فتح ارتباط تشعبي',
'Copy Hyperlink': 'نسخ ارتباط تشعبي',
'Remove Hyperlink': 'أزله ارتباط تشعبي',
'Paragraph': 'الفقره',
'Linked(Paragraph and Character)': 'مرتبط (فقره وحرف)',
'Character': 'حرف',
'Merge Cells': 'دمج الخلايا',
'Insert Above': 'ادراج أعلاه',
'Insert Below': 'ادراج أدناه',
'Insert Left': 'ادراج إلى اليسار',
'Insert Right': 'ادراج اليمين',
>Delete': 'حذف',
>Delete Table': 'حذف جدول',
>Delete Row': 'حذف صف',
>Delete Column': 'حذف عمود',
'File Name': 'اسم الملف',
'Format Type': 'نوع التنسيق',
'Save': 'حفظ',
'Navigation': 'التنقل',
'Results': 'نتائج',
'Replace': 'استبدال',
'Replace All': 'استبدال الكل',
'We replaced all': 'استبدلنا جميع',
'Find': 'العثور',
'No matches': 'لا توجد تطابقات',
'All Done': 'كل القيام به',
'Result': 'نتيجه',
'of': 'من',
'instances': 'الحالات',
'with': 'مع',
'Click to follow link': 'انقر لمتابعه الارتباط',
'Continue Numbering': 'متابعه الترقيم',
'Bookmark name': 'اسم الإشارة المرجعية',
'Close': 'اغلق',
'Restart At': 'أعاده التشغيل عند',
'Properties': 'خصائص',
'Name': 'اسم',
'Style type': 'نوع النمط',
'Style based on': 'نمط استنادا إلى',
'Style for following paragraph': 'نمط للفقره التاليه',
'Formatting': 'التنسيق',
'Numbering and Bullets': 'الترقيم والتعداد النقطي',
'Numbering': 'ترقيم',
'Update Field': 'تحديث الحقل',
'Edit Field': 'تحرير الحقل',
'Bookmark': 'الإشارة المرجعية',
'Page Setup': 'اعداد الصفحة',
'No bookmarks found': 'لم يتم العثور علي إشارات مرجعيه',
'Number format tooltip information': 'تنسيق رقم أحادي المستوي'
+ '</br>' + '[' + 'بأدئه' + '%[مستوي الاعداد] للاحقه' + '</br>'
// tslint:disable-next-line:max-line-length

```

```

+ 'على سبيل المثال ، "الفصل 1%". سيتم عرض الترقيم مثل' +
'</br>' + 'البند - الفصل الأول' + '</br>' + 'البند' + '</br>...' +
+ 'الفصل نون-البند' + '</br>'
// tslint:disable-next-line:max-line-length
+ 'البند' + '</br>' + 'تنسيق رقم متعدد الإعدادات' + '</br>' +
'المستوي المستوي]' + 'باده' + 'باده' + 'لاحقه' + 'لاحقه' +
'على سبيل المثال ، "1% 2%". سيتم عرض الترقيم
' + '</br>' + 'البند 1.1' + '</br>' + 'البند 1.2' + '</br>...' + '</br>'
+ 'نون-البند 1.',
'Format': 'تنسيق',
'Create New Style': 'إنشاء نمط جديد',
'Modify Style': 'تعديل النمط',
'New': 'الجديد',
'Bullets': 'الرصص',
'Use bookmarks': 'استخدام الإشارات المرجعية',
'Table of Contents': 'جدول المحتويات',
'AutoFit': 'الاحتواء',
'AutoFit to Contents': 'احتواء تلقائي للمحتويات',
'AutoFit to Window': 'احتواء تلقائي للإطار',
'Fixed Column Width': 'عرض العمود الثابت',
'Reset': 'إعادة تعيين',
'Match case': 'حاله المباراة',
'Whole words': 'كلمات كامل',
'Add': 'اضافه',
'Go To': 'الانتقال إلى',
'Search for': 'البحث عن',
'Replace with': 'استبدال',
'TOC 1': 'جدول المحتويات 1',
'TOC 2': 'جدول المحتويات 2',
'TOC 3': 'جدول المحتويات 3',
'TOC 4': 'جدول المحتويات 4',
'TOC 5': 'جدول المحتويات 5',
'TOC 6': 'جدول المحتويات 6',
'TOC 7': 'جدول المحتويات 7',
'TOC 8': 'جدول المحتويات 8',
'TOC 9': 'جدول المحتويات 9',
'Right-to-left': 'من اليمين إلى اليسار',
'Left-to-right': 'من اليسار إلى اليمين',
'Direction': 'الاتجاه',
'Table direction': 'اتجاه الجدول',
'Indent from right': 'مسافة بادئه من اليمين',
'Page': 'صفحه',
'Fit one page': 'احتواء صفحه واحد',
'Fit page width': 'احتواء عرض الصفحة',
// tslint:disable-next-line:max-line-length
'The current page number in the document. Click or tap to
رقم الصفحة الحالية في المستند. انقر أو اضغط:
'للتنقل في صفحه معينه'
},
'colorpicker': {
'Apply': 'تطبيق',
'Cancel': 'إلغاء الأمر',
'ModeSwitcher': 'مفتاح كهربائي الوضع'
}
}
});

```

```

@Component({
  imports: [

    ButtonModule,
    DocumentEditorAllModule

  ],
  standalone: true,
  selector: 'app-container',
  //specifies the template string for the Document Editor component with
  //all required service enabled.
  template: `<ejs-documenteditor #document_editor id="container"
height="330px" style="display:block" [isReadOnly]=false
[enableSelection]=true
[enablePrint]=true [enableSfdtExport]=true [enableWordExport]=true
[enableOptionsPane]=true [enableContextMenu]=true
[enableHyperlinkDialog]=true [enableBookmarkDialog]=true
[enableTableOfContentsDialog]=true [enableSearch]=true
[enableParagraphDialog]=true [enableListDialog]=true
[enableTablePropertiesDialog]=true [enableBordersAndShadingDialog]=true
[enablePageSetupDialog]=true [enableStyleDialog]=true
[enableFontDialog]=true [enableTableOptionsDialog]=true
[enableTableDialog]=true [enableImageResizer]=true
[enableEditor]=true [enableEditorHistory]=true [enableRtl]=true
[locale]="culture" (created)="onCreated()">
    </ejs-documenteditor>`,
  encapsulation: ViewEncapsulation.None,
  //Provide require service.
  providers: [PrintService, SfdtExportService, WordExportService,
TextExportService, SelectionService, SearchService, EditorService,
ImageResizerService, EditorHistoryService, ContextMenuService,
OptionsPaneService, HyperlinkDialogService, TableDialogService,
BookmarkDialogService, TableOfContentsDialogService,
PageSetupDialogService, StyleDialogService, ListDialogService,
ParagraphDialogService, BulletsAndNumberingDialogService,
FontDialogService, TablePropertiesDialogService,
BordersAndShadingDialogService, TableOptionsDialogService,
CellOptionsDialogService, StylesDialogService]
})
export class AppComponent {
  @ViewChild('document_editor')
  public documentEditor?: DocumentEditorComponent;
  //Culture constant.
  public culture: string = 'ar-AE';
  onCreated() {
    if ((this.documentEditor as
DocumentEditorComponent).isDocumentLoaded) {
      let sfdt: string = `{
        "sections": [
          {
            "blocks": [
              {
                "characterFormat": {
                  "fontSize": 18.0,
                  "fontFamily": "Calibri",
                  "fontFamilyBidi": "Calibri"
                },
                "paragraphFormat": {

```

```

        "beforeSpacing": 18.0,
        "afterSpacing": 30.0,
        "styleName": "Heading 1",
        "bidirectional": true
    },
    "inlines": [
        {
            "text": "اعمال المغامرة دورات",
            "characterFormat": {
                "fontSize": 18.0,
                "bidirectional": true,
                "fontSizeBidi": 18.0
            }
        }
    ]
}

]

}

]

};
//Open the default document in Document Editor.
(this.documentEditor as DocumentEditorComponent).open(sfds);
}
}

```

MAIN.TS

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

Chart in Angular Document editor component

Document Editor provides chart preservation support. Using Document Editor, you can see the chart reports from your Word document.

The following example shows chart preservation in Document Editor.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { DocumentEditorAllModule } from '@syncfusion/ej2-angular-
documenteditor'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import {
    DocumentEditorComponent
} from '@syncfusion/ej2-angular-documenteditor';
@Component({
imports: [

        ButtonModule,
        DocumentEditorAllModule

    ],
```



```

standalone: true,
    selector: 'app-container',
    //specifies the template string for the DocumentEditorContainer
    component
    template: `<div style="width:100%;">
        <ejs-documenteditor #document_editor id="container"
(created)="onCreated()" height="330px" style="display:block"></ejs-
documenteditor>
        </div>`,
    encapsulation: ViewEncapsulation.None,
})
export class AppComponent {
    @ViewChild('document_editor')
    public documentEditor?: DocumentEditorComponent;
    onCreated() {
        if ((this.documentEditor as
DocumentEditorComponent).isDocumentLoaded) {
            let sfdt: string =
`{"sections":[{"sectionFormat":{"pageWidth":612,"pageHeight":792,"leftMargin
":72,"rightMargin":72,"topMargin":72,"bottomMargin":72,"differentFirstPage":
false,"differentOddAndEvenPages":false,"headerDistance":36,"footerDistance":
36,"bidi":false},"blocks":[{"paragraphFormat":{"textAlignment":"Center","aft
erSpacing":0,"lineSpacing":1,"lineSpacingType":"Multiple","styleName":"Norma
l","listFormat":{}},"characterFormat":{"bold":true,"fontSize":12,"fontFamily
":"Verdana","fontSizeBidi":12,"fontFamilyBidi":"Verdana"},"inlines":[{"chara
cterFormat":{"bold":true,"fontSize":14,"fontFamily":"Verdana","fontColor":"#
17365DFF","styleName":"a","fontSizeBidi":14,"fontFamilyBidi":"Verdana"},"tex
t":"Northwind Management
Report"}]},{"paragraphFormat":{"afterSpacing":0,"lineSpacing":1,"lineSpacing
Type":"Multiple","styleName":"Normal","listFormat":{}},"characterFormat":{"f
ontSize":10,"fontFamily":"Verdana","fontSizeBidi":10,"fontFamilyBidi":"Verda
na"},"inlines":[]},{paragraphFormat":{"afterSpacing":0,"styleName":"Normal"
,"listFormat":{}},"characterFormat":{,"inlines":[{"characterFormat":{"fontS
ize":10,"fontFamily":"Verdana","styleName":"a","fontSizeBidi":10,"fontFamily
Bidi":"Verdana"},"text":"This management report provides information
obtained through data analysis, regarding the
"},{"characterFormat":{"fontSize":10,"fontFamily":"Verdana","styleName":"a",
"fontSizeBidi":10,"fontFamilyBidi":"Verdana"},"text":"performance of
Northwind Traders. This report will pay
particular"},{"characterFormat":{"fontSize":10,"fontFamily":"Verdana","style
Name":"a","fontSizeBidi":10,"fontFamilyBidi":"Verdana"},"text":"
"},{"characterFormat":{"fontSize":10,"fontFamily":"Verdana","styleName":"a",
"fontSizeBidi":10,"fontFamilyBidi":"Verdana"},"text":" attention to the
"},{"characterFormat":{"fontSize":10,"fontFamily":"Verdana","styleName":"a",
"fontSizeBidi":10,"fontFamilyBidi":"Verdana"},"text":"best-selling products,
of our company.
"},{"characterFormat":{"fontSize":10,"fontFamily":"Verdana","fontSizeBidi":1
0,"fontFamilyBidi":"Times New Roman"},"text":"The best-selling products of
Northwind Traders
"},{"characterFormat":{"fontSize":10,"fontFamily":"Verdana","fontSizeBidi":1
0,"fontFamilyBidi":"Times New Roman"},"text":"Company as follows:
"}]},{paragraphFormat":{"afterSpacing":0,"styleName":"Normal","listFormat":
{}},"characterFormat":{,"inlines":[]},{rows":[{"cells":[{"blocks":[{"parag
raphFormat":{"rightIndent":26.850000381469727,"styleName":"Normal","listForm
at":{}},"characterFormat":{,"inlines":[{"characterFormat":{"fontSize":10,"f
ontFamily":"Verdana","fontSizeBidi":10,"fontFamilyBidi":"Times New
Roman"},"text":"S.No"}]},{cellFormat":{"borders":{"top":{"color":"#4472C4FF

```

```

", "hasNoneStyle": false, "lineStyle": "Single", "lineWidth": 0.5, "shadow": false, "
space": 0}, "left": {"color": "#4472C4FF", "hasNoneStyle": false, "lineStyle": "Sing
le", "lineWidth": 0.5, "shadow": false, "space": 0}, "right": {"color": "#8EAADBFF", "
hasNoneStyle": false, "lineStyle": "Single", "lineWidth": 0.5, "shadow": false, "spa
ce": 0}, "bottom": {"color": "#4472C4FF", "hasNoneStyle": false, "lineStyle": "Singl
e", "lineWidth": 0.5, "shadow": false, "space": 0}, "diagonalDown": {"color": "#000000
", "hasNoneStyle": false, "lineStyle": "None", "lineWidth": 0, "shadow": false, "spa
ce": 0}, "diagonalUp": {"color": "#000000", "hasNoneStyle": false, "lineStyle": "Non
e", "lineWidth": 0, "shadow": false, "space": 0}, "horizontal": {"color": "#000000", "
hasNoneStyle": false, "lineStyle": "None", "lineWidth": 0, "shadow": false, "space":
0}, "vertical": {"color": "#000000", "hasNoneStyle": false, "lineStyle": "None", "li
neWidth": 0, "shadow": false, "space": 0}}, "shading": {"backgroundColor": "#4472C4F
F", "foregroundColor": "empty", "textureStyle": "TextureNone"}, "preferredWidth":
13.420000076293945, "preferredWidthType": "Percent", "cellWidth": 64.71214527422
465, "columnSpan": 1, "rowSpan": 1, "verticalAlignment": "Top"}, {"columnIndex": 0}, {
"blocks": [{"paragraphFormat": {"styleName": "Normal", "listFormat": {}}, "charact
erFormat": {}, "inlines": [{"characterFormat": {"fontSize": 10, "fontFamily": "Verd
ana", "fontSizeBidi": 10, "fontFamilyBidi": "Times New Roman"}, "text": "Product
Name"}]}], "cellFormat": {"borders": {"top": {"color": "#4472C4FF", "hasNoneStyle"
: false, "lineStyle": "Single", "lineWidth": 0.5, "shadow": false, "space": 0}, "left"
: {"color": "#8EAADBFF", "hasNoneStyle": false, "lineStyle": "Single", "lineWidth":
0.5, "shadow": false, "space": 0}, "right": {"color": "#8EAADBFF", "hasNoneStyle": fa
lse, "lineStyle": "Single", "lineWidth": 0.5, "shadow": false, "space": 0}, "bottom":
{"color": "#4472C4FF", "hasNoneStyle": false, "lineStyle": "Single", "lineWidth": 0
.5, "shadow": false, "space": 0}, "diagonalDown": {"color": "#000000", "hasNoneStyle
": false, "lineStyle": "None", "lineWidth": 0, "shadow": false, "space": 0}, "diagonal
Up": {"color": "#000000", "hasNoneStyle": false, "lineStyle": "None", "lineWidth": 0
, "shadow": false, "space": 0}, "horizontal": {"color": "#000000", "hasNoneStyle": fa
lse, "lineStyle": "None", "lineWidth": 0, "shadow": false, "space": 0}, "vertical": {"
color": "#000000", "hasNoneStyle": false, "lineStyle": "None", "lineWidth": 0, "shad
ow": false, "space": 0}}, "shading": {"backgroundColor": "#4472C4FF", "foregroundColo
r": "empty", "textureStyle": "TextureNone"}, "preferredWidth": 48.8600006103515
6, "preferredWidthType": "Percent", "cellWidth": 292.87942351880633, "columnSpan"
: 1, "rowSpan": 1, "verticalAlignment": "Top"}, {"columnIndex": 1}, {"blocks": [{"para
graphFormat": {"styleName": "Normal", "listFormat": {}}, "characterFormat": {}, "in
lines": [{"characterFormat": {"fontSize": 10, "fontFamily": "Verdana", "fontSizeBi
di": 10, "fontFamilyBidi": "Times New Roman"}, "text": "Sum of Sales(in
$)"}]}], "cellFormat": {"borders": {"top": {"color": "#4472C4FF", "hasNoneStyle": f
alse, "lineStyle": "Single", "lineWidth": 0.5, "shadow": false, "space": 0}, "left": {
"color": "#8EAADBFF", "hasNoneStyle": false, "lineStyle": "Single", "lineWidth": 0.
5, "shadow": false, "space": 0}, "right": {"color": "#4472C4FF", "hasNoneStyle": fals
e, "lineStyle": "Single", "lineWidth": 0.5, "shadow": false, "space": 0}, "bottom": {"
color": "#4472C4FF", "hasNoneStyle": false, "lineStyle": "Single", "lineWidth": 0.5
, "shadow": false, "space": 0}, "diagonalDown": {"color": "#000000", "hasNoneStyle":
false, "lineStyle": "None", "lineWidth": 0, "shadow": false, "space": 0}, "diagonalUp
": {"color": "#000000", "hasNoneStyle": false, "lineStyle": "None", "lineWidth": 0, "
shadow": false, "space": 0}, "horizontal": {"color": "#000000", "hasNoneStyle": fals
e, "lineStyle": "None", "lineWidth": 0, "shadow": false, "space": 0}, "vertical": {"co
lor": "#000000", "hasNoneStyle": false, "lineStyle": "None", "lineWidth": 0, "shadow
": false, "space": 0}}, "shading": {"backgroundColor": "#4472C4FF", "foregroundColo
r": "empty", "textureStyle": "TextureNone"}, "preferredWidth": 37.720001220703125
, "preferredWidthType": "Percent", "cellWidth": 117.95841899993776, "columnSpan":
1, "rowSpan": 1, "verticalAlignment": "Top"}, {"columnIndex": 2}], "rowFormat": {"hei
ght": 14.399999618530273, "allowBreakAcrossPages": true, "heightType": "Exactly",
"isHeader": false, "borders": {"top": {"color": "#8EAADBFF", "hasNoneStyle": false,
"lineStyle": "Single", "lineWidth": 0.5, "shadow": false, "space": 0}, "left": {"colo
r": "#8EAADBFF", "hasNoneStyle": false, "lineStyle": "Single", "lineWidth": 0.5, "sh

```

```
adown":{"color":"black","space":0},"right":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},"bottom":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},"diagonalDown":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0},"diagonalUp":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0},"horizontal":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},"vertical":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0}},{"gridBefore":0,"gridBeforeWidth":0,"gridBeforeWidthType":"Point"},"gridAfter":0,"gridAfterWidth":0,"gridAfterWidthType":"Point"}}, {"cells":[{"blocks":[{"paragraphFormat":{"styleName":"Normal","listFormat":{}},"characterFormat":{},"inlines":[{"characterFormat":{"fontSize":10,"fontFamily":"Verdana","fontSizeBidi":10,"fontFamilyBidi":"Times New Roman"},"text":["1"]]}]}, {"cellFormat":{"borders":{"top":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},"left":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},"right":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},"bottom":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},"diagonalDown":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0},"diagonalUp":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0},"horizontal":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0},"vertical":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0}}, "shading":{"backgroundColor":"#D9E2F3FF","foregroundColor":"empty","textureStyle":"TextureNone"},"preferredWidth":13.420000076293945,"preferredWidthType":"Percent","cellWidth":64.71214527422465,"columnSpan":1,"rowSpan":1,"verticalAlignment":"Top"},"columnIndex":0},{ "blocks":[{"paragraphFormat":{"styleName":"Normal","listFormat":{}},"characterFormat":{},"inlines":[{"characterFormat":{"fontSize":10,"fontFamily":"Verdana","fontSizeBidi":10,"fontFamilyBidi":"Times New Roman"},"text":["Côte de Blaye"]}]}]}, {"cellFormat":{"borders":{"top":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},"left":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},"right":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},"bottom":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},"diagonalDown":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0},"diagonalUp":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0},"horizontal":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0},"vertical":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0}}, "shading":{"backgroundColor":"#D9E2F3FF","foregroundColor":"empty","textureStyle":"TextureNone"},"preferredWidth":48.86000061035156,"preferredWidthType":"Percent","cellWidth":292.87942351880633,"columnSpan":1,"rowSpan":1,"verticalAlignment":"Top"},"columnIndex":1},{ "blocks":[{"paragraphFormat":{"styleName":"Normal","listFormat":{}},"characterFormat":{},"inlines":[{"characterFormat":{"fontSize":10,"fontFamily":"Verdana","fontSizeBidi":10,"fontFamilyBidi":"Times New Roman"},"text":["141.396"]}]}]}, {"cellFormat":{"borders":{"top":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},"left":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},"right":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,
```

```
space":0},"bottom":{"color":"#8EADBFFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},"diagonalDown":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0},"diagonalUp":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0},"horizontal":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0},"vertical":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0}},"shading":{"backgroundColor":"#D9E2F3FF"},"foregroundColor":"empty","textureStyle":"TextureNone"},"preferredWidth":37.720001220703125,"preferredWidthType":"Percent","cellWidth":117.95841899993776,"columnSpan":1,"rowSpan":1,"verticalAlignment":"Top"},"columnIndex":2}],"rowFormat":{"height":14.3999999618530273,"allowBreakAcrossPages":true,"heightType":"Exactly","isHeader":false,"borders":{"top":{"color":"#8EADBFFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},"left":{"color":"#8EADBFFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},"right":{"color":"#8EADBFFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},"bottom":{"color":"#8EADBFFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},"diagonalDown":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0},"diagonalUp":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0},"horizontal":{"color":"#8EADBFFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},"vertical":{"color":"#8EADBFFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0}},"gridBeforeWidth":0,"gridBeforeHeight":0,"gridAfterWidth":0,"gridAfterHeight":0,"cells":[{"blocks":[{"paragraphFormat":{"styleName":"Normal","listFormat":{},"characterFormat":{},"inlines":[{"characterFormat":{"fontSize":10,"fontFamily":"Verdana","fontSizeBidi":10,"fontFamilyBidi":"Times New Roman"},"text":"2"}]}]}],"cellFormat":{"borders":{"top":{"color":"#8EADBFFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},"left":{"color":"#8EADBFFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},"right":{"color":"#8EADBFFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},"bottom":{"color":"#8EADBFFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},"diagonalDown":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0},"diagonalUp":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0},"horizontal":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0},"vertical":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0}},"shading":{"backgroundColor":"#FFFFFFF","foregroundColor":"empty","textureStyle":"TextureNone"},"preferredWidth":13.420000076293945,"preferredWidthType":"Percent","cellWidth":64.71214527422465,"columnSpan":1,"rowSpan":1,"verticalAlignment":"Top"},"columnIndex":0},{"blocks":[{"paragraphFormat":{"styleName":"Normal","listFormat":{},"characterFormat":{},"inlines":[{"characterFormat":{"fontSize":10,"fontFamily":"Verdana","fontSizeBidi":10,"fontFamilyBidi":"Times New Roman"},"text":"Thüringer Rostbratwurst"}]}]}],"cellFormat":{"borders":{"top":{"color":"#8EADBFFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},"left":{"color":"#8EADBFFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},"right":{"color":"#8EADBFFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},"bottom":{"color":"#8EADBFFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},"diagonalDown":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0}}
```

```

"diagonalUp":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0},"horizontal":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0},"vertical":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0}},"shading":{"backgroundColor":"#FFFFFF","foregroundColor":"empty","textureStyle":"TextureNone"},"preferredWidth":48.86000061035156,"preferredWidthType":"Percent","cellWidth":292.87942351880633,"columnSpan":1,"rowSpan":1,"verticalAlignment":"Top"},"columnIndex":1},{
"blocks":[{"paragraphFormat":{"styleName":"Normal","listFormat":{},"characterFormat":{},"inlines":[{"characterFormat":{"fontSize":10,"fontFamily":"Verdana","fontSizeBidi":10,"fontFamilyBidi":"Times New Roman"},"text":"80.368"}]}]},{"cellFormat":{"borders":{"top":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},"left":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},"right":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},"bottom":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},"diagonalDown":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0},"diagonalUp":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0},"horizontal":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0},"vertical":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0}},"shading":{"backgroundColor":"#FFFFFF","foregroundColor":"empty","textureStyle":"TextureNone"},"preferredWidth":37.720001220703125,"preferredWidthType":"Percent","cellWidth":117.95841899993776,"columnSpan":1,"rowSpan":1,"verticalAlignment":"Top"},"columnIndex":2}],
"rowFormat":{"height":14.399999618530273,"allowBreakAcrossPages":true,"heightType":"Exactly","isHeader":false,"borders":{"top":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},"left":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},"right":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},"bottom":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},"diagonalDown":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0},"diagonalUp":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0},"horizontal":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},"vertical":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0}},"gridBefore":0,"gridBeforeWidth":0,"gridBeforeWidthType":"Point","gridAfter":0,"gridAfterWidth":0,"gridAfterWidthType":"Point"}},{
"cells":[{"blocks":[{"paragraphFormat":{"styleName":"Normal","listFormat":{},"characterFormat":{},"inlines":[{"characterFormat":{"fontSize":10,"fontFamily":"Verdana","fontSizeBidi":10,"fontFamilyBidi":"Times New Roman"},"text":"3"}]}]},{"cellFormat":{"borders":{"top":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},"left":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},"right":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},"bottom":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},"diagonalDown":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0},"diagonalUp":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0},"horizontal":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0}},

```

```

"vertical":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineW
idth":0,"shadow":false,"space":0}},"shading":{"backgroundColor":"#D9E2F3FF",
"foregroundColor":"empty","textureStyle":"TextureNone"},"preferredWidth":13.
420000076293945,"preferredWidthType":"Percent","cellWidth":64.71214527422465
,"columnSpan":1,"rowSpan":1,"verticalAlignment":"Top"},"columnIndex":0},{
"blocks":[{"paragraphFormat":{"styleName":"Normal","listFormat":{}},
"characterFormat":{},"inlines":[{"characterFormat":{"fontSize":10,"fontFamily":"Verdana",
"fontSizeBidi":10,"fontFamilyBidi":"Times New Roman"},"text":"Raclette
Courdavault"}]}],"cellFormat":{"borders":{"top":{"color":"#8EAADBFF","hasNon
eStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0}
,"left":{"color":"#8EAADBFF","hasNoneStyle":false,"lineStyle":"Single","line
Width":0.5,"shadow":false,"space":0},"right":{"color":"#8EAADBFF","hasNoneSt
yle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},"b
ottom":{"color":"#8EAADBFF","hasNoneStyle":false,"lineStyle":"Single","lineW
idth":0.5,"shadow":false,"space":0},"diagonalDown":{"color":"#000000","hasNo
neStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0},"d
iagonalUp":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineW
idth":0,"shadow":false,"space":0},"horizontal":{"color":"#000000","hasNoneSt
yle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0},"verti
cal":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineWidth":
0,"shadow":false,"space":0}},"shading":{"backgroundColor":"#D9E2F3FF","foreg
roundColor":"empty","textureStyle":"TextureNone"},"preferredWidth":48.860000
61035156,"preferredWidthType":"Percent","cellWidth":292.87942351880633,"colu
mnSpan":1,"rowSpan":1,"verticalAlignment":"Top"},"columnIndex":1},{
"blocks":[{"paragraphFormat":{"styleName":"Normal","listFormat":{}},
"characterFormat":{},"inlines":[{"characterFormat":{"fontSize":10,"fontFamily":"Verdana",
"fontSizeBidi":10,"fontFamilyBidi":"Times New
Roman"},"text":"71.155"}]}],"cellFormat":{"borders":{"top":{"color":"#8EAADB
FF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false
,"space":0},"left":{"color":"#8EAADBFF","hasNoneStyle":false,"lineStyle":"Si
ngle","lineWidth":0.5,"shadow":false,"space":0},"right":{"color":"#8EAADBFF"
,"hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"s
pace":0},"bottom":{"color":"#8EAADBFF","hasNoneStyle":false,"lineStyle":"Sin
gle","lineWidth":0.5,"shadow":false,"space":0},"diagonalDown":{"color":"#000
000","hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"s
pace":0},"diagonalUp":{"color":"#000000","hasNoneStyle":false,"lineStyle":"N
one","lineWidth":0,"shadow":false,"space":0},"horizontal":{"color":"#000000"
,"hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space
":0},"vertical":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","
lineWidth":0,"shadow":false,"space":0}},"shading":{"backgroundColor":"#D9E2F
3FF","foregroundColor":"empty","textureStyle":"TextureNone"},"preferredWidth
":37.720001220703125,"preferredWidthType":"Percent","cellWidth":117.95841899
993776,"columnSpan":1,"rowSpan":1,"verticalAlignment":"Top"},"columnIndex":2
}],
"rowFormat":{"height":14.399999618530273,"allowBreakAcrossPages":true,"he
ightType":"Exactly","isHeader":false,"borders":{"top":{"color":"#8EAADBFF","
hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"spa
ce":0},"left":{"color":"#8EAADBFF","hasNoneStyle":false,"lineStyle":"Single"
,"lineWidth":0.5,"shadow":false,"space":0},"right":{"color":"#8EAADBFF","has
NoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space
":0},"bottom":{"color":"#8EAADBFF","hasNoneStyle":false,"lineStyle":"Single",
"lineWidth":0.5,"shadow":false,"space":0},"diagonalDown":{"color":"#000000",
"hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space
":0},"diagonalUp":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None",
"lineWidth":0,"shadow":false,"space":0},"horizontal":{"color":"#8EAADBFF","h
asNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"spac
e":0},"vertical":{"color":"#8EAADBFF","hasNoneStyle":false,"lineStyle":"Sing
le","lineWidth":0.5,"shadow":false,"space":0}},
"gridBefore":0,"gridBeforeWid

```



```

false,"lineStyle":"Single","lineWidth":0.5,"shadow>false,"spa
ce":0},"left":{"color":"#8EADBFF","hasNoneStyle>false,"lineStyle":"Single"
,"lineWidth":0.5,"shadow>false,"space":0},"right":{"color":"#8EADBFF","has
NoneStyle>false,"lineStyle":"Single","lineWidth":0.5,"shadow>false,"space"
:0},"bottom":{"color":"#8EADBFF","hasNoneStyle>false,"lineStyle":"Single",
"lineWidth":0.5,"shadow>false,"space":0},"diagonalDown":{"color":"#000000",
"hasNoneStyle>false,"lineStyle":"None","lineWidth":0,"shadow>false,"space"
:0},"diagonalUp":{"color":"#000000","hasNoneStyle>false,"lineStyle":"None",
"lineWidth":0,"shadow>false,"space":0},"horizontal":{"color":"#000000","has
NoneStyle>false,"lineStyle":"None","lineWidth":0,"shadow>false,"space":0},
"vertical":{"color":"#000000","hasNoneStyle>false,"lineStyle":"None","lineW
idth":0,"shadow>false,"space":0}},"shading":{"backgroundColor":"#FFFFFFF",
"foregroundColor":"empty","textureStyle":"TextureNone"},"preferredWidth":13.
420000076293945,"preferredWidthType":"Percent","cellWidth":64.71214527422465
,"columnSpan":1,"rowSpan":1,"verticalAlignment":"Top"},"columnIndex":0},{ "bl
ocks":[{"paragraphFormat":{"styleName":"Normal","listFormat":{}}, "characterF
ormat":{},"inlines":[{"characterFormat":{"fontSize":10,"fontFamily":"Verdana
","fontSizeBidi":10,"fontFamilyBidi":"Times New Roman"},"text":"Tarte au
sucr
e"}]}]},{"cellFormat":{"borders":{"top":{"color":"#8EADBFF","hasNoneStyle":fal
se,"lineStyle":"Single","lineWidth":0.5,"shadow>false,"space":0},"left":{"c
olor":"#8EADBFF","hasNoneStyle>false,"lineStyle":"Single","lineWidth":0.5,
"shadow>false,"space":0},"right":{"color":"#8EADBFF","hasNoneStyle>false,
"lineStyle":"Single","lineWidth":0.5,"shadow>false,"space":0},"bottom":{"co
lor":"#8EADBFF","hasNoneStyle>false,"lineStyle":"Single","lineWidth":0.5,"
shadow>false,"space":0},"diagonalDown":{"color":"#000000","hasNoneStyle":fa
lse,"lineStyle":"None","lineWidth":0,"shadow>false,"space":0},"diagonalUp":
{"color":"#000000","hasNoneStyle>false,"lineStyle":"None","lineWidth":0,"sh
adow>false,"space":0},"horizontal":{"color":"#000000","hasNoneStyle>false,
"lineStyle":"None","lineWidth":0,"shadow>false,"space":0},"vertical":{"colo
r":"#000000","hasNoneStyle>false,"lineStyle":"None","lineWidth":0,"shadow":
false,"space":0}},"shading":{"backgroundColor":"#FFFFFFF","foregroundColor"
:"empty","textureStyle":"TextureNone"},"preferredWidth":48.86000061035156,"p
REFERREDWidthType":"Percent","cellWidth":292.87942351880633,"columnSpan":1,"
rowSpan":1,"verticalAlignment":"Top"},"columnIndex":1},{ "blocks":[{"paragrap
hFormat":{"styleName":"Normal","listFormat":{}}, "characterFormat":{},"inline
s":[{"characterFormat":{"fontSize":10,"fontFamily":"Verdana","fontSizeBidi":
10,"fontFamilyBidi":"Times New
Roman"},"text":"47.234"}]}, {"characterFormat":{},"bookmarkType":1,"name": "_GoB
ack"}]}]},{"cellFormat":{"borders":{"top":{"color":"#8EADBFF","hasNoneStyle":
false,"lineStyle":"Single","lineWidth":0.5,"shadow>false,"space":0},"left":
{"color":"#8EADBFF","hasNoneStyle>false,"lineStyle":"Single","lineWidth":0
.5,"shadow":false,"space":0},"right":{"color":"#8EADBFF","hasNoneStyle":fal
se,"lineStyle":"Single","lineWidth":0.5,"shadow>false,"space":0},"bottom":{"
color":"#8EADBFF","hasNoneStyle>false,"lineStyle":"Single","lineWidth":0.
5,"shadow":false,"space":0},"diagonalDown":{"color":"#000000","hasNoneStyle"
:false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0},"diagonalU
p":{"color":"#000000","hasNoneStyle>false,"lineStyle":"None","lineWidth":0,
"shadow":false,"space":0},"horizontal":{"color":"#000000","hasNoneStyle":fal
se,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0},"vertical":{"c
olor":"#000000","hasNoneStyle>false,"lineStyle":"None","lineWidth":0,"shado

```

```

w":false,"space":0}},{"shading":{"backgroundColor":"#FFFFFFF","foregroundColor":"empty","textureStyle":"TextureNone"},"preferredWidth":37.720001220703125,"preferredWidthType":"Percent","cellWidth":117.95841899993776,"columnSpan":1,"rowSpan":1,"verticalAlignment":"Top"},"columnIndex":2}]],"rowFormat":{"height":14.399999618530273,"allowBreakAcrossPages":true,"heightType":"Exactly","isHeader":false,"borders":{"top":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},"left":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},"right":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},"bottom":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},"diagonalDown":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0},"diagonalUp":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0},"horizontal":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},"vertical":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0}},{"gridBefore":0,"gridBeforeWidth":0,"gridBeforeWidthType":"Point","gridAfter":0,"gridAfterWidth":0,"gridAfterWidthType":"Point"},"cells":[{"blocks":[{"paragraphFormat":{"styleName":"Normal","listFormat":{},"characterFormat":{},"inlines":[{"characterFormat":{},"bookmarkType":0,"name":"_GoBack"},{"characterFormat":{"fontSize":10,"fontFamily":"Verdana","fontSizeBidi":10,"fontFamilyBidi":"Times New Roman"},"text":"5"}]}]]},"cellFormat":{"borders":{"top":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},"left":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},"right":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},"bottom":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},"diagonalDown":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0},"diagonalUp":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0},"horizontal":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0},"vertical":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0}},{"shading":{"backgroundColor":"#D9E2F3FF","foregroundColor":"empty","textureStyle":"TextureNone"},"preferredWidth":13.420000076293945,"preferredWidthType":"Percent","cellWidth":64.71214527422465,"columnSpan":1,"rowSpan":1,"verticalAlignment":"Top"},"columnIndex":0},"blocks":[{"paragraphFormat":{"styleName":"Normal","listFormat":{},"characterFormat":{},"inlines":[{"characterFormat":{"fontSize":10,"fontFamily":"Verdana","fontSizeBidi":10,"fontFamilyBidi":"Times New Roman"},"text":"Camembert Pierrot"}]}]]},"cellFormat":{"borders":{"top":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},"left":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},"right":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},"bottom":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},"diagonalDown":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0},"diagonalUp":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0},"horizontal":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0},"vertical":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0}},{"shading":{"backgroundColor":"#D9E2F3FF","foregroundColor":"empty","textureStyle":"TextureNone"},"preferredWidth":48.86000061035156,"p

```



```

referredWidthType":"Percent","cellWidth":292.87942351880633,"columnSpan":1,"
rowSpan":1,"verticalAlignment":"Top"},"columnIndex":1},{ "blocks":[{"paragraph
hFormat":{"styleName":"Normal","listFormat":{},"characterFormat":{},"inlines
":[{"characterFormat":{"fontSize":10,"fontFamily":"Verdana","fontSizeBidi":
10,"fontFamilyBidi":"Times New
Roman"},"text":"46.825"}]}],"cellFormat":{"borders":{"top":{"color":"#8EAADB
FF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false
,"space":0},"left":{"color":"#8EAADBFF","hasNoneStyle":false,"lineStyle":"Si
ngle","lineWidth":0.5,"shadow":false,"space":0},"right":{"color":"#8EAADBFF"
,"hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"s
pace":0},"bottom":{"color":"#8EAADBFF","hasNoneStyle":false,"lineStyle":"Sin
gle","lineWidth":0.5,"shadow":false,"space":0},"diagonalDown":{"color":"#000
000","hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"s
pace":0},"diagonalUp":{"color":"#000000","hasNoneStyle":false,"lineStyle":"N
one","lineWidth":0,"shadow":false,"space":0},"horizontal":{"color":"#000000"
,"hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space
":0},"vertical":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","
lineWidth":0,"shadow":false,"space":0},"shading":{"backgroundColor":"#D9E2F
3FF","foregroundColor":"empty","textureStyle":"TextureNone"},"preferredWidth
":37.720001220703125,"preferredWidthType":"Percent","cellWidth":117.95841899
993776,"columnSpan":1,"rowSpan":1,"verticalAlignment":"Top"},"columnIndex":2
}],"rowFormat":{"height":14.399999618530273,"allowBreakAcrossPages":true,"he
ightType":"Exactly","isHeader":false,"borders":{"top":{"color":"#8EAADBFF","
hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"spa
ce":0},"left":{"color":"#8EAADBFF","hasNoneStyle":false,"lineStyle":"Single"
,"lineWidth":0.5,"shadow":false,"space":0},"right":{"color":"#8EAADBFF","has
NoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space"
:0},"bottom":{"color":"#8EAADBFF","hasNoneStyle":false,"lineStyle":"Single"
,"lineWidth":0.5,"shadow":false,"space":0},"diagonalDown":{"color":"#000000"
,"hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space
":0},"diagonalUp":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None"
,"lineWidth":0,"shadow":false,"space":0},"horizontal":{"color":"#8EAADBFF","h
asNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"spac
e":0},"vertical":{"color":"#8EAADBFF","hasNoneStyle":false,"lineStyle":"Sing
le","lineWidth":0.5,"shadow":false,"space":0},"gridBefore":0,"gridBeforeWid
th":0,"gridBeforeWidthType":"Point","gridAfter":0,"gridAfterWidth":0,"gridAf
terWidthType":"Point"}},{ "cells":[{"blocks":[{"paragraphFormat":{"styleName"
:"Normal","listFormat":{},"characterFormat":{},"inlines":[{"characterFormat
":{"fontSize":10,"fontFamily":"Verdana","fontSizeBidi":10,"fontFamilyBidi":"
Times New
Roman"},"text":"6"}]}],"cellFormat":{"borders":{"top":{"color":"#8EAADBFF","
hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"spa
ce":0},"left":{"color":"#8EAADBFF","hasNoneStyle":false,"lineStyle":"Single"
,"lineWidth":0.5,"shadow":false,"space":0},"right":{"color":"#8EAADBFF","has
NoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space"
:0},"bottom":{"color":"#8EAADBFF","hasNoneStyle":false,"lineStyle":"Single"
,"lineWidth":0.5,"shadow":false,"space":0},"diagonalDown":{"color":"#000000"
,"hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space
":0},"diagonalUp":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None"
,"lineWidth":0,"shadow":false,"space":0},"horizontal":{"color":"#000000","has
NoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0},
"vertical":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineW
idth":0,"shadow":false,"space":0},"shading":{"backgroundColor":"#FFFFFFF","
foregroundColor":"empty","textureStyle":"TextureNone"},"preferredWidth":13.
420000076293945,"preferredWidthType":"Percent","cellWidth":64.71214527422465
,"columnSpan":1,"rowSpan":1,"verticalAlignment":"Top"},"columnIndex":0},{ "bl
ocks":[{"paragraphFormat":{"styleName":"Normal","listFormat":{},"characterF

```

```

ormat":{}, "inlines": [{"characterFormat": {"fontSize": 10, "fontFamily": "Verdana", "fontSizeBidi": 10, "fontFamilyBidi": "Times New Roman"}, "text": "Gnocchi di nonna Alice"}]], "cellFormat": {"borders": {"top": {"color": "#8EAADBFF", "hasNoneStyle": false, "lineStyle": "Single", "lineWidth": 0.5, "shadow": false, "space": 0}, "left": {"color": "#8EAADBFF", "hasNoneStyle": false, "lineStyle": "Single", "lineWidth": 0.5, "shadow": false, "space": 0}, "right": {"color": "#8EAADBFF", "hasNoneStyle": false, "lineStyle": "Single", "lineWidth": 0.5, "shadow": false, "space": 0}, "bottom": {"color": "#8EAADBFF", "hasNoneStyle": false, "lineStyle": "Single", "lineWidth": 0.5, "shadow": false, "space": 0}, "diagonalDown": {"color": "#000000", "hasNoneStyle": false, "lineStyle": "None", "lineWidth": 0, "shadow": false, "space": 0}, "diagonalUp": {"color": "#000000", "hasNoneStyle": false, "lineStyle": "None", "lineWidth": 0, "shadow": false, "space": 0}, "horizontal": {"color": "#000000", "hasNoneStyle": false, "lineStyle": "None", "lineWidth": 0, "shadow": false, "space": 0}, "vertical": {"color": "#000000", "hasNoneStyle": false, "lineStyle": "None", "lineWidth": 0, "shadow": false, "space": 0}}, "shading": {"backgroundColor": "#FFFFFF", "foregroundColor": "empty", "textureStyle": "TextureNone"}, "preferredWidth": 48.86000061035156, "preferredWidthType": "Percent", "cellWidth": 292.87942351880633, "columnSpan": 1, "rowSpan": 1, "verticalAlignment": "Top", "columnIndex": 1}, {"blocks": [{"paragraphFormat": {"styleName": "Normal", "listFormat": {}}, "characterFormat": {}, "inlines": [{"characterFormat": {"fontSize": 10, "fontFamily": "Verdana", "fontSizeBidi": 10, "fontFamilyBidi": "Times New Roman"}, "text": "42.593"}]]}, {"cellFormat": {"borders": {"top": {"color": "#8EAADBFF", "hasNoneStyle": false, "lineStyle": "Single", "lineWidth": 0.5, "shadow": false, "space": 0}, "left": {"color": "#8EAADBFF", "hasNoneStyle": false, "lineStyle": "Single", "lineWidth": 0.5, "shadow": false, "space": 0}, "right": {"color": "#8EAADBFF", "hasNoneStyle": false, "lineStyle": "Single", "lineWidth": 0.5, "shadow": false, "space": 0}, "bottom": {"color": "#8EAADBFF", "hasNoneStyle": false, "lineStyle": "Single", "lineWidth": 0.5, "shadow": false, "space": 0}, "diagonalDown": {"color": "#000000", "hasNoneStyle": false, "lineStyle": "None", "lineWidth": 0, "shadow": false, "space": 0}, "diagonalUp": {"color": "#000000", "hasNoneStyle": false, "lineStyle": "None", "lineWidth": 0, "shadow": false, "space": 0}, "horizontal": {"color": "#000000", "hasNoneStyle": false, "lineStyle": "None", "lineWidth": 0, "shadow": false, "space": 0}, "vertical": {"color": "#000000", "hasNoneStyle": false, "lineStyle": "None", "lineWidth": 0, "shadow": false, "space": 0}}, "shading": {"backgroundColor": "#FFFFFF", "foregroundColor": "empty", "textureStyle": "TextureNone"}, "preferredWidth": 37.720001220703125, "preferredWidthType": "Percent", "cellWidth": 117.95841899993776, "columnSpan": 1, "rowSpan": 1, "verticalAlignment": "Top", "columnIndex": 2}], "rowFormat": {"height": 14.3999999618530273, "allowBreakAcrossPages": true, "heightType": "Exactly", "isHeader": false, "borders": {"top": {"color": "#8EAADBFF", "hasNoneStyle": false, "lineStyle": "Single", "lineWidth": 0.5, "shadow": false, "space": 0}, "left": {"color": "#8EAADBFF", "hasNoneStyle": false, "lineStyle": "Single", "lineWidth": 0.5, "shadow": false, "space": 0}, "right": {"color": "#8EAADBFF", "hasNoneStyle": false, "lineStyle": "Single", "lineWidth": 0.5, "shadow": false, "space": 0}, "bottom": {"color": "#8EAADBFF", "hasNoneStyle": false, "lineStyle": "Single", "lineWidth": 0.5, "shadow": false, "space": 0}, "diagonalDown": {"color": "#000000", "hasNoneStyle": false, "lineStyle": "None", "lineWidth": 0, "shadow": false, "space": 0}, "diagonalUp": {"color": "#000000", "hasNoneStyle": false, "lineStyle": "None", "lineWidth": 0, "shadow": false, "space": 0}, "horizontal": {"color": "#8EAADBFF", "hasNoneStyle": false, "lineStyle": "Single", "lineWidth": 0.5, "shadow": false, "space": 0}, "vertical": {"color": "#8EAADBFF", "hasNoneStyle": false, "lineStyle": "Single", "lineWidth": 0.5, "shadow": false, "space": 0}}, "gridBefore": 0, "gridBeforeWidth": 0, "gridBeforeWidthType": "Point", "gridAfter": 0, "gridAfterWidth": 0, "gridAfterWidthType": "Point"}, {"cells": [{"blocks": [{"paragraphFormat": {"styleName": "Normal", "listFormat": {}}, "characterFormat": {}, "inlines": [{"characterFormat": {"fontSize": 10, "fontFamily": "Verdana", "fontSizeBidi": 10, "fontFamilyBidi": "Times New

```

```

Roman"}], "text": "7"}]]], "cellFormat": {"borders": {"top": {"color": "#8EAADBFF", "hasNoneStyle": false, "lineStyle": "Single", "lineWidth": 0.5, "shadow": false, "space": 0}, "left": {"color": "#8EAADBFF", "hasNoneStyle": false, "lineStyle": "Single", "lineWidth": 0.5, "shadow": false, "space": 0}, "right": {"color": "#8EAADBFF", "hasNoneStyle": false, "lineStyle": "Single", "lineWidth": 0.5, "shadow": false, "space": 0}, "bottom": {"color": "#8EAADBFF", "hasNoneStyle": false, "lineStyle": "Single", "lineWidth": 0.5, "shadow": false, "space": 0}, "diagonalDown": {"color": "#000000", "hasNoneStyle": false, "lineStyle": "None", "lineWidth": 0, "shadow": false, "space": 0}, "diagonalUp": {"color": "#000000", "hasNoneStyle": false, "lineStyle": "None", "lineWidth": 0, "shadow": false, "space": 0}, "horizontal": {"color": "#000000", "hasNoneStyle": false, "lineStyle": "None", "lineWidth": 0, "shadow": false, "space": 0}, "vertical": {"color": "#000000", "hasNoneStyle": false, "lineStyle": "None", "lineWidth": 0, "shadow": false, "space": 0}}, "shading": {"backgroundColor": "#D9E2F3FF", "foregroundColor": "empty", "textureStyle": "TextureNone"}, "preferredWidth": 13.420000076293945, "preferredWidthType": "Percent", "cellWidth": 64.71214527422465, "columnSpan": 1, "rowSpan": 1, "verticalAlignment": "Top"}, {"blocks": [{"paragraphFormat": {"styleName": "Normal", "listFormat": {}}, "characterFormat": {}, "inlines": [{"characterFormat": {"fontSize": 10, "fontFamily": "Verdana", "fontSizeBidi": 10, "fontFamilyBidi": "Times New Roman"}, "text": "Manjimup Dried Apples"}]]], "cellFormat": {"borders": {"top": {"color": "#8EAADBFF", "hasNoneStyle": false, "lineStyle": "Single", "lineWidth": 0.5, "shadow": false, "space": 0}, "left": {"color": "#8EAADBFF", "hasNoneStyle": false, "lineStyle": "Single", "lineWidth": 0.5, "shadow": false, "space": 0}, "right": {"color": "#8EAADBFF", "hasNoneStyle": false, "lineStyle": "Single", "lineWidth": 0.5, "shadow": false, "space": 0}, "bottom": {"color": "#8EAADBFF", "hasNoneStyle": false, "lineStyle": "Single", "lineWidth": 0.5, "shadow": false, "space": 0}, "diagonalDown": {"color": "#000000", "hasNoneStyle": false, "lineStyle": "None", "lineWidth": 0, "shadow": false, "space": 0}, "diagonalUp": {"color": "#000000", "hasNoneStyle": false, "lineStyle": "None", "lineWidth": 0, "shadow": false, "space": 0}, "horizontal": {"color": "#000000", "hasNoneStyle": false, "lineStyle": "None", "lineWidth": 0, "shadow": false, "space": 0}, "vertical": {"color": "#000000", "hasNoneStyle": false, "lineStyle": "None", "lineWidth": 0, "shadow": false, "space": 0}}, "shading": {"backgroundColor": "#D9E2F3FF", "foregroundColor": "empty", "textureStyle": "TextureNone"}, "preferredWidth": 48.86000061035156, "preferredWidthType": "Percent", "cellWidth": 292.87942351880633, "columnSpan": 1, "rowSpan": 1, "verticalAlignment": "Top"}, {"blocks": [{"paragraphFormat": {"styleName": "Normal", "listFormat": {}}, "characterFormat": {}, "inlines": [{"characterFormat": {"fontSize": 10, "fontFamily": "Verdana", "fontSizeBidi": 10, "fontFamilyBidi": "Times New Roman"}, "text": "41.819"}]]], "cellFormat": {"borders": {"top": {"color": "#8EAADBFF", "hasNoneStyle": false, "lineStyle": "Single", "lineWidth": 0.5, "shadow": false, "space": 0}, "left": {"color": "#8EAADBFF", "hasNoneStyle": false, "lineStyle": "Single", "lineWidth": 0.5, "shadow": false, "space": 0}, "right": {"color": "#8EAADBFF", "hasNoneStyle": false, "lineStyle": "Single", "lineWidth": 0.5, "shadow": false, "space": 0}, "bottom": {"color": "#8EAADBFF", "hasNoneStyle": false, "lineStyle": "Single", "lineWidth": 0.5, "shadow": false, "space": 0}, "diagonalDown": {"color": "#000000", "hasNoneStyle": false, "lineStyle": "None", "lineWidth": 0, "shadow": false, "space": 0}, "diagonalUp": {"color": "#000000", "hasNoneStyle": false, "lineStyle": "None", "lineWidth": 0, "shadow": false, "space": 0}, "horizontal": {"color": "#000000", "hasNoneStyle": false, "lineStyle": "None", "lineWidth": 0, "shadow": false, "space": 0}, "vertical": {"color": "#000000", "hasNoneStyle": false, "lineStyle": "None", "lineWidth": 0, "shadow": false, "space": 0}}, "shading": {"backgroundColor": "#D9E2F3FF", "foregroundColor": "empty", "textureStyle": "TextureNone"}, "preferredWidth": 37.720001220703125, "preferredWidthType": "Percent", "cellWidth": 117.95841899993776, "columnSpan": 1, "rowSpan": 1, "verticalAlignment": "Top"}, {"height": 14.3999999618530273, "allowBreakAcrossPages": true, "heightType": "Exactly", "isHeader": false, "borders": {"top": {"color": "#8EAADBFF", "

```

```

hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},
"left":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},
"right":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},
"bottom":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},
"diagonalDown":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0},
"diagonalUp":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0},
"horizontal":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},
"vertical":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0}},
"gridBefore":0,"gridBeforeWidth":0,"gridAfterWidth":0,"gridAfterWidthType":"Point"},
{"cells":[{"blocks":[{"paragraphFormat":{"styleName":"Normal","listFormat":{},"characterFormat":{"fontSize":10,"fontFamily":"Verdana","fontSizeBidi":10,"fontFamilyBidi":"Times New Roman"},"text":"8"}]}],
"cellFormat":{"borders":{"top":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},
"left":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},
"right":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},
"bottom":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},
"diagonalDown":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0},
"diagonalUp":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0},
"horizontal":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0},
"vertical":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0}},
"shading":{"backgroundColor":"#FFFFFF","foregroundColor":"empty","textureStyle":"TextureNone"},"preferredWidth":13.420000076293945,"preferredWidthType":"Percent","cellWidth":64.71214527422465,"columnSpan":1,"rowSpan":1,"verticalAlignment":"Top"},
{"blocks":[{"paragraphFormat":{"styleName":"Normal","listFormat":{},"characterFormat":{"fontSize":10,"fontFamily":"Verdana","fontSizeBidi":10,"fontFamilyBidi":"Times New Roman"},"text":"Alice Mutton"}]}],
"cellFormat":{"borders":{"top":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},
"left":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},
"right":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},
"bottom":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},
"diagonalDown":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0},
"diagonalUp":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0},
"horizontal":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0},
"vertical":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0}},
"shading":{"backgroundColor":"#FFFFFF","foregroundColor":"empty","textureStyle":"TextureNone"},"preferredWidth":48.86000061035156,"preferredWidthType":"Percent","cellWidth":292.87942351880633,"columnSpan":1,"rowSpan":1,"verticalAlignment":"Top"},
{"blocks":[{"paragraphFormat":{"styleName":"Normal","listFormat":{},"characterFormat":{"fontSize":10,"fontFamily":"Verdana","fontSizeBidi":10,"fontFamilyBidi":"Times New Roman"},"text":"32.698"}]}],
"cellFormat":{"borders":{"top":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false

```

```

"space":0}, {"color":"#8EAADBFF", "hasNoneStyle":false, "lineStyle":"Single", "lineWidth":0.5, "shadow":false, "space":0}, {"right":{"color":"#8EAADBFF", "hasNoneStyle":false, "lineStyle":"Single", "lineWidth":0.5, "shadow":false, "space":0}, {"bottom":{"color":"#8EAADBFF", "hasNoneStyle":false, "lineStyle":"Single", "lineWidth":0.5, "shadow":false, "space":0}, {"diagonalDown":{"color":"#000000", "hasNoneStyle":false, "lineStyle":"None", "lineWidth":0, "shadow":false, "space":0}, {"diagonalUp":{"color":"#000000", "hasNoneStyle":false, "lineStyle":"None", "lineWidth":0, "shadow":false, "space":0}, {"horizontal":{"color":"#000000", "hasNoneStyle":false, "lineStyle":"None", "lineWidth":0, "shadow":false, "space":0}, {"vertical":{"color":"#000000", "hasNoneStyle":false, "lineStyle":"None", "lineWidth":0, "shadow":false, "space":0}}, {"shading":{"backgroundColor":"#FFFFFF", "foregroundColor":"empty", "textureStyle":"TextureNone"}}, {"preferredWidth":37.720001220703125, "preferredWidthType":"Percent", "cellWidth":117.95841899993776, "columnSpan":1, "rowSpan":1, "verticalAlignment":"Top"}, {"columnIndex":2}], {"rowFormat":{"height":14.399999618530273, "allowBreakAcrossPages":true, "heightType":"Exactly", "isHeader":false, "borders":{"top":{"color":"#8EAADBFF", "hasNoneStyle":false, "lineStyle":"Single", "lineWidth":0.5, "shadow":false, "space":0}, {"left":{"color":"#8EAADBFF", "hasNoneStyle":false, "lineStyle":"Single", "lineWidth":0.5, "shadow":false, "space":0}, {"right":{"color":"#8EAADBFF", "hasNoneStyle":false, "lineStyle":"Single", "lineWidth":0.5, "shadow":false, "space":0}, {"bottom":{"color":"#8EAADBFF", "hasNoneStyle":false, "lineStyle":"Single", "lineWidth":0.5, "shadow":false, "space":0}, {"diagonalDown":{"color":"#000000", "hasNoneStyle":false, "lineStyle":"None", "lineWidth":0, "shadow":false, "space":0}, {"diagonalUp":{"color":"#000000", "hasNoneStyle":false, "lineStyle":"None", "lineWidth":0, "shadow":false, "space":0}, {"horizontal":{"color":"#8EAADBFF", "hasNoneStyle":false, "lineStyle":"Single", "lineWidth":0.5, "shadow":false, "space":0}, {"vertical":{"color":"#8EAADBFF", "hasNoneStyle":false, "lineStyle":"Single", "lineWidth":0.5, "shadow":false, "space":0}}, {"gridBefore":0, "gridBeforeWidth":0, "gridBeforeWidthType":"Point", "gridAfter":0, "gridAfterWidth":0, "gridAfterWidthType":"Point"}}, {"cells":[{"blocks":[{"paragraphFormat":{"styleName":"Normal", "listFormat":{}}, {"characterFormat":{}}, {"inlines":[{"characterFormat":{"fontSize":10, "fontFamily":"Verdana", "fontSizeBidi":10, "fontFamilyBidi":"Times New Roman"}}, {"text":"9"}]}]}], {"cellFormat":{"borders":{"top":{"color":"#8EAADBFF", "hasNoneStyle":false, "lineStyle":"Single", "lineWidth":0.5, "shadow":false, "space":0}, {"left":{"color":"#8EAADBFF", "hasNoneStyle":false, "lineStyle":"Single", "lineWidth":0.5, "shadow":false, "space":0}, {"right":{"color":"#8EAADBFF", "hasNoneStyle":false, "lineStyle":"Single", "lineWidth":0.5, "shadow":false, "space":0}, {"bottom":{"color":"#8EAADBFF", "hasNoneStyle":false, "lineStyle":"Single", "lineWidth":0.5, "shadow":false, "space":0}, {"diagonalDown":{"color":"#000000", "hasNoneStyle":false, "lineStyle":"None", "lineWidth":0, "shadow":false, "space":0}, {"diagonalUp":{"color":"#000000", "hasNoneStyle":false, "lineStyle":"None", "lineWidth":0, "shadow":false, "space":0}, {"horizontal":{"color":"#000000", "hasNoneStyle":false, "lineStyle":"None", "lineWidth":0, "shadow":false, "space":0}, {"vertical":{"color":"#000000", "hasNoneStyle":false, "lineStyle":"None", "lineWidth":0, "shadow":false, "space":0}}, {"shading":{"backgroundColor":"#D9E2F3FF", "foregroundColor":"empty", "textureStyle":"TextureNone"}}, {"preferredWidth":13.420000076293945, "preferredWidthType":"Percent", "cellWidth":64.71214527422465, "columnSpan":1, "rowSpan":1, "verticalAlignment":"Top"}, {"columnIndex":0}, {"blocks":[{"paragraphFormat":{"styleName":"Normal", "listFormat":{}}, {"characterFormat":{}}, {"inlines":[{"characterFormat":{"fontSize":10, "fontFamily":"Verdana", "fontSizeBidi":10, "fontFamilyBidi":"Times New Roman"}}, {"text":"Carnarvon Tigers"}]}]}], {"cellFormat":{"borders":{"top":{"color":"#8EAADBFF", "hasNoneStyle":false, "lineStyle":"Single", "lineWidth":0.5, "shadow":false, "space":0}, {"left":{"color":"#8EAADBFF", "hasNoneStyle":false, "lineStyle":"Single", "lineWidth":0.5, "shadow":false, "space":0}, {"right":{"color":"#8EAADBFF", "hasNoneStyle":false, "lineStyle":"Single", "lineWidth":0.5, "shadow":false, "space":0}, {"bottom":

```

```

":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},"diagonalDown":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0},"diagonalUp":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0},"horizontal":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0},"vertical":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0},"shading":{"backgroundColor":"#D9E2F3FF","foregroundColor":"empty","textureStyle":"TextureNone"},"preferredWidth":48.86000061035156,"preferredWidthType":"Percent","cellWidth":292.87942351880633,"columnSpan":1,"rowSpan":1,"verticalAlignment":"Top"},"columnIndex":1},{ "blocks":[{"paragraphFormat":{"styleName":"Normal","listFormat":{},"characterFormat":{},"inlines":[{"characterFormat":{"fontSize":10,"fontFamily":"Verdana","fontSizeBidi":10,"fontFamilyBidi":"Times New Roman"},"text":"29.171"}]}]},"cellFormat":{"borders":{"top":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},"left":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},"right":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},"bottom":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},"diagonalDown":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0},"diagonalUp":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0},"horizontal":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0},"vertical":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0},"shading":{"backgroundColor":"#D9E2F3FF","foregroundColor":"empty","textureStyle":"TextureNone"},"preferredWidth":37.720001220703125,"preferredWidthType":"Percent","cellWidth":117.95841899993776,"columnSpan":1,"rowSpan":1,"verticalAlignment":"Top"},"columnIndex":2}],"rowFormat":{"height":14.399999618530273,"allowBreakAcrossPages":true,"rightType":"Exactly","isHeader":false,"borders":{"top":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},"left":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},"right":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},"bottom":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},"diagonalDown":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0},"diagonalUp":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0},"horizontal":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},"vertical":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},"gridBefore":0,"gridBeforeWidth":0,"gridBeforeWidthType":"Point","gridAfter":0,"gridAfterWidth":0,"gridAfterWidthType":"Point"}},{ "cells":[{"blocks":[{"paragraphFormat":{"styleName":"Normal","listFormat":{},"characterFormat":{},"inlines":[{"characterFormat":{"fontSize":10,"fontFamily":"Verdana","fontSizeBidi":10,"fontFamilyBidi":"Times New Roman"},"text":"10"}]}]}]},"cellFormat":{"borders":{"top":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},"left":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},"right":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},"bottom":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},"diagonalDown":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0}}}}

```



```

":0},"diagonalUp":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None",
,"lineWidth":0,"shadow":false,"space":0},"horizontal":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0},
,"vertical":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0}},"shading":{"backgroundColor":"#FFFFFFF",
,"foregroundColor":"empty","textureStyle":"TextureNone"},"preferredWidth":13.420000076293945,"preferredWidthType":"Percent","cellWidth":64.71214527422465,"columnSpan":1,"rowSpan":1,"verticalAlignment":"Top"},"columnIndex":0},{
"blocks":[{"paragraphFormat":{"styleName":"Normal","listFormat":{},"characterFormat":{},"inlines":[{"characterFormat":{"fontSize":10,"fontFamily":"Verdana",
"fontSizeBidi":10,"fontFamilyBidi":"Times New Roman"},"text":"Rössle Sauerkraut."}]}]},"cellFormat":{"borders":{"top":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single",
"lineWidth":0.5,"shadow":false,"space":0},"left":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single",
"lineWidth":0.5,"shadow":false,"space":0},"right":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single",
"lineWidth":0.5,"shadow":false,"space":0},"bottom":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single",
"lineWidth":0.5,"shadow":false,"space":0},"diagonalDown":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None",
"lineWidth":0,"shadow":false,"space":0},"diagonalUp":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None",
"lineWidth":0,"shadow":false,"space":0},"horizontal":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None",
"lineWidth":0,"shadow":false,"space":0},"vertical":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None",
"lineWidth":0,"shadow":false,"space":0}},"shading":{"backgroundColor":"#FFFFFFF",
,"foregroundColor":"empty","textureStyle":"TextureNone"},"preferredWidth":48.86000061035156,"preferredWidthType":"Percent",
"cellWidth":292.87942351880633,"columnSpan":1,"rowSpan":1,"verticalAlignment":"Top"},"columnIndex":1},{
"blocks":[{"paragraphFormat":{"styleName":"Normal","listFormat":{},"characterFormat":{},"inlines":[{"characterFormat":{"fontSize":10,"fontFamily":"Verdana",
"fontSizeBidi":10,"fontFamilyBidi":"Times New Roman"},"text":"25.696"}]}]},"cellFormat":{"borders":{"top":{"color":"#8EADBFF",
"hasNoneStyle":false,"lineStyle":"Single",
"lineWidth":0.5,"shadow":false,"space":0},"left":{"color":"#8EADBFF",
"hasNoneStyle":false,"lineStyle":"Single",
"lineWidth":0.5,"shadow":false,"space":0},"right":{"color":"#8EADBFF",
"hasNoneStyle":false,"lineStyle":"Single",
"lineWidth":0.5,"shadow":false,"space":0},"bottom":{"color":"#8EADBFF",
"hasNoneStyle":false,"lineStyle":"Single",
"lineWidth":0.5,"shadow":false,"space":0},"diagonalDown":{"color":"#000000",
"hasNoneStyle":false,"lineStyle":"None",
"lineWidth":0,"shadow":false,"space":0},"diagonalUp":{"color":"#000000",
"hasNoneStyle":false,"lineStyle":"None",
"lineWidth":0,"shadow":false,"space":0},"horizontal":{"color":"#000000",
"hasNoneStyle":false,"lineStyle":"None",
"lineWidth":0,"shadow":false,"space":0},"vertical":{"color":"#000000",
"hasNoneStyle":false,"lineStyle":"None",
"lineWidth":0,"shadow":false,"space":0}},"shading":{"backgroundColor":"#FFFFFFF",
,"foregroundColor":"empty","textureStyle":"TextureNone"},"preferredWidth":37.720001220703125,"preferredWidthType":"Percent",
"cellWidth":117.9584189993776,"columnSpan":1,"rowSpan":1,"verticalAlignment":"Top"},"columnIndex":2}],
"rowFormat":{"height":14.399999618530273,"allowBreakAcrossPages":true,"rightType":"Exactly","isHeader":false,"borders":{"top":{"color":"#8EADBFF",
"hasNoneStyle":false,"lineStyle":"Single",
"lineWidth":0.5,"shadow":false,"space":0},"left":{"color":"#8EADBFF",
"hasNoneStyle":false,"lineStyle":"Single",
"lineWidth":0.5,"shadow":false,"space":0},"right":{"color":"#8EADBFF",
"hasNoneStyle":false,"lineStyle":"Single",
"lineWidth":0.5,"shadow":false,"space":0},"bottom":{"color":"#8EADBFF",
"hasNoneStyle":false,"lineStyle":"Single",
"lineWidth":0.5,"shadow":false,"space":0},"diagonalDown":{"color":"#000000",
"hasNoneStyle":false,"lineStyle":"None",
"lineWidth":0,"shadow":false,"space":0},"diagonalUp":{"color":"#000000",
"hasNoneStyle":false,"lineStyle":"None",
"lineWidth":0,"shadow":false,"space":0},"horizontal":{"color":"#8EADBFF",

```

```

NoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},"vertical":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0}},"gridBefore":0,"gridBeforeWidth":0,"gridBeforeWidthType":"Point","gridAfter":0,"gridAfterWidth":0,"gridAfterWidthType":"Point"}]],"grid":[64.71214527422465,292.87942351880633,117.95841899993776],"tableFormat":{"borders":{"top":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},"left":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},"right":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},"bottom":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},"diagonalDown":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0},"diagonalUp":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0},"horizontal":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},"vertical":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0}},"shading":{"backgroundColor":"#FFFFFF","foregroundColor":"empty","textureStyle":"TextureNone"},"cellSpacing":0,"leftIndent":0,"tableAlignment":"Left","topMargin":0,"rightMargin":0.5,"leftMargin":0.5,"bottomMargin":0,"preferredWidth":475.54998779296875,"preferredWidthType":"Point","bidi":false,"allowAutoFit":true},"description":null,"title":null},{ "paragraphFormat":{"afterSpacing":0,"styleName":"Normal","listFormat":{},"characterFormat":{"fontFamily":"Calibri","fontColor":"#000000FF","fontFamilyBidi":"Calibri"},"inlines":[]},{ "paragraphFormat":{"afterSpacing":0,"styleName":"Normal","listFormat":{},"characterFormat":{"inlines":{"characterFormat":{"fontSize":10,"fontFamily":"Verdana","fontSizeBidi":10,"fontFamilyBidi":"Times New Roman"},"text":"The best-selling product of the company is Cote de Blaye, being part of the Beverages"}},{ "characterFormat":{"fontSize":10,"fontFamily":"Verdana","fontSizeBidi":10,"fontFamilyBidi":"Times New Roman"},"text":"category. The contribution of this product to the sum of our sales is $141.396."}}},{ "paragraphFormat":{"afterSpacing":0,"lineSpacing":1,"lineSpacingType":"Multiple","styleName":"Normal","listFormat":{},"characterFormat":{"fontSize":10,"fontFamily":"Verdana","fontSizeBidi":10,"fontFamilyBidi":"Times New Roman"},"inlines":[]},{ "paragraphFormat":{"afterSpacing":0,"lineSpacing":1,"lineSpacingType":"Multiple","styleName":"Normal","listFormat":{},"characterFormat":{"fontSize":10,"fontFamily":"Verdana","fontSizeBidi":10,"fontFamilyBidi":"Times New Roman"},"inlines":[]},{ "paragraphFormat":{"styleName":"Normal","listFormat":{},"characterFormat":{},"inlines":[{"characterFormat":{},"chartLegend":{"position":"Right","chartTitleArea":{"fontName":"+mn-1t","fontSize":9,"layout":{"layoutX":0,"layoutY":0},"dataFormat":{"fill":{"foreColor":"000000","rgb":"#000000"},"line":{"color":"808080","rgb":"#808080"}}}},"chartTitleArea":{"fontName":"+mn-1t","fontSize":14,"layout":{"layoutX":0,"layoutY":0},"dataFormat":{"fill":{"foreColor":"000000","rgb":"#000000"},"line":{"color":"000000","rgb":"#000000"}}},"chartArea":{"foreColor":"#FFFFFFF","plotArea":{"foreColor":"#000000FF"},"chartCategory":[{"chartData":[{"yValue":141.396}],"categoryXName":"Côte de Blaye"}, {"chartData":[{"yValue":80.368}],"categoryXName":"Thüringer Rostbratwurst"}, {"chartData":[{"yValue":71.155}],"categoryXName":"Raclette Courdavault"}, {"chartData":[{"yValue":47.234}],"categoryXName":"Tarte au sucre"}, {"chartData":[{"yValue":46.825}],"categoryXName":"Camembert Pierrot"}, {"chartData":[{"yValue":42.593}],"categoryXName":"Gnocchi di nonna Alice"}, {"chartData":[{"yValue":41.819}],"categoryXName":"Manjimup Dried Apples"}, {"chartData":[{"yValue":32.698}],"categoryXName":"Alice

```



```

Mutton"}, {"chartData": [{"yValue": 29.171}], "categoryXName": "Carnarvon
Tigers"}, {"chartData": [{"yValue": 25.696}], "categoryXName": "Rössle
Sauerkraut"}], "chartSeries": [{"dataPoints": [{"fill": {"foreColor": "4472c4", "r
gb": "#4472c4"}, "line": {"color": "ffffff", "rgb": "#ffffff"}}, {"fill": {"foreColo
r": "ed7d31", "rgb": "#ed7d31"}, "line": {"color": "ffffff", "rgb": "#ffffff"}}, {"fi
ll": {"foreColor": "a5a5a5", "rgb": "#a5a5a5"}, "line": {"color": "ffffff", "rgb": "#
ffffff"}}, {"fill": {"foreColor": "ffc000", "rgb": "#ffc000"}, "line": {"color": "ff
ffff", "rgb": "#ffffff"}}, {"fill": {"foreColor": "5b9bd5", "rgb": "#5b9bd5"}, "line
": {"color": "ffffff", "rgb": "#ffffff"}}, {"fill": {"foreColor": "70ad47", "rgb": "#
70ad47"}, "line": {"color": "ffffff", "rgb": "#ffffff"}}, {"fill": {"foreColor": "26
4379", "rgb": "#264379"}, "line": {"color": "ffffff", "rgb": "#ffffff"}}, {"fill": {"
foreColor": "9f480e", "rgb": "#9f480e"}, "line": {"color": "ffffff", "rgb": "#ffffff
"}, {"fill": {"foreColor": "636363", "rgb": "#636363"}, "line": {"color": "ffffff",
"rgb": "#ffffff"}}, {"fill": {"foreColor": "9a7200", "rgb": "#9a7200"}, "line": {"co
lor": "ffffff", "rgb": "#ffffff"}}], "seriesName": "Sales"}], "chartPrimaryCategor
yAxis": {"chartTitle": null, "chartTitleArea": {"layout": {}, "dataFormat": {"fill"
: {}, "line": {}}, "categoryType": "Automatic", "fontSize": 11, "fontName": "Calibri
", "numberFormat": "General", "maximumValue": 0, "minimumValue": 0, "majorUnit": 0, "
hasMajorGridLines": false, "hasMinorGridLines": false, "majorTickMark": "TickMark
_Outside", "minorTickMark": "TickMark_None", "tickLabelPosition": "TickLabelPosi
tion_NextToAxis"}, "chartPrimaryValueAxis": {"chartTitle": null, "chartTitleArea
": {"layout": {}, "dataFormat": {"fill": {}, "line": {}}, "fontSize": 11, "fontName":
"Calibri", "maximumValue": 0, "minimumValue": 0, "majorUnit": 0, "hasMajorGridLines
": false, "hasMinorGridLines": false, "majorTickMark": "TickMark_Outside", "minorT
ickMark": "TickMark_None", "tickLabelPosition": "TickLabelPosition_NextToAxis"}
, "chartTitle": "Best Selling
Products", "chartType": "Pie", "gapWidth": 0, "overlap": 0, "height": 225, "width": 43
2}}], {"paragraphFormat": {"styleName": "Normal", "listFormat": {}}, "characterFor
mat": {}, "inlines": []}, {"paragraphFormat": {"afterSpacing": 0, "lineSpacing": 1, "
lineSpacingType": "Multiple", "styleName": "Normal", "listFormat": {}}, "character
Format": {"fontSize": 10, "fontFamily": "Verdana", "fontSizeBidi": 10, "fontFamilyB
idi": "Verdana"}, "inlines": [{"characterFormat": {"fontSize": 10, "fontFamily": "V
erdana", "styleName": "a", "fontSizeBidi": 10, "fontFamilyBidi": "Verdana"}, "text"
: "According to the above chart, the total count of the selling products is
24 and the average
"}, {"characterFormat": {"fontSize": 10, "fontFamily": "Verdana", "styleName": "a",
"fontSizeBidi": 10, "fontFamilyBidi": "Verdana"}, "text": "sales attributed to
this product is $ 5.891 with highest sale $ 15.810 in the month of May in
"}, {"characterFormat": {"fontSize": 10, "fontFamily": "Verdana", "styleName": "a",
"fontSizeBidi": 10, "fontFamilyBidi": "Verdana"}, "text": "2014. In the same
year, in the month of March the same product reached the amount of $
"}, {"characterFormat": {"fontSize": 10, "fontFamily": "Verdana", "styleName": "a",
"fontSizeBidi": 10, "fontFamilyBidi": "Verdana"}, "text": "15.019. These were the
highest sales of the product among the other products for the year
"}, {"characterFormat": {"fontSize": 10, "fontFamily": "Verdana", "styleName": "a",
"fontSizeBidi": 10, "fontFamilyBidi": "Verdana"}, "text": "2014."}]]], "headersFoo
ters": {}], "characterFormat": {"bold": false, "italic": false, "fontSize": 11, "fon
tFamily": "Calibri", "underline": "None", "strikethrough": "None", "baselineAlignm
ent": "Normal", "highlightColor": "NoColor", "fontColor": "#000000", "fontSizeBidi
": 11, "fontFamilyBidi": "Calibri"}, "paragraphFormat": {"leftIndent": 0, "rightInd
ent": 0, "firstLineIndent": 0, "textAlignment": "Left", "beforeSpacing": 0, "afterSp
acing": 8, "lineSpacing": 1.0791666507720947, "lineSpacingType": "Multiple", "list
Format": {}, "bidi": false}, "defaultTabWidth": 36, "enforcement": false, "hashValue
": "", "saltValue": "", "formatting": false, "protectionType": "NoProtection", "styl
es": [{"name": "Normal", "type": "Paragraph", "paragraphFormat": {"listFormat": {}
}, "characterFormat": {}, "next": "Normal"}, {"name": "Heading
1", "type": "Paragraph", "paragraphFormat": {"beforeSpacing": 12, "afterSpacing": 3

```

```

, "lineSpacing": 1, "lineSpacingType": "Multiple", "outlineLevel": "Level1", "listFormat": {}, "characterFormat": { "bold": true, "fontSize": 16, "fontFamily": "Arial", "boldBidi": true, "fontSizeBidi": 16, "fontFamilyBidi": "Arial" }, "basedOn": "Normal", "link": "Heading 1 Char", "next": "Normal" }, { "name": "Heading 1 Char", "type": "Character", "characterFormat": { "bold": true, "fontSize": 16, "fontFamily": "Arial", "boldBidi": true, "fontSizeBidi": 16, "fontFamilyBidi": "Arial" }, "basedOn": "Default Paragraph Font", { "name": "Default Paragraph Font", "type": "Character", "characterFormat": {}, { "name": "Balloon Text", "type": "Paragraph", "paragraphFormat": { "afterSpacing": 0, "lineSpacing": 1, "lineSpacingType": "Multiple", "listFormat": {}, "characterFormat": { "fontSize": 9, "fontFamily": "Segoe UI", "fontSizeBidi": 9, "fontFamilyBidi": "Segoe UI" }, "basedOn": "Normal", "link": "Balloon Text Char", { "name": "Balloon Text Char", "type": "Character", "characterFormat": { "fontSize": 9, "fontFamily": "Segoe UI", "fontSizeBidi": 9, "fontFamilyBidi": "Segoe UI" }, "basedOn": "Default Paragraph Font", { "name": "a", "type": "Character", "characterFormat": {}, "basedOn": "Default Paragraph Font", { "name": "Heading 2", "type": "Paragraph", "paragraphFormat": { "leftIndent": 0, "rightIndent": 0, "firstLineIndent": 0, "textAlignment": "Left", "beforeSpacing": 2, "afterSpacing": 0, "lineSpacing": 1.0791666507720947, "lineSpacingType": "Multiple", "outlineLevel": "Level2", "listFormat": {}, "characterFormat": { "fontSize": 13, "fontFamily": "Calibri Light", "fontColor": "#2F5496" }, "basedOn": "Normal", "link": "Heading 2 Char", "next": "Normal" }, { "name": "Heading 2 Char", "type": "Character", "characterFormat": { "fontSize": 13, "fontFamily": "Calibri Light", "fontColor": "#2F5496" }, "basedOn": "Default Paragraph Font", { "name": "Heading 3", "type": "Paragraph", "paragraphFormat": { "leftIndent": 0, "rightIndent": 0, "firstLineIndent": 0, "textAlignment": "Left", "beforeSpacing": 2, "afterSpacing": 0, "lineSpacing": 1.0791666507720947, "lineSpacingType": "Multiple", "outlineLevel": "Level3", "listFormat": {}, "characterFormat": { "fontSize": 12, "fontFamily": "Calibri Light", "fontColor": "#1F3763" }, "basedOn": "Normal", "link": "Heading 3 Char", "next": "Normal" }, { "name": "Heading 3 Char", "type": "Character", "characterFormat": { "fontSize": 12, "fontFamily": "Calibri Light", "fontColor": "#1F3763" }, "basedOn": "Default Paragraph Font", { "name": "Heading 4", "type": "Paragraph", "paragraphFormat": { "leftIndent": 0, "rightIndent": 0, "firstLineIndent": 0, "textAlignment": "Left", "beforeSpacing": 2, "afterSpacing": 0, "lineSpacing": 1.0791666507720947, "lineSpacingType": "Multiple", "outlineLevel": "Level4", "listFormat": {}, "characterFormat": { "italic": true, "fontFamily": "Calibri Light", "fontColor": "#2F5496" }, "basedOn": "Normal", "link": "Heading 4 Char", "next": "Normal" }, { "name": "Heading 4 Char", "type": "Character", "characterFormat": { "italic": true, "fontFamily": "Calibri Light", "fontColor": "#2F5496" }, "basedOn": "Default Paragraph Font", { "name": "Heading 5", "type": "Paragraph", "paragraphFormat": { "leftIndent": 0, "rightIndent": 0, "firstLineIndent": 0, "textAlignment": "Left", "beforeSpacing": 2, "afterSpacing": 0, "lineSpacing": 1.0791666507720947, "lineSpacingType": "Multiple", "outlineLevel": "Level5", "listFormat": {}, "characterFormat": { "fontFamily": "Calibri Light", "fontColor": "#2F5496" }, "basedOn": "Normal", "link": "Heading 5 Char", "next": "Normal" }, { "name": "Heading 5 Char", "type": "Character", "characterFormat": { "fontFamily": "Calibri Light", "fontColor": "#2F5496" }, "basedOn": "Default Paragraph Font", { "name": "Heading 6", "type": "Paragraph", "paragraphFormat": { "leftIndent": 0, "rightIndent": 0, "firstLineIndent": 0, "textAlignment": "Left", "beforeSpacing": 2, "afterSpacing": 0, "lineSpacing": 1.0791666507720947, "lineSpacingType": "Multiple", "outlineLevel": "Level6", "listFormat": {}, "characterFormat": { "fontFamily": "Calibri

```

```

Light", "fontColor": "#1F3763"}, "basedOn": "Normal", "link": "Heading 6
Char", "next": "Normal"}, {"name": "Heading 6
Char", "type": "Character", "characterFormat": {"fontFamily": "Calibri
Light", "fontColor": "#1F3763"}, "basedOn": "Default Paragraph
Font"}], "lists": [], "abstractLists": []}`;
        //Open the default document in Document Editor.
        (this.documentEditor as DocumentEditorComponent).open(sfddt);
    }
}
}

```

MAIN.TS

```

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

```

Supported Chart Types

The following chart types are supported in document editor

- Scatter_Markers
- Bubble
- Area
- Area_Stacked
- AreaStacked100
- Bar_Clustered
- Bar_Stacked
- BarStacked100
- Column_Clustered
- Column_Stacked
- ColumnStacked100
- Pie
- Doughnut
- Line
- Line_Markers
- LineMarkersStacked
- LineMarkersStacked_100
- Line_Stacked
- LineStacked100

Document management in Angular Document editor component

Document Editor provides support to restrict editing. When the protected document includes range permission, then unique user or user group only authorized to edit separate text area.

Set current user

You can use the `currentUser` property to authorize the current document user by name, email, or user group name.

The following code shows how to set currentUser

```
`typescript
```

```
this.container.documentEditor.currentUser = 'engineer@mycompany.com';
```

```
,
```

Highlighting the text area

You can highlight the editable region of the current user using the `userColor` property.

The following code shows how to set `userColor`.

```
`typescript
```

```
this.container.documentEditor.userColor = '#fff000';
```

```
,
```

You can toggle the highlight the editable region value using the "highlightEditableRanges" property.

The following code shows how to toggle the highlight editable region value.

```
`typescript
```

```
container.documentEditor.documentEditorSettings.highlightEditableRanges = true;
```

```
,
```

Restrict Editing Pane

Restrict Editing Pane provides the following options to manage the document:

- To apply formatting restrictions to the current document, select the allow formatting check box.
- To apply editing restrictions to the current document, select the read only check box.
- To add users to the current document, select more users option and add user from the popup dialog.
- To include range permission to the current document, select parts of the document and choose users who are allowed to freely edit them from the listed check box.
- To apply the chosen editing restrictions, click the **YES,START ENFORCING PROTECTION** button. A dialog box displays asking for a password to protect.
- To stop protection, select **STOP PROTECTION** button. A dialog box displays asking for a password to stop protection.

The following code shows Restrict Editing Pane. To unprotect the document, use password '123'.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DocumentEditorContainerModule } from '@syncfusion/ej2-angular-documenteditor'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import { ToolbarService, DocumentEditorContainerComponent } from '@syncfusion/ej2-angular-documenteditor';
@Component({
  imports: [
    DocumentEditorContainerModule
  ],
  standalone: true,
```

```

        selector: 'app-container',
        // specifies the template string for the DocumentEditorContainer
        component
        template: `<ejs-documenteditorcontainer #document_editor
(created)="onCreated()" height="600px" style="display:block"
[enableToolbar]=true> </ejs-documenteditorcontainer>`,
        providers: [ToolbarService]
    })
    export class AppComponent {
        @ViewChild('document_editor')
        public container?: DocumentEditorContainerComponent;
        onCreated() {
            if ((this.container as DocumentEditorContainerComponent)
).documentEditor.isDocumentLoaded) {
                let sfdt: string =
`{"sections":[{"blocks":[{"characterFormat":{"fontSize":14.0,"fontSizeBidi":
14.0},"paragraphFormat":{"lineSpacing":32.0,"lineSpacingType":"Exactly","sty
leName":"Normal"},"inlines":[{"text":"Name","characterFormat":{"bold":true,"
fontSize":14.0,"boldBidi":true,"fontSizeBidi":14.0}},{text":":","characterF
ormat":{"fontSize":14.0,"fontSizeBidi":14.0}}]},{rows":[{"rowFormat":{"allo
wBreakAcrossPages":true,"isHeader":false,"height":20.0,"heightType":"AtLeast
","borders":{"left":{"lineStyle":"None","lineWidth":0.0,"shadow":false,"spac
e":0.0,"hasNoneStyle":false},"right":{"lineStyle":"None","lineWidth":0.0,"sh
adow":false,"space":0.0,"hasNoneStyle":false},"top":{"lineStyle":"None","lin
eWidth":0.0,"shadow":false,"space":0.0,"hasNoneStyle":false},"bottom":{"line
Style":"None","lineWidth":0.0,"shadow":false,"space":0.0,"hasNoneStyle":fals
e},"vertical":{"lineStyle":"None","lineWidth":0.0,"shadow":false,"space":0.0
,"hasNoneStyle":false},"horizontal":{"lineStyle":"None","lineWidth":0.0,"sha
dow":false,"space":0.0,"hasNoneStyle":false},"diagonalDown":{"lineStyle":"No
ne","lineWidth":0.0,"shadow":false,"space":0.0,"hasNoneStyle":false},"diagon
alUp":{"lineStyle":"None","lineWidth":0.0,"shadow":false,"space":0.0,"hasNon
eStyle":false}}]},{cells":[{"blocks":[{"paragraphFormat":{"styleName":"Normal
"},"inlines":[{"editRangeId":"1348272392","columnFirst":0,"columnLast":0,"us
er":"engineer@mycompany.com"},{text:"Enter
name"},{editRangeId":"1348272392","editableRangeStart":{"editRangeId":"1348
272392","columnFirst":0,"columnLast":0,"user":"engineer@mycompany.com"}}]},{
"cellFormat":{"columnSpan":1,"rowSpan":1,"preferredWidth":467.5,"preferredWi
dthType":"Point","verticalAlignment":"Center","isSamePaddingAsTable":true,"b
orders":{"left":{"lineStyle":"None","lineWidth":0.0,"shadow":false,"space":0
.0,"hasNoneStyle":false},"right":{"lineStyle":"None","lineWidth":0.0,"shadow
":false,"space":0.0,"hasNoneStyle":false},"top":{"lineStyle":"None","lineWid
th":0.0,"shadow":false,"space":0.0,"hasNoneStyle":false},"bottom":{"lineStyl
e":"None","lineWidth":0.0,"shadow":false,"space":0.0,"hasNoneStyle":false},"
vertical":{"lineStyle":"None","lineWidth":0.0,"shadow":false,"space":0.0,"ha
sNoneStyle":false},"horizontal":{"lineStyle":"None","lineWidth":0.0,"shadow"
: false,"space":0.0,"hasNoneStyle":false},"diagonalDown":{"lineStyle":"None"
,"lineWidth":0.0,"shadow":false,"space":0.0,"hasNoneStyle":false},"diagonalUp
":{"lineStyle":"None","lineWidth":0.0,"shadow":false,"space":0.0,"hasNoneStyl
e":false}}]},{title:null,"description":null,"tableFormat":{"allowAutoFit":
true,"leftIndent":0.0,"tableAlignment":"Left","preferredWidthType":"Auto"
,"borders":{"left":{"lineStyle":"Single","lineWidth":0.5,"shadow":false,"spa
ce":0.0,"hasNoneStyle":false},"right":{"lineStyle":"Single","lineWidth":0.5
,"shadow":false,"space":0.0,"hasNoneStyle":false},"top":{"lineStyle":"Single
","lineWidth":0.5,"shadow":false,"space":0.0,"hasNoneStyle":false},"bottom":{
"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0.0,"hasNoneStyl
e":false},"vertical":{"lineStyle":"Single","lineWidth":0.5,"shadow":false,"s
pace":0.0,"hasNoneStyle":false},"horizontal":{"lineStyle":"Single","lineWidt

```

```

h":0.5,"shadow":false,"space":0.0,"hasNoneStyle":false},"diagonalDown":{"lin
eStyle":"None","lineWidth":0.0,"shadow":false,"space":0.0,"hasNoneStyle":fal
se},"diagonalUp":{"lineStyle":"None","lineWidth":0.0,"shadow":false,"space":
0.0,"hasNoneStyle":false}},{"bidi":false}},{"characterFormat":{"bold":true,"f
ontSize":14.0,"boldBidi":true,"fontSizeBidi":14.0},"paragraphFormat":{"lines
pacing":32.0,"lineSpacingType":"Exactly","styleName":"Normal"},"inlines":[{"
text":"Designation:", "characterFormat":{"bold":true,"fontSize":14.0,"boldBidi":true,"fontSizeBidi":14.0}}]}, {"rows":[{"rowFormat":{"allowBreakAcrossPage
s":true,"isHeader":false,"height":20.0,"heightType":"AtLeast","borders":{"lef
t":{"lineStyle":"None","lineWidth":0.0,"shadow":false,"space":0.0,"hasNoneS
tyle":false},"right":{"lineStyle":"None","lineWidth":0.0,"shadow":false,"spa
ce":0.0,"hasNoneStyle":false},"top":{"lineStyle":"None","lineWidth":0.0,"sha
dow":false,"space":0.0,"hasNoneStyle":false},"bottom":{"lineStyle":"None","l
ineWidth":0.0,"shadow":false,"space":0.0,"hasNoneStyle":false},"vertical":{"
lineStyle":"None","lineWidth":0.0,"shadow":false,"space":0.0,"hasNoneStyle":
false},"horizontal":{"lineStyle":"None","lineWidth":0.0,"shadow":false,"spac
e":0.0,"hasNoneStyle":false},"diagonalDown":{"lineStyle":"None","lineWidth":
0.0,"shadow":false,"space":0.0,"hasNoneStyle":false},"diagonalUp":{"lineStyl
e":"None","lineWidth":0.0,"shadow":false,"space":0.0,"hasNoneStyle":false}}
},"cells":[{"blocks":[{"paragraphFormat":{"styleName":"Normal"},"inlines":[{"
editRangeId":"808933422","columnFirst":0,"columnLast":0,"user":"engineer@myc
ompany.com"}],"text":"Enter
designation"}],"editRangeId":"808933422","editableRangeStart":{"editRangeId"
:"808933422","columnFirst":0,"columnLast":0,"user":"engineer@mycompany.com"}
}]}],"cellFormat":{"columnSpan":1,"rowSpan":1,"preferredWidth":467.5,"prefer
redWidthType":"Point","verticalAlignment":"Center","isSamePaddingAsTable":tr
ue,"borders":{"left":{"lineStyle":"None","lineWidth":0.0,"shadow":false,"spa
ce":0.0,"hasNoneStyle":false},"right":{"lineStyle":"None","lineWidth":0.0,"s
hadow":false,"space":0.0,"hasNoneStyle":false},"top":{"lineStyle":"None","li
neWidth":0.0,"shadow":false,"space":0.0,"hasNoneStyle":false},"bottom":{"lin
eStyle":"None","lineWidth":0.0,"shadow":false,"space":0.0,"hasNoneStyle":fal
se},"vertical":{"lineStyle":"None","lineWidth":0.0,"shadow":false,"space":0.
0,"hasNoneStyle":false},"horizontal":{"lineStyle":"None","lineWidth":0.0,"sh
adow":false,"space":0.0,"hasNoneStyle":false},"diagonalDown":{"lineStyle":"N
one","lineWidth":0.0,"shadow":false,"space":0.0,"hasNoneStyle":false},"diago
nalUp":{"lineStyle":"None","lineWidth":0.0,"shadow":false,"space":0.0,"hasNo
neStyle":false}}]}]}],"title":null,"description":null,"tableFormat":{"allowA
utoFit":true,"leftIndent":0.0,"tableAlignment":"Left","preferredWidthType":"
Auto","borders":{"left":{"lineStyle":"Single","lineWidth":0.5,"shadow":false
,"space":0.0,"hasNoneStyle":false},"right":{"lineStyle":"Single","lineWidth"
:0.5,"shadow":false,"space":0.0,"hasNoneStyle":false},"top":{"lineStyle":"Si
ngle","lineWidth":0.5,"shadow":false,"space":0.0,"hasNoneStyle":false},"bott
om":{"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0.0,"hasNon
eStyle":false},"vertical":{"lineStyle":"Single","lineWidth":0.5,"shadow":fal
se,"space":0.0,"hasNoneStyle":false},"horizontal":{"lineStyle":"Single","lin
eWidth":0.5,"shadow":false,"space":0.0,"hasNoneStyle":false},"diagonalDown":
{"lineStyle":"None","lineWidth":0.0,"shadow":false,"space":0.0,"hasNoneStyle
":false},"diagonalUp":{"lineStyle":"None","lineWidth":0.0,"shadow":false,"sp
ace":0.0,"hasNoneStyle":false}},{"bidi":false}},{"characterFormat":{"bold":tr
ue,"fontSize":14.0,"boldBidi":true,"fontSizeBidi":14.0},"paragraphFormat":{"
lineSpacing":32.0,"lineSpacingType":"Exactly","styleName":"Normal"},"inlines
":[{"text":"Email
Address:", "characterFormat":{"bold":true,"fontSize":14.0,"boldBidi":true,"fo
ntSizeBidi":14.0}}, {"name":"_GoBack","bookmarkType":0}, {"name":"_GoBack","bo
okmarkType":1}]}],"rows":[{"rowFormat":{"allowBreakAcrossPages":true,"isHead
er":false,"height":20.0,"heightType":"AtLeast","borders":{"left":{"lineStyle
":"None","lineWidth":0.0,"shadow":false,"space":0.0,"hasNoneStyle":false},"r

```

```

ight":{"lineStyle":"None","lineWidth":0.0,"shadow":false,"space":0.0,"hasNone
eStyle":false},"top":{"lineStyle":"None","lineWidth":0.0,"shadow":false,"spa
ce":0.0,"hasNoneStyle":false},"bottom":{"lineStyle":"None","lineWidth":0.0,"
shadow":false,"space":0.0,"hasNoneStyle":false},"vertical":{"lineStyle":"Non
e","lineWidth":0.0,"shadow":false,"space":0.0,"hasNoneStyle":false},"horizon
tal":{"lineStyle":"None","lineWidth":0.0,"shadow":false,"space":0.0,"hasNone
Style":false},"diagonalDown":{"lineStyle":"None","lineWidth":0.0,"shadow":fa
lse,"space":0.0,"hasNoneStyle":false},"diagonalUp":{"lineStyle":"None","line
Width":0.0,"shadow":false,"space":0.0,"hasNoneStyle":false}}},"cells":[{"blo
cks":[{"paragraphFormat":{"styleName":"Normal"},"inlines":[{"editRangeId":"8
10441411","columnFirst":0,"columnLast":0,"user":"engineer@mycompany.com"}},{
"text":"Enter email
address"}},{editRangeId":"810441411","editableRangeStart":{"editRangeId":"81
0441411","columnFirst":0,"columnLast":0,"user":"engineer@mycompany.com"}}]]
,"cellFormat":{"columnSpan":1,"rowSpan":1,"preferredWidth":467.5,"preferredW
idthType":"Point","verticalAlignment":"Center","isSamePaddingAsTable":true,"
borders":{"left":{"lineStyle":"None","lineWidth":0.0,"shadow":false,"space":
0.0,"hasNoneStyle":false},"right":{"lineStyle":"None","lineWidth":0.0,"shado
w":false,"space":0.0,"hasNoneStyle":false},"top":{"lineStyle":"None","lineWi
dth":0.0,"shadow":false,"space":0.0,"hasNoneStyle":false},"bottom":{"lineSty
le":"None","lineWidth":0.0,"shadow":false,"space":0.0,"hasNoneStyle":false},
"vertical":{"lineStyle":"None","lineWidth":0.0,"shadow":false,"space":0.0,"h
asNoneStyle":false},"horizontal":{"lineStyle":"None","lineWidth":0.0,"shadow
":false,"space":0.0,"hasNoneStyle":false},"diagonalDown":{"lineStyle":"None"
,"lineWidth":0.0,"shadow":false,"space":0.0,"hasNoneStyle":false},"diagonalU
p":{"lineStyle":"None","lineWidth":0.0,"shadow":false,"space":0.0,"hasNoneSt
yle":false}}}}]]},"title":null,"description":null,"tableFormat":{"allowAutoF
it":true,"leftIndent":0.0,"tableAlignment":"Left","preferredWidthType":"Auto
","borders":{"left":{"lineStyle":"Single","lineWidth":0.5,"shadow":false,"sp
ace":0.0,"hasNoneStyle":false},"right":{"lineStyle":"Single","lineWidth":0.5
,"shadow":false,"space":0.0,"hasNoneStyle":false},"top":{"lineStyle":"Single
","lineWidth":0.5,"shadow":false,"space":0.0,"hasNoneStyle":false},"bottom":
{"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0.0,"hasNoneSty
le":false},"vertical":{"lineStyle":"Single","lineWidth":0.5,"shadow":false,"
space":0.0,"hasNoneStyle":false},"horizontal":{"lineStyle":"Single","lineWid
th":0.5,"shadow":false,"space":0.0,"hasNoneStyle":false},"diagonalDown":{"li
neStyle":"None","lineWidth":0.0,"shadow":false,"space":0.0,"hasNoneStyle":fa
lse},"diagonalUp":{"lineStyle":"None","lineWidth":0.0,"shadow":false,"space"
:0.0,"hasNoneStyle":false}}},"bidi":false},"characterFormat":{"bold":true,"
fontSize":14.0,"boldBidi":true,"fontSizeBidi":14.0},"paragraphFormat":{"line
Spacing":32.0,"lineSpacingType":"Exactly","styleName":"Normal"},"inlines":[{"
"text":"Feedbacks:", "characterFormat":{"bold":true,"fontSize":14.0,"boldBidi
":true,"fontSizeBidi":14.0}}]}}, {"rows":[{"rowFormat":{"allowBreakAcrossPages
":true,"isHeader":false,"height":20.0,"heightType":"AtLeast","borders":{"lef
t":{"lineStyle":"None","lineWidth":0.0,"shadow":false,"space":0.0,"hasNoneSt
yle":false},"right":{"lineStyle":"None","lineWidth":0.0,"shadow":false,"spac
e":0.0,"hasNoneStyle":false},"top":{"lineStyle":"None","lineWidth":0.0,"shad
ow":false,"space":0.0,"hasNoneStyle":false},"bottom":{"lineStyle":"None","li
neWidth":0.0,"shadow":false,"space":0.0,"hasNoneStyle":false},"vertical":{"l
ineStyle":"None","lineWidth":0.0,"shadow":false,"space":0.0,"hasNoneStyle":f
alse},"horizontal":{"lineStyle":"None","lineWidth":0.0,"shadow":false,"space
":0.0,"hasNoneStyle":false},"diagonalDown":{"lineStyle":"None","lineWidth":0
.0,"shadow":false,"space":0.0,"hasNoneStyle":false},"diagonalUp":{"lineStyle
":"None","lineWidth":0.0,"shadow":false,"space":0.0,"hasNoneStyle":false}}},
"cells":[{"blocks":[{"paragraphFormat":{"styleName":"Normal"},"inlines":[{"e
ditRangeId":"1016946268","columnFirst":0,"columnLast":0,"user":"manager@myco
mpany.com"}},{ "text":"Enter the

```



```

feedbacks"}}, {"editRangeId": "1016946268", "editableRangeStart": {"editRangeId":
"1016946268", "columnFirst": 0, "columnLast": 0, "user": "manager@mycompany.com"}}
]]], "cellFormat": {"columnSpan": 1, "rowSpan": 1, "preferredWidth": 467.5, "preferre
dWidthType": "Point", "verticalAlignment": "Center", "isSamePaddingAsTable": tru
e, "borders": {"left": {"lineStyle": "None", "lineWidth": 0.0, "shadow": false, "spac
e": 0.0, "hasNoneStyle": false}, "right": {"lineStyle": "None", "lineWidth": 0.0, "sh
adow": false, "space": 0.0, "hasNoneStyle": false}, "top": {"lineStyle": "None", "lin
eWidth": 0.0, "shadow": false, "space": 0.0, "hasNoneStyle": false}, "bottom": {"line
Style": "None", "lineWidth": 0.0, "shadow": false, "space": 0.0, "hasNoneStyle": fals
e}, "vertical": {"lineStyle": "None", "lineWidth": 0.0, "shadow": false, "space": 0.0
, "hasNoneStyle": false}, "horizontal": {"lineStyle": "None", "lineWidth": 0.0, "sha
dow": false, "space": 0.0, "hasNoneStyle": false}, "diagonalDown": {"lineStyle": "No
ne", "lineWidth": 0.0, "shadow": false, "space": 0.0, "hasNoneStyle": false}, "diagon
alUp": {"lineStyle": "None", "lineWidth": 0.0, "shadow": false, "space": 0.0, "hasNon
eStyle": false}}}], "title": null, "description": null, "tableFormat": {"allowAu
toFit": true, "leftIndent": 0.0, "tableAlignment": "Left", "preferredWidthType": "A
uto", "borders": {"left": {"lineStyle": "Single", "lineWidth": 0.5, "shadow": false,
"space": 0.0, "hasNoneStyle": false}, "right": {"lineStyle": "Single", "lineWidth":
0.5, "shadow": false, "space": 0.0, "hasNoneStyle": false}, "top": {"lineStyle": "Sin
gle", "lineWidth": 0.5, "shadow": false, "space": 0.0, "hasNoneStyle": false}, "botto
m": {"lineStyle": "Single", "lineWidth": 0.5, "shadow": false, "space": 0.0, "hasNone
Style": false}, "vertical": {"lineStyle": "Single", "lineWidth": 0.5, "shadow": fals
e, "space": 0.0, "hasNoneStyle": false}, "horizontal": {"lineStyle": "Single", "line
Width": 0.5, "shadow": false, "space": 0.0, "hasNoneStyle": false}, "diagonalDown": {
"lineStyle": "None", "lineWidth": 0.0, "shadow": false, "space": 0.0, "hasNoneStyle"
: false}, "diagonalUp": {"lineStyle": "None", "lineWidth": 0.0, "shadow": false, "spa
ce": 0.0, "hasNoneStyle": false}}, {"characterFormat": {"bold": tru
e, "fontSize": 14.0, "boldBidi": true, "fontSizeBidi": 14.0}, "paragraphFormat": {"l
ineSpacing": 32.0, "lineSpacingType": "Exactly", "styleName": "Normal"}, "inlines"
: [{"text": "Review
comments", "characterFormat": {"bold": true, "fontSize": 14.0, "boldBidi": true, "f
ontSizeBidi": 14.0}}], {"rows": [{"rowFormat": {"allowBreakAcrossPages": true, "i
sHeader": false, "height": 20.0, "heightType": "AtLeast", "borders": {"left": {"line
Style": "None", "lineWidth": 0.0, "shadow": false, "space": 0.0, "hasNoneStyle": fals
e}, "right": {"lineStyle": "None", "lineWidth": 0.0, "shadow": false, "space": 0.0, "h
asNoneStyle": false}, "top": {"lineStyle": "None", "lineWidth": 0.0, "shadow": false
, "space": 0.0, "hasNoneStyle": false}, "bottom": {"lineStyle": "None", "lineWidth":
0.0, "shadow": false, "space": 0.0, "hasNoneStyle": false}, "vertical": {"lineStyle"
: "None", "lineWidth": 0.0, "shadow": false, "space": 0.0, "hasNoneStyle": false}, "ho
rizontal": {"lineStyle": "None", "lineWidth": 0.0, "shadow": false, "space": 0.0, "ha
sNoneStyle": false}, "diagonalDown": {"lineStyle": "None", "lineWidth": 0.0, "shado
w": false, "space": 0.0, "hasNoneStyle": false}, "diagonalUp": {"lineStyle": "None",
"lineWidth": 0.0, "shadow": false, "space": 0.0, "hasNoneStyle": false}}}], "cells": [
{"blocks": [{"paragraphFormat": {"styleName": "Normal"}, "inlines": [{"editRangeId
": "1373703080", "columnFirst": 0, "columnLast": 0, "user": "manager@mycompany.com
"}, {"text": "Enter the
comments", "editRangeId": "1373703080", "editableRangeStart": {"editRangeId": "
1373703080", "columnFirst": 0, "columnLast": 0, "user": "manager@mycompany.com"}}]
}], "cellFormat": {"columnSpan": 1, "rowSpan": 1, "preferredWidth": 467.5, "preferre
dWidthType": "Point", "verticalAlignment": "Center", "isSamePaddingAsTable": true
, "borders": {"left": {"lineStyle": "None", "lineWidth": 0.0, "shadow": false, "space
": 0.0, "hasNoneStyle": false}, "right": {"lineStyle": "None", "lineWidth": 0.0, "sha
dow": false, "space": 0.0, "hasNoneStyle": false}, "top": {"lineStyle": "None", "line
Width": 0.0, "shadow": false, "space": 0.0, "hasNoneStyle": false}, "bottom": {"lines
tyle": "None", "lineWidth": 0.0, "shadow": false, "space": 0.0, "hasNoneStyle": false
}, "vertical": {"lineStyle": "None", "lineWidth": 0.0, "shadow": false, "space": 0.0,
"hasNoneStyle": false}, "horizontal": {"lineStyle": "None", "lineWidth": 0.0, "shad

```



```

ow":false,"space":0.0,"hasNoneStyle":false},"diagonalDown":{"lineStyle":"None","lineWidth":0.0,"shadow":false,"space":0.0,"hasNoneStyle":false},"diagonalUp":{"lineStyle":"None","lineWidth":0.0,"shadow":false,"space":0.0,"hasNoneStyle":false}}}}]],"title":null,"description":null,"tableFormat":{"allowAutoFit":true,"leftIndent":0.0,"tableAlignment":"Left","preferredWidthType":"Auto","borders":{"left":{"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0.0,"hasNoneStyle":false},"right":{"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0.0,"hasNoneStyle":false},"top":{"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0.0,"hasNoneStyle":false},"bottom":{"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0.0,"hasNoneStyle":false},"vertical":{"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0.0,"hasNoneStyle":false},"horizontal":{"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0.0,"hasNoneStyle":false},"diagonalDown":{"lineStyle":"None","lineWidth":0.0,"shadow":false,"space":0.0,"hasNoneStyle":false},"diagonalUp":{"lineStyle":"None","lineWidth":0.0,"shadow":false,"space":0.0,"hasNoneStyle":false}}},"bidi":false},"paragraphFormat":{"styleName":"Normal"},"inlines":[]},"headersFooters":{"header":{"blocks":[{"paragraphFormat":{"styleName":"Header"},"inlines":[{"text":"Employee's Details"}]}]},"sectionFormat":{"headerDistance":36.0,"footerDistance":36.0,"pageWidth":612.0,"pageHeight":792.0,"leftMargin":72.0,"rightMargin":72.0,"topMargin":72.0,"bottomMargin":72.0,"differentFirstPage":false,"differentOddAndEvenPages":false,"bidi":false},"characterFormat":{"fontSize":11.0,"fontFamily":"Calibri","fontSizeBidi":11.0,"fontFamilyBidi":"Calibri"},"paragraphFormat":{"afterSpacing":8.0,"lineSpacing":1.0791666507720947,"lineSpacingType":"Multiple"},"background":{"color":"#FFFFFF"},"styles":[{"type":"Paragraph","name":"Normal","next":"Normal"}, {"type":"Character","name":"Default Paragraph Font"}, {"type":"Paragraph","name":"List Paragraph","basedOn":"Normal","paragraphFormat":{"leftIndent":36.0,"contextualSpacing":true}}, {"type":"Paragraph","name":"Header","basedOn":"Normal","next":"Normal","link":"Header Char","paragraphFormat":{"afterSpacing":0.0,"lineSpacing":1.0,"lineSpacingType":"Multiple","tabs":[{"tabJustification":"Center","position":234.0,"tabLeader":"None","deletePosition":0.0}, {"tabJustification":"Right","position":468.0,"tabLeader":"None","deletePosition":0.0}]}}, {"type":"Character","name":"Header Char","basedOn":"Default Paragraph Font"}, {"type":"Paragraph","name":"Footer","basedOn":"Normal","link":"Footer Char","paragraphFormat":{"afterSpacing":0.0,"lineSpacing":1.0,"lineSpacingType":"Multiple","tabs":[{"tabJustification":"Center","position":234.0,"tabLeader":"None","deletePosition":0.0}, {"tabJustification":"Right","position":468.0,"tabLeader":"None","deletePosition":0.0}]}}, {"type":"Character","name":"Footer Char","basedOn":"Default Paragraph Font"}],"defaultTabWidth":36.0,"formatting":false,"protectionType":"ReadOnly","enforcement":true,"hashValue":"TQGGuJuLceQCe234Ygx4q6NFgHpRMfilhjFTTojyKzbQOkwk+ckEM9CjNIdkiUhSR/e/7sfMxO4sbPcg/DBzztMg==","saltValue":"FXbkr1RtDIIIZfw1M71dMg=="}`;

    (this.container as DocumentEditorContainerComponent).serviceUrl
= 'https://ej2services.syncfusion.com/production/web-services/api/documenteditor/';
    //Open the default document in Document Editor
    (this.container as
DocumentEditorContainerComponent).documentEditor.open(sfdd);
    }
}
}

```

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 protect the document in form filling mode](#)
- [How to protect the document in comments only mode](#)
- [How to protect the document in track changes only mode](#)

Spell check in Angular Document editor component

Document Editor supports performing spell checking for any input text. You can perform spell checking for the text in Document Editor and it will provide suggestions for the mis-spelled words through dialog and in context menu. Document editor's spell checker is compatible with [hunspell dictionary files](#).

`typescript

```
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import { ToolbarService, DocumentEditorContainerComponent } from '@syncfusion/ej2-angular-documenteditor';

@Component({
  selector: 'app-container',
  // specifies the template string for the DocumentEditorContainer component
  template: <ejs-documenteditorcontainer #document_editor (created)="onCreated()"
[enableToolbar]=true [enableSpellCheck]=true> </ejs-documenteditorcontainer>,
  providers: [ToolbarService]
})
export class AppComponent {
  @ViewChild('document_editor')
  public container: DocumentEditorContainerComponent;

  onCreated() {
    //Specifies the language id to map server side dictionary.
    this.container.documentEditor.spellChecker.languageID = 1033;
    this.container.documentEditor.spellChecker.removeUnderline = false;
    this.container.documentEditor.spellChecker.allowSpellCheckAndSuggestion = true;
  }
}
```

Features

- Supports context menu suggestions.
- Provides built-in options to Ignore, Ignore All, Change, Change All for error words in spell checker dialog.

Enable SpellCheck

To enable spell check in DocumentEditor, set [enableSpellCheck](#) property as `true` and then configure SpellCheckSettings.

Disable SpellCheck

To disable spell check in DocumentEditor, set [enableSpellCheck](#) property as `false` or remove [enableSpellCheck](#) property initialization code. The default value of this property is false.

Spell check settings

Remove Underline

By default, mis-spelled words are marked with squiggly line. You can also disable this behavior by enabling the [removeUnderline](#) API and now, the squiggly lines will never be rendered for mis-spelled words.

```
`typescript
```

```
this.container.documentEditor.spellChecker.removeUnderline = false;
```

```
,
```

AllowSpellCheckAndSuggestion

By default, on performing spell check in Document Editor, both spelling and suggestions of the mis-spelled words will be retrieved, and this mis-spelled words can be corrected through context menu suggestions. You can modify this behavior using the [allowSpellCheckAndSuggestion](#) API, which will perform only spell check.

```
`typescript
```

```
this.container.documentEditor.spellChecker.allowSpellCheckAndSuggestion = false;
```

```
,
```

LanguageID

Document Editor provides multi-language spell check support. You can add as many languages (dictionaries) in the server-side and to use that language for spell checking in Document Editor, it must be matched with [languageID](#) you pass in the Document Editor.

```
`typescript
```

```
this.container.documentEditor.spellChecker.languageID = 1033; //LCID of "en-us";
```

```
,
```

- Refer to the [Spell checker](#) link for configuring spell checker in server-side.

EnableOptimizedSpellCheck

Document Editor provides option to spellcheck page by page when loading the documents. The default value of this property is false, so when opening the document spellcheck web API will be called for each

word in the document. To optimize the frequency of spellcheck web API calls, you can enable this property.

The following code example illustrates how to enable optimized spell checking.

```
`typescript
this.container.documentEditor.spellChecker.enableOptimizedSpellCheck = true;
`
```

Spell check dictionary cache

Starting from **v20.1.0.xx**, we have optimized the performance and memory usage of spell checker by adding a static method to initialize the dictionaries with specified cache count.

By default, the spell checker holds only one language dictionary in memory. If you want to hold multiple dictionaries in memory, you need to set the cache limit by using **InitializeDictionaries** method as in the below example.

```
`csharp
List<DictionaryData> spellDictCollection = new List<DictionaryData>();
string personalDictPath = string.Empty;
int cacheCount = 2;
// Initialize dictionaries
SpellChecker.InitializeDictionaries(spellDictCollection, personalDictPath, cacheCount);
`
```

If dictionaries are initialized using **InitializeDictionaries** method, then we should use default constructor of the **SpellChecker** to check spelling and get suggestion as in the below example code, it will prevent reinitialization of already loaded dictionaries.

```
`csharp
public string SpellCheck([FromBody] SpellCheckJsonData spellChecker)
{
    try
    {
        SpellChecker spellCheck = new SpellChecker();
        spellCheck.GetSuggestions(spellChecker.LanguageID, spellChecker.TexttoCheck,
            spellChecker.CheckSpelling, spellChecker.CheckSuggestion, spellChecker.AddWord);
        return Newtonsoft.Json.JsonConvert.SerializeObject(spellCheck);
    }
    catch
    {
        return "{\"SpellCollection\":[],\"HasSpellingError\":false,\"Suggestions\":null}";
    }
}
```

```
}  
}  
`
```

Previously on every `SpellChecker.GetSuggestion()` method call, the `.aff` and dictionary data will be parsed to generate suggestion for miss spelled word. But, starting from `v20.1.0.xx`, the `.aff` and dictionary data will be parsed only for the first time alone while calling `SpellChecker.GetSuggestion()` method.

Add new root word and possible words to dictionary

If you find any root word is missing in the dictionary file, then you can add that new root word and the rule to form the possible words to dictionary file using `AddNewWord` API in the server-side Spell check library.

Note:

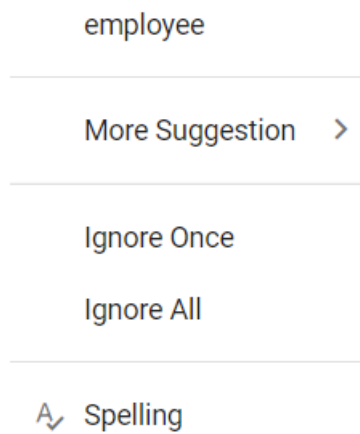
1. The rules are framed automatically using the root word, the possible words and affix file.
 2. If you pass null for the parameters `affPath` and `possibleWords`, then it will add a single root word to dictionary.
 3. This API is included starting from `v20.2.0.xx`.
-

The following code example demonstrates how to add a new root word to the dictionary along with the rule to form the possible words.

```
`csharp  
SpellChecker spellChecker = new SpellChecker();  
  
// Adds the specified new root word to the dictionary along with the rule to form the possible words.  
spellChecker.AddNewWord("en.dic", "en.aff", "construct", new string[] { "constructs", "reconstruct",  
"constructed", "constructive" });  
`
```

Context menu

Right click on error word to open the context menu with spell check options. Please see below screenshot for your reference.



Suggestions

Context menu shows the suggestions for mis-spelled words. By clicking on the required word from suggestion, the error word gets replaced automatically.

Add To Dictionary

Using this option, you can add the current word to the dictionary. So that the spell checker does not consider that word as error in future.

Ignore Once and Ignore All

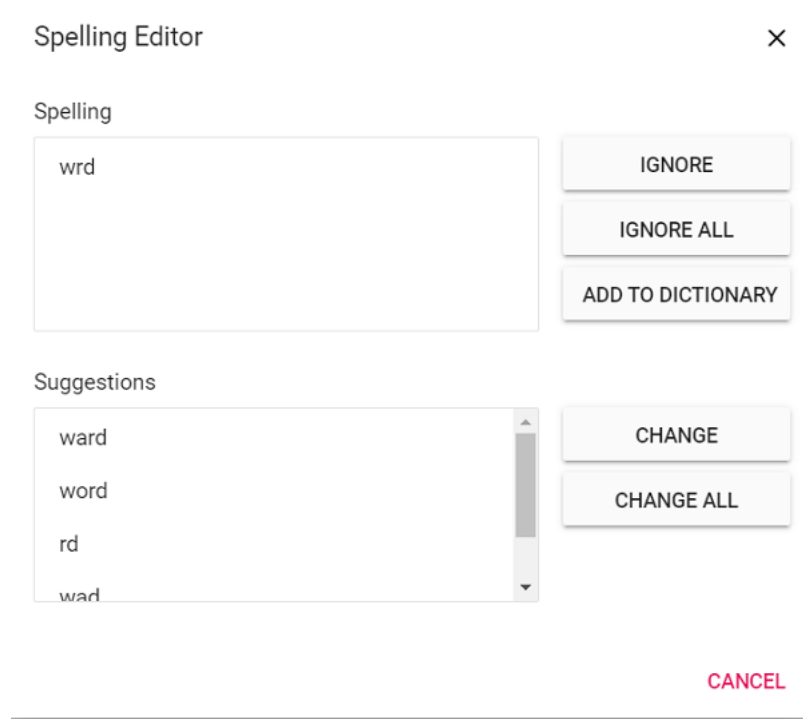
If you do not wish to add the word to dictionary and do not want to show error, use Ignore Once or Ignore All options.

Ignore: ignore only the current occurrence of a word from error.

Ignore All: ignore all occurrence of a word from error in the entire document.

Spelling

Using this option, you can open spell check dialog. Please see below screenshot for your reference.



Global local in Angular Document editor component

Localization

The Localization library allows you to localize default text content of the DocumentEditor. The document editor component has static text on some features (like find & replace, context-menu, dialogs) that can be changed to other cultures (Arabic, Deutsch, French, etc.) by defining the locale value and translation object. Please refer the sample link [RTL](#)

Note: Please refer the [Locale](#).

Document Editor

The following list of properties and its values are used in the document editor.

Locale keywords |Text

New | New

Open | Open

Undo | Undo

Redo | Redo

Image | Image

Table | Table

Link | Link

Bookmark | Bookmark

Table of Contents | Table of Contents

HEADING - - - - 1 | HEADING - - - - 1

HEADING ---- 2 | HEADING ---- 2

HEADING ---- 3 | HEADING ---- 3

Header | Header

Footer | Footer

Page Setup | Page Setup

Page Number | Page Number

Break | Break

Find | Find

Local Clipboard | Local Clipboard

Restrict Editing | Restrict Editing

Upload from computer | Upload from computer

By URL | By URL

Page Break | Page Break

Section Break | Section Break

Header And Footer | Header & Footer

Options | Options

Levels | Levels

Different First Page | Different First Page

Different header and footer for odd and even pages | Different header and footer for odd and even pages.

Different Odd And Even Pages | Different Odd & Even Pages

Different header and footer for first page | Different header and footer for first page.

Position | Position

Header from Top | Header from Top

Footer from Bottom | Footer from Bottom

Distance from top of the page to top of the header | Distance from top of the page to top of the header.

Distance from bottom of the page to bottom of the footer | Distance from bottom of the page to bottom of the footer.

Aspect ratio | Aspect ratio

W | W

H | H

Width | Width

Height | Height

Text | Text

Paragraph | Paragraph

Fill | Fill

Fill color | Fill color

Border Style | Border Style

Outside borders | Outside borders

All borders | All borders

Inside borders | Inside borders

Left border | Left border

Inside vertical border | Inside vertical border

Right border | Right border

Top border | Top border

Inside horizontal border | Inside horizontal border

Bottom border | Bottom border

Border color | Border color

Border width | Border width

Cell | Cell

Merge cells | Merge cells

Insert Or Delete | Insert / Delete

Insert columns to the left | Insert columns to the left

Insert columns to the right | Insert columns to the right

Insert rows above | Insert rows above

Insert rows below | Insert rows below

Delete rows | Delete rows

Delete columns | Delete columns

Cell Margin | Cell Margin

Top | Top

Bottom | Bottom

Left | Left

Right | Right

Align Text | Align Text

Align top | Align top

Align bottom | Align bottom

Align center | Align center

Number of heading or outline levels to be shown in table of contents | Number of heading or outline levels to be shown in table of contents.

Show page numbers | Show page numbers

Show page numbers in table of contents | Show page numbers in table of contents.

Right align page numbers | Right align page numbers

Right align page numbers in table of contents | Right align page numbers in table of contents.

Use hyperlinks | Use hyperlinks

Use hyperlinks instead of page numbers | Use hyperlinks instead of page numbers.

Font | Font

Font Size | Font Size

Font color | Font color

Text highlight color | Text highlight color

Clear all formatting | Clear all formatting

Bold Tooltip | Bold (Ctrl+B)

Italic Tooltip | Italic (Ctrl+I)

Underline Tooltip | Underline (Ctrl+U)

Strikethrough | Strikethrough

Superscript Tooltip | Superscript (Ctrl+Shift++)

Subscript Tooltip | Subscript (Ctrl+=)

Align left Tooltip | Align left (Ctrl+L)

Center Tooltip | Center (Ctrl+E)

Align right Tooltip | Align right (Ctrl+R)

Justify Tooltip | Justify (Ctrl+J)

Decrease indent | Decrease indent

Increase indent | Increase indent

Line spacing | Line spacing

Bullets | Bullets

Numbering | Numbering

Styles | Styles

Manage Styles | Manage Styles

Page | Page

of | of

Fit one page | Fit one page

Spell Check | Spell Check

Underline errors | Underline errors

Fit page width | Fit page width

Update | Update

Cancel | Cancel

Insert | Insert

No Border | No Border

Create a new document | Create a new document.

Open a document | Open a document.

Undo Tooltip | Undo the last operation (Ctrl+Z).

Redo Tooltip | Redo the last operation (Ctrl+Y).

Insert inline picture from a file | Insert inline picture from a file.

Insert a table into the document | Insert a table into the document

Create Hyperlink | Create a link in your document for quick access to web pages and files (Ctrl+K).

Insert a bookmark in a specific place in this document | Insert a bookmark in a specific place in this document.

Provide an overview of your document by adding a table of contents | Provide an overview of your document by adding a table of contents.

Add or edit the header | Add or edit the header.

Add or edit the footer | Add or edit the footer.

Open the page setup dialog | Open the page setup dialog.

Add page numbers | Add page numbers.

Find Text | Find text in the document (Ctrl+F).

Toggle between the internal clipboard and system clipboard | Toggle between the internal clipboard and system clipboard.

Access to system clipboard through script is denied due to browsers security policy. Instead, 1. You can enable internal clipboard to cut, copy and paste within the component. 2. You can use the keyboard shortcuts (Ctrl+X, Ctrl+C and Ctrl+V) to cut, copy and paste with system clipboard.

Current Page Number | The current page number in the document. Click or tap to navigate specific page.

Read only | Read only

Protections | Protections

Error in establishing connection with web server | Error in establishing connection with web server

Single | Single

Double | Double

New comment | New comment

Comments | Comments

Print layout | Print layout

Web layout | Web layout

Text Form | Text Form

Check Box | Check Box

DropDown | Drop-Down

Update Fields | Update Fields

Update cross reference fields | Update cross reference fields

Hide properties pane | Hide properties pane

Show properties pane | Show properties pane

[Color Picker](#)

The following list of properties and its values are used in the color picker.

Locale keywords | Text

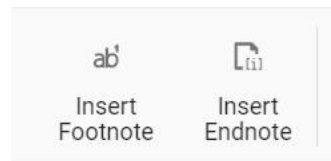
Apply | Apply

Cancel | Cancel

ModeSwitcher | Switch Mode

[Notes in Angular Document editor component](#)

DocumentEditorContainer component provides support for inserting footnotes and endnotes through the in-built toolbar. Refer to the following screenshot.



The Footnotes and endnotes are both ways of adding extra bits of information to your writing outside of the main text. You can use footnotes and endnotes to add side comments to your work or to place other publications like books, articles, or websites.

[Insert footnotes](#)

Document Editor exposes an API to insert footnotes at cursor position programmatically or can be inserted to the end of selected text.

```
`typescript
```

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

```
import {
```

```
DocumentEditorComponent, PrintService, SfdtExportService, WordExportService, TextExportService,  
SelectionService,
```

```

SearchService, EditorService, ImageResizerService, EditorHistoryService, ContextMenuService,
OptionsPaneService, HyperlinkDialogService, TableDialogService, BookmarkDialogService,
TableOfContentsDialogService,
PageSetupDialogService, StyleDialogService, ListDialogService, ParagraphDialogService,
BulletsAndNumberingDialogService,
FontDialogService, TablePropertiesDialogService, BordersAndShadingDialogService,
TableOptionsDialogService,
CellOptionsDialogService, StylesDialogService
} from '@syncfusion/ej2-angular-documenteditor';

@Component({
  selector: 'app-container',
  template: `<div><button ej2-button (click)="insertFootnote()" >Insert Footnote</button><ejs-
documenteditor id="container" serviceUrl="https://ej2services.syncfusion.com/production/web-
services/api/documenteditor/" style="display:block;height:400px" [isReadOnly]=false
[enableSelection]=true
[enablePrint]=true [enableSfdtExport]=true [enableWordExport]=true [enableOptionsPane]=true
[enableContextMenu]=true
[enableHyperlinkDialog]=true [enableBookmarkDialog]=true [enableTableOfContentsDialog]=true
[enableSearch]=true
[enableParagraphDialog]=true [enableListDialog]=true [enableTablePropertiesDialog]=true
[enableBordersAndShadingDialog]=true
[enablePageSetupDialog]=true [enableStyleDialog]=true [enableFontDialog]=true
[enableTableOptionsDialog]=true
[enableTableDialog]=true [enableImageResizer]=true [enableEditor]=true [enableEditorHistory]=true>
</ejs-documenteditor>`,
  encapsulation: ViewEncapsulation.None,
  providers: [PrintService, SfdtExportService, WordExportService, TextExportService, SelectionService,
SearchService, EditorService,
ImageResizerService, EditorHistoryService, ContextMenuService, OptionsPaneService,
HyperlinkDialogService, TableDialogService,
BookmarkDialogService, TableOfContentsDialogService, PageSetupDialogService, StyleDialogService,
ListDialogService,
ParagraphDialogService, BulletsAndNumberingDialogService, FontDialogService,
TablePropertiesDialogService,
BordersAndShadingDialogService, TableOptionsDialogService, CellOptionsDialogService,
StylesDialogService]
})
export class AppComponent {

```

```
@ViewChild('document_editor')
public documentEditor: DocumentEditorComponent;
public insertFootnote(): void {
  //Insert foot note.
  this.documentEditor.editor.insertFootnote();
}
}
```

Insert endnotes

Document Editor exposes an API to insert endnotes at cursor position programmatically or can be inserted to the end of selected text.

```
`typescript
import { Component, ViewEncapsulation } from '@angular/core';
import {
  DocumentEditorComponent, PrintService, SfdtExportService, WordExportService, TextExportService,
  SelectionService,
  SearchService, EditorService, ImageResizerService, EditorHistoryService, ContextMenuService,
  OptionsPaneService, HyperlinkDialogService, TableDialogService, BookmarkDialogService,
  TableOfContentsDialogService,
  PageSetupDialogService, StyleDialogService, ListDialogService, ParagraphDialogService,
  BulletsAndNumberingDialogService,
  FontDialogService, TablePropertiesDialogService, BordersAndShadingDialogService,
  TableOptionsDialogService,
  CellOptionsDialogService, StylesDialogService
} from '@syncfusion/ej2-angular-documenteditor';
@Component({
  selector: 'app-container',
  //specifies the template string for the Document Editor component
  template: `<div><button ej2-button (click)="insertEndnote()" >Insert Footnote</button><ejs-
documenteditor id="container" serviceUrl="https://ej2services.syncfusion.com/production/web-
services/api/documenteditor/" style="display:block;height:400px" [isReadOnly]=false
[enableSelection]=true
[enablePrint]=true [enableSfdtExport]=true [enableWordExport]=true [enableOptionsPane]=true
[enableContextMenu]=true
[enableHyperlinkDialog]=true [enableBookmarkDialog]=true [enableTableOfContentsDialog]=true
[enableSearch]=true`
```

```

[enableParagraphDialog]=true [enableListDialog]=true [enableTablePropertiesDialog]=true
[enableBordersAndShadingDialog]=true

[enablePageSetupDialog]=true [enableStyleDialog]=true [enableFontDialog]=true
[enableTableOptionsDialog]=true

[enableTableDialog]=true [enableImageResizer]=true [enableEditor]=true [enableEditorHistory]=true>
</ejs-documenteditor>`,

encapsulation: ViewEncapsulation.None,

providers: [PrintService, SfdtExportService, WordExportService, TextExportService, SelectionService,
SearchService, EditorService,

ImageResizerService, EditorHistoryService, ContextMenuService, OptionsPaneService,
HyperlinkDialogService, TableDialogService,

BookmarkDialogService, TableOfContentsDialogService, PageSetupDialogService, StyleDialogService,
ListDialogService,

ParagraphDialogService, BulletsAndNumberingDialogService, FontDialogService,
TablePropertiesDialogService,

BordersAndShadingDialogService, TableOptionsDialogService, CellOptionsDialogService,
StylesDialogService]
})

export class AppComponent {
  @ViewChild('document_editor')
  public documentEditor: DocumentEditorComponent;

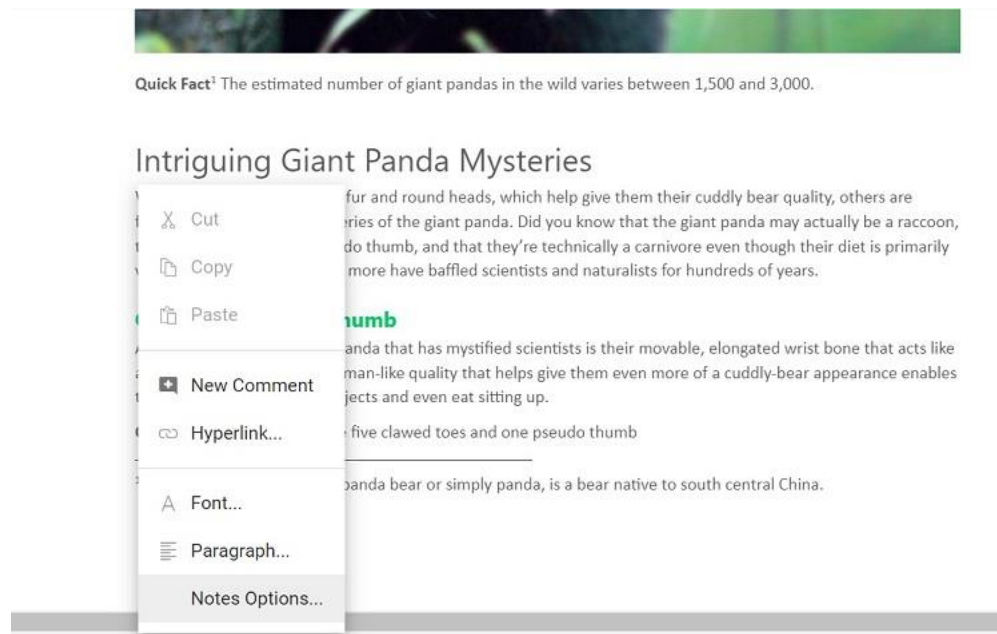
  public insertEndnote(): void {
    //Insert end note.
    this.documentEditor.editor.insertEndnote();
  }
}
`

```

Update or edit footnotes and endnotes

You can update or edit the footnotes and endnotes using the built-in context menu shown up by right-clicking it.

the footnote endnote dialog box popup and you can customize the number format and start at. Refer to the following screenshot.



Collaborative Editing (preview)

Allows multiple users to work on the same document simultaneously. This can be done in real-time, so that collaborators can see the changes as they are made. Collaborative editing can be a great way to improve efficiency, as it allows team members to work together on a document without having to wait for others to finish their changes.

Note: Collaborative editing support is currently in preview mode only and is not yet ready for production environments.

Prerequisites

The following are needed to enable collaborative editing in Document Editor.

- SignalR
- Microsoft SQL Server

How to enable collaborative editing in client side

Step 1: Enable collaborative editing in Document Editor

To enable collaborative editing, inject **CollaborativeEditingHandler** and set the property **enableCollaborativeEditing** to true in the Document Editor, like in the code snippet below.

```
`html
<!-- ./app.component.html -->
<div>
<div id="documenteditor_titlebar" class="e-de-ctn-title" style="height:35px;"></div>
<ejs-documenteditorcontainer #documenteditor_default [enableToolbar]="true"
(created)="onCreated()"
(contentChange)="onContentChange($event)" height="600px" style="display: block;"></ejs-
documenteditorcontainer>
```



```
</div>
`typescript
import { Component, ViewChild } from '@angular/core';
import { DocumentEditorContainerModule, ToolbarService, DocumentEditorContainerComponent,
ContainerContentChangeEventArgs, Operation } from '@syncfusion/ej2-angular-documenteditor';
import { CommonModule } from '@angular/common';
import { RouterOutlet } from '@angular/router';
import { DocumentEditor, CollaborativeEditingHandler } from '@syncfusion/ej2-documenteditor';
import { TitleBar } from "../title-bar"
import { HubConnectionBuilder, HttpTransportType, HubConnectionState, HubConnection } from
'@microsoft/signalr';
import { hideSpinner, showSpinner } from '@syncfusion/ej2-popups';
//Inject collaborative editing module.
DocumentEditorComponent.Inject(CollaborativeEditingHandler);
@Component({
selector: 'app-root',
standalone: true,
imports: [DocumentEditorContainerModule, CommonModule, RouterOutlet],
templateUrl: './app.component.html',
styleUrl: './app.component.scss',
providers: [ToolbarService],
})
@ViewChild("#documenteditor_default")
private container!: DocumentEditorContainerComponent;
private collaborativeEditingHandler!: CollaborativeEditingHandler;
private serviceUrl: string = "http://localhost:5212/";
onCreated() {
// Enable collaborative editing in Document Editor.
this.container.documentEditor.enableCollaborativeEditing = true;
this.collaborativeEditingHandler = this.container.documentEditor.collaborativeEditingHandlerModule;
this.loadDocumentFromServer();
}
```

Step 2: Configure SignalR to send and receive changes

To broadcast the changes made and receive changes from remote users, configure SignalR like below.

```
`typescript
public connection?: HubConnection;
public connectionId: string = "";
initializeSignalR = (): void => {
  // SignalR connection
  this.connection = new HubConnectionBuilder().withUrl(this.serviceUrl + 'documenteditorhub', {
    skipNegotiation: true,
    transport: HttpTransportType.WebSockets
  }).withAutomaticReconnect().build();
  //Event handler for signalR connection
  this.connection.on('dataReceived', this.onDataRecived.bind(this));
  this.connection.onclose(async () => {
    if (this.connection && this.connection.state === HubConnectionState.Disconnected) {
      alert('Connection lost. Please relod the browser to continue.');
```

```

}
}
}
//Apply the remote action in DocumentEditor
this.collaborativeEditingHandler.applyRemoteAction(action, data);
}
}
public connectToRoom(data: any) {
  try {
    if (this.connection) {
      // start the connection.
      this.connection.start().then(() => {
        // Join the room.
        if (this.connection) {
          this.connection.send('JoinGroup', { roomName: data.roomName, currentUser: data.currentUser });
        }
        console.log('server connected!!!');
      });
    }
    } catch (err) {
      console.log(err);
      //Attempting to reconnect in 5 seconds
      setTimeout(this.connectToRoom, 5000);
    }
  };
  `

```

Step 3: Join SignalR room while opening the document

When opening a document, we need to generate a unique ID for each document. These unique IDs are then used to create rooms using SignalR, which facilitates sending and receiving data from the server.

```

`typescript
openDocument(responseText: string, roomName: string): void {
  showSpinner(document.getElementById('container') as HTMLElement);
  let data = JSON.parse(responseText);
  if (this.container) {

```

```

this.collaborativeEditingHandler = this.container.documentEditor.collaborativeEditingHandlerModule;
//Update the room and version information to collaborative editing handler.
this.collaborativeEditingHandler.updateRoomInfo(roomName, data.version, this.serviceUrl +
'api/CollaborativeEditing/');
//Open the document
this.container.documentEditor.open(data.sfddt);
setTimeout(() => {
if (this.container) {
// connect to server using signalR
this.connectToRoom({ action: 'connect', roomName: roomName, currentUser:
this.container.currentUser });
}
});
}
hideSpinner(document.getElementById('container') as HTMLElement);
}
`

```

Step 4: Broadcast current editing changes to remote users

Changes made on the client-side need to be sent to the server-side to broadcast them to other connected users. To send the changes made to the server, use the method shown below from the document editor using the `contentChange` event.

```

`typescript
onContentChange = (args: ContainerContentChangeEventArgs) => {
if (this.collaborativeEditingHandler) {
//Send the editing action to server
this.collaborativeEditingHandler.sendActionToServer(args.operations as Operation[])
}
}
`

```

How to enable collaborative editing in ASP.NET Core

Step 1: Configure SignalR in ASP.NET Core

We are using Microsoft SignalR to broadcast the changes. Please add the following configuration to your application's Program.cs file.

```

`csharp
using Microsoft.Azure.SignalR;

```

```

.....
builder.Services.AddSignalR();
.....
.....
.....
app.MapHub<DocumentEditorHub>("/documenteditorhub");
.....
.....
`

```

Step 2: Configure SignalR hub to create room for collaborative editing session

To manage groups for each document, create a folder named "Hub" and add a file named "DocumentEditorHub.cs" inside it. Add the following code to the file to manage SignalR groups using room names.

Join the group by using unique id of the document by using `JoinGroup` method.

```

`csharp
static Dictionary<string, ActionInfo> userManager = new Dictionary<string, ActionInfo>();
internal static Dictionary<string, List<ActionInfo>> groupManager = new Dictionary<string,
List<ActionInfo>>>();

// Join to the specified room name
public async Task JoinGroup(ActionInfo info)
{
    if (!userManager.ContainsKey(Context.ConnectionId))
    {
        userManager.Add(Context.ConnectionId, info);
    }
    info.ConnectionId = Context.ConnectionId;
    //Add the current connected use to the specified group
    await Groups.AddToGroupAsync(Context.ConnectionId, info.RoomName);
    if (groupManager.ContainsKey(info.RoomName))
    {
        await Clients.Caller.SendAsync("dataReceived", "addUser", groupManager[info.RoomName]);
    }
    lock (groupManager)
    {

```

```
if (groupManager.ContainsKey(info.RoomName))
{
    groupManager[info.RoomName].Add(info);
}
else
{
    List<ActionInfo> actions = new List<ActionInfo>
    {
        info
    };
    groupManager.Add(info.RoomName, actions);
}

// Notify other users in the group about new user joined the collaborative editing session.
Clients.GroupExcept(info.RoomName, Context.ConnectionId).SendAsync("dataReceived", "addUser",
info);
}
```

Handle user disconnection using SignalR.

```
`csharp
//Handle disconnection from group.
public override Task OnDisconnectedAsync(Exception? e)
{
    string roomName = userManager[Context.ConnectionId].RoomName;
    if (groupManager.ContainsKey(roomName))
    {
        groupManager[roomName].Remove(userManager[Context.ConnectionId]);
        if (groupManager[roomName].Count == 0)
        {
            groupManager.Remove(roomName);
        }
        //If all user disconnected from current room. Auto save the change to source document.
        CollaborativeEditingController.UpdateOperationsToSourceDocument(roomName,
        "<<documentpath>>", false);
    }
}
```

```

}
if (userManager.ContainsKey(Context.ConnectionId))
{
    //Notify other user in the group about user exit the collaborative editing session
    Clients.OthersInGroup(roomName).SendAsync("dataReceived", "removeUser", Context.ConnectionId);
    Groups.RemoveFromGroupAsync(Context.ConnectionId, roomName);
    userManager.Remove(Context.ConnectionId);
}
return base.OnDisconnectedAsync(e);
}
`

```

Step 3: Configure Microsoft SQL database connection string in application level

Configure the SQL database that stores temporary data for the collaborative editing session. Provide the SQL database connection string in `appsettings.json` file.

```

`json
.....
"ConnectionStrings": {
  "DocumentEditorDatabase": "<SQL server connection string>"
}
.....
`

```

Step 4: Configure Web API actions for collaborative editing

Import File

1. When opening a document, create a database table to store temporary data for the collaborative editing session. 2. If the table already exists, retrieve the records from the table and apply them to the WordDocument instance using the `UpdateActions` method before converting it to the SFDT format.

```

`csharp
public string ImportFile([FromBody] FileInfo param)
{
    .....
    .....
    DocumentContent content = new DocumentContent();
    .....
    //Get source document from database/file system/blob storage
    WordDocument document = GetDocumentFromDatabase(param.fileName, param.documentOwner);

```

.....

```
//Get temporary records from database
List<ActionInfo> actions = CreatedTable(param.fileName);
if(actions!=null)
{
//Apply temporary data to the document.
document.UpdateActions(actions);
}
string json = Newtonsoft.Json.JsonConvert.SerializeObject(document);
content.version = 0;
content.sfdt = json;
return Newtonsoft.Json.JsonConvert.SerializeObject(content);
}
、
```

[Update editing records to database.](#)

Each edit operation made by the user is sent to the server and is pushed to the database. Each operation receives a version number after being inserted into the database.

After inserting the records to the server, the position of the current editing operation must be transformed against any previous editing operations not yet synced with the client using the TransformOperation method.

After performing the transformation, the current operation is broadcast to all connected users within the group.

`csharp

```
public async Task<ActionInfo> UpdateAction([FromBody] ActionInfo param)
{
try
{
ActionInfo modifiedAction = AddOperationsToTable(param);
//After transformation broadcast changes to all users in the group
await _hubContext.Clients.Group(param.RoomName).SendAsync("dataReceived", "action",
modifiedAction);
return modifiedAction;
}
catch
{

```



```

return null;
}
}
private ActionInfo AddOperationsToTable(ActionInfo action)
{
    int clientVersion = action.Version;
    string tableName = action.RoomName;
    .....
    .....
    .....
    .....
    List<ActionInfo> actions = GetOperationsQueue(table);
    foreach (ActionInfo info in actions)
    {
        if (!info.IsTransformed)
        {
            CollaborativeEditingHandler.TransformOperation(info, actions);
        }
    }
    action = actions[actions.Count - 1];
    action.Version = updateVersion;
    //Return the transformed operation to broadcast it to other clients.
    return action;
}
`

```

[Add Web API to get previous operation as a backup to get lost operations](#)

On the client side, messages broadcast using SignalR may be received in a different order, or some operations may be missed due to network issues. In these cases, we need a backup method to retrieve missing records from the database.

Using the following method, we can retrieve all operations after the last successful client-synced version and return all missing operations to the requesting client.

```

`csharp
public async Task<ActionInfo> UpdateAction([FromBody] ActionInfo param)
{

```

```
try
{
    ActionInfo modifiedAction = AddOperationsToTable(param);
    //After transformation broadcast changes to all users in the group
    await _hubContext.Clients.Group(param.RoomName).SendAsync("dataReceived", "action",
        modifiedAction);
    return modifiedAction;
}
catch
{
    return null;
}
}

private ActionInfo AddOperationsToTable(ActionInfo action)
{
    int clientVersion = action.Version;
    string tableName = action.RoomName;

    .....

    .....

    .....

    .....

    List<ActionInfo> actions = GetOperationsQueue(tableName);
    foreach (ActionInfo info in actions)
    {
        if (!info.IsTransformed)
        {
            CollaborativeEditingHandler.TransformOperation(info, actions);
        }
    }

    action = actions[actions.Count - 1];
    action.Version = updateVersion;

    //Return the transformed operation to broadcast it to other clients.
    return action;
}
```

```
}  
`
```

How to perform Scaling in Collaborative Editing.

As the number of user increases, maintaining responsiveness and performance becomes challenging. Scaling tackles this by distributing the workload across resources. You can scale the collaborative editing application using either **Azure SignalR service or Redis backplane service**

1. Scaling with Azure SignalR

Azure SignalR Service is a scalable, managed service for real-time communication in web applications. It enables real-time messaging between web clients (browsers) and your server-side application(across multiple servers).

Below is a code snippet to configure Azure SignalR in an ASP.NET Core application using the **AddAzureSignalR** method

```
`csharp  
builder.Services.AddSignalR().AddAzureSignalR("<your-connection-string>", options => {  
    // Specify the channel name  
    options.Channels.Add("document-editor");  
});  
`
```

2. Scaling with Redis backplane

Using a Redis backplane, you achieve horizontal scaling of your SignalR application. The SignalR leverages Redis to efficiently broadcast messages across multiple servers. This allows your application to handle large user bases with minimal latency.

In the SignalR app, install the following NuGet package:

- **Microsoft.AspNetCore.SignalR.StackExchangeRedis**

Below is a code snippet to configure Redis backplane in an ASP.NET Core application using the **AddStackExchangeRedis** method

```
`csharp  
builder.Services.AddSignalR().AddStackExchangeRedis("<yourRedisconnection_string>");  
`
```

Configure options as needed:

The following example shows how to add a channel prefix in the ConfigurationOptions object.

```
`csharp  
builder.Services.AddDistributedMemoryCache().AddSignalR().AddStackExchangeRedis(connectionString,  
options =>  
{  
    options.Configuration.ChannelPrefix = "document-editor";  
}
```

```
});  
`
```

Full version of the code discussed about can be found in below GitHub location.

GitHub Example: [Collaborative editing examples](#)

Collaborative Editing (preview)

Allows multiple users to work on the same document simultaneously. This can be done in real-time, so that collaborators can see the changes as they are made. Collaborative editing can be a great way to improve efficiency, as it allows team members to work together on a document without having to wait for others to finish their changes.

Prerequisites

The following are needed to enable collaborative editing in Document Editor

- Sock JS
- PostgreSQL database

How to enable collaborative editing in client side

Step 1: Enable collaborative editing in Document Editor

To enable collaborative editing, inject `CollaborativeEditingHandler` and set the property `enableCollaborativeEditing` to true in the Document Editor, like in the code snippet below.

```
`html  

<!-- ./app.component.html -->  

<div>  

  <div id="documenteditor_titlebar" class="e-de-ctn-title" style="height:35px;"></div>  

  <ejs-documenteditorcontainer #documenteditor_default [enableToolbar]="true"  

  (created)="onCreated()" (contentChange)="onContentChange($event)" height="600px" style="display:  

  block;"> </ejs-documenteditorcontainer>  

</div>  

`  

`typescript  

import { DocumentEditorContainerModule, ToolbarService, DocumentEditorContainerComponent,  

  ContainerContentChangeEventArgs, Operation } from '@syncfusion/ej2-angular-documenteditor';  

//Inject collaborative editing module.  

DocumentEditorComponent.Inject(CollaborativeEditingHandler);  

@Component({  

  selector: 'app-root',  

  standalone: true,  

  imports: [DocumentEditorContainerModule, CommonModule, RouterOutlet],  

  templateUrl: './app.component.html',
```

```
styleUrl: './app.component.scss',
providers: [ToolbarService],
})
@ViewChild("#documenteditor_default")
private collaborativeEditingHandler!: CollaborativeEditingHandler;
onCreated() {
// Enable collaborative editing in Document Editor.
this.container.documentEditor.enableCollaborativeEditing = true;
}
`
```

Step2: Configure SockJS to send and receive changes

To broadcast the changes made and receive changes from remote users, configure SockJS like below.

```
`typescript
import { Stomp } from '@stomp/stompjs';
import * as SockJS from 'sockjs-client';
//Initialize SockJS
initializeSockJS(): void {
let ws = new SockJS(this.webSocketEndPoint);
this.stompClient = Stomp.over(ws);
this.stompClient.connect({}, () => {
console.log('WebSocket connection established.');
```

```
this.onConnected();
});
}
// Subscribe to the topic.
onConnected() {
if (this.stompClient.connected) {
// Subscribe to the specific user
var roomName = this.fileName;
this.stompClient.subscribe('/topic/public/' + this.fileName, this.onMessageReceived);
this.joinGroup(this.fileName);
}
}
```

```
//Receive the remote action and apply to currenty document.
onMessageReceived = (data: any) => {
  var content = JSON.parse(data.body);
  if (content.payload.operations != null) {
    this.collaborativeEditingHandler.applyRemoteAction("action", content.payload);
  }
}
`
```

Step 3: Subscribe to specific topic while opening the document

When opening a document, we need to generate a unique ID for each document. These unique IDs are then used to create rooms using SockJS, which facilitates sending and receiving data from the server.

```
`typescript
//Load the document
loadDefaultDocument(): void {
  // Define HTTP headers
  const headers = new HttpHeaders({
    'Content-Type': 'application/json;charset=UTF-8'
  });
  this.http.post<any>(this.serviceurl + 'ImportFile', { "fileName": this.fileName, "documentOwner":
  this.documentOwner }, { headers }).subscribe(
    (response) => {
      // On success
      this.container.documentEditor.open(response.sfdt);
      this.collaborativeEditingHandler.updateRoomInfo(this.fileName, response.version, this.serviceurl);
      setTimeout(() => {
        this.initializeSockJS();
      });
    });
  //Send the user information to the other users that I have joined.
  joinGroup(documentName: any) {
    var userInfo = {
      currentUser: this.username,
      clientVersion: 0,
```

```

roomName: documentName,
connectionId: "",
};
// Send the joinGroup message to the server
this.stompClient.send("/app/join/" + documentName, {}, JSON.stringify(userInfo));
}
`

```

Step 4: Broadcast current editing changes to remote users

Changes made on the client-side need to be sent to the server-side to broadcast them to other connected users. To send the changes made to the server, use the method shown below from the document editor using the `contentChange` event.

```

`typescript
onContentChange = (args: ContainerContentChangeEventArgs) => {
if (this.collaborativeEditingHandler) {
this.collaborativeEditingHandler.sendActionToServer(args.operations as Operation[])
}
}
`

```

How to enable collaborative editing in Java

Step 1: Configure SockJS hub to create room for collaborative editing session.

To manage groups for each document, create a folder named “Hub” and add a file named `DocumentEditorHub.java` inside it. Add the following code to the file to manage SockJS groups using room names.

Join the group by using unique id of the document by using `JoinGroup` method.

```

`java
@MessageMapping("/join/{documentName}")

public void joinGroup(ActionInfo info, SimpMessageHeaderAccessor headerAccessor,
@DestinationVariable String documentName) {

.....

.....

broadcastToRoom(info.getRoomName(), info, headers);
if (!actions.containsKey(connectionId)) {
actions.put(connectionId, info);
}

ArrayList<ActionInfo> actionsList = roomList.computeIfAbsent(documentName, k -> new ArrayList<>());

```

```
// Add the new user info to the list
actionsList.add(info);

.....

broadcastToRoom(docName, actionsList, addUserheaders);
}

public static void broadcastToRoom(String roomName, Object payload, MessageHeaders headers) {
    if (payload instanceof HashMap) {
        HashMap<String, ActionInfo> actionsMap = (HashMap<String, ActionInfo>) payload;
        ArrayList<ActionInfo> actionsList = new ArrayList<>(actionsMap.values());
        messagingTemplate.convertAndSend("/topic/public/" + roomName,
            MessageBuilder.createMessage(actionsList, headers));
    } else {
        messagingTemplate.convertAndSend("/topic/public/" + roomName,
            MessageBuilder.createMessage(payload, headers));
    }
}
\
```

Step 2: Handle user disconnection using SockJS.

```
`java
@EventListener
public void handleWebSocketDisconnectListener(SessionDisconnectEvent event) {
    String sessionId = event.getSessionId();
    HashMap<String, ActionInfo> userDetails = DocumentEditorHub.actions;
    if (userDetails.containsKey(sessionId)) {
        ActionInfo info = userDetails.get(sessionId);
        .....
        .....
        ArrayList<ActionInfo> actionsList = roomList.computeIfAbsent(info.getRoomName(), k -> new
        ArrayList<>());
        for (ActionInfo action : actionsList) {
            if (action.getConnectionId() == sessionId) {
                actionsList.remove(action);
                break;
            }
        }
    }
}
```



```

}
}
if (userDetails.isEmpty()) {
    Connection connection = DriverManager.getConnection(datasourceUrl, datasourceUsername,
datasourcePassword);

    CollaborativeEditingController.updateOperationsToSourceDocument(docName, false, 0, connection,
datasourceAccessKey, datasourceSecretKey, datasourceBucketName);
}
}
}
,

```

Step 3: Configure PostgreSQL database connection string and Bucket s3 in application level.

Configure the PostgreSQL database that stores temporary data and Bucket S3 credential to get the document for the collaborative editing session. Provide the PostgreSQL database connection string and credential for bucket S3 in application.properties file.

```

`java
//PostgreSQL
spring.datasource.url="<PostgreSQL server connection string>"
spring.datasource.username="<PostgreSQL username>"
spring.datasource.password="<PostgreSQL password>"
,

```

Step 4: Configure Web API actions for collaborative editing.

Import File

- When opening a document, create a database table to store temporary data for the collaborative editing session. - If the table already exists, retrieve the records from the table and apply them to the WordDocument instance using the UpdateActions method before converting it to the SFDT format.

```

`java
public String ImportFile(@RequestBody FilesPathInfo file) {
    DocumentContent content = new DocumentContent();
    // Load the document from local.
    WordProcessorHelper document =
    WordProcessorHelper.load(classLoader.getResourceAsStream("static/files/" + file.getFileName()), false);
    // table in database to store temporary data in collaborative editing session.
    ArrayList<ActionInfo> actions = createRecordForCollaborativeEditing(file.getRoomName(),
lastSyncedVersion_out);
    if (actions != null && actions.size() > 0) {

```

```
document.updateActions(actions);
}
.....
return data;
}
、
```

Update editing records to database

- Each edit operation made by the user is sent to the server and is pushed to the database. Each operation receives a version number after being inserted into the database. - After inserting the records to the server, the position of the current editing operation must be transformed against any previous editing operations not yet synced with the client using the TransformOperation method. - After performing the transformation, the current operation is broadcast to all connected users within the group.

```
`java
public ActionInfo UpdateAction(@RequestBody ActionInfo param) {
    ActionInfo transformedAction = addOperationsToTable(param.getRoomName());
    HashMap<String, Object> action = new HashMap<>();
    action.put("action", "updateAction");
    DocumentEditorHub.broadcastToRoom(roomName, transformedAction, new MessageHeaders(action));
    return transformedAction;
}

private ActionInfo addOperationsToTable(ActionInfo action) {
    int clientVersion = action.getVersion();
    String tableName = action.getRoomName();
    .....
    .....
    ArrayList<ActionInfo> actions = getOperationsQueue(tableName);
    for (ActionInfo info : actions) {
        if (!info.isTransformed()) {
            CollaborativeEditingHandler.transformOperation(info, actions);
        }
    }
    action = actions.get(actions.size() - 1);
    action.setVersion(updateVersion);
    return action;
}
```

```
}
,
```

[Add Web API to get previous operation as a backup to get lost operations](#)

On the client side, messages send from server using SockJS may be received in a different order, or some operations may be missed due to network issues. In these cases, we need a backup method to retrieve missing records from the database.

Using the following method, we can retrieve all operations after the last successful client-synced version and return all missing operations to the requesting client.

```
`java
public ActionInfo UpdateAction(@RequestBody ActionInfo param) {
    ActionInfo transformedAction = addOperationsToTable(param.getRoomName());
    HashMap<String, Object> action = new HashMap<>();
    action.put("action", "updateAction");
    DocumentEditorHub.broadcastToRoom(roomName, transformedAction, new MessageHeaders(action));
    return transformedAction;
}

private ActionInfo addOperationsToTable(ActionInfo action)
throws Exception {
    int clientVersion = action.getVersion();
    String tableName = action.getRoomName();
    .....
    .....
    ArrayList<ActionInfo> actions = getOperationsQueue(tableName);
    for (ActionInfo info : actions) {
        if (!info.isTransformed()) {
            CollaborativeEditingHandler.transformOperation(info, actions);
        }
    }
    action = actions.get(actions.size() - 1);
    action.setVersion(updateVersion);
    return action;
}
,
```

How to perform Scaling in Collaborative Editing.

Role of Scaling in Collaborative editing

As the number of users increases, collaborative application face challenges in maintaining responsiveness and performance. This is where scaling becomes crucial. Scaling refers to the ability of an application to handle growing demands by effectively distributing the workload across multiple resources.

During scaling the users may connected to different servers, so collaborative editing application introduces a specific challenge like, updating the edit operations to all the users connected in different serves. To overcome this issue you need to use **Redis Cache pub/sub** for message relay(syncing the editing operations to the users connected to different server instance)

Use of Redis Pub/Sub in scaling environment

Redis offers Pub/Sub functionality. The publish/subscribe (pub/sub) pattern provides asynchronous communication among multiple AWS services without creating interdependency. When a user edits a document, the application can publish the changes to a Redis channel. Clients (in different server instances) subscribed to that channel receive real-time updates, reflecting the changes in their document views.

Steps to configure Redis in Collaborative Editing Application

Refer to the below steps to know about the Redis pub/sub implementation to sync the messages.

Step 1: Configure Redis in application level to establish the connection.

```
`java
//Redis configuration
spring.datasource.redishost= "<Redis host link>"
spring.datasource.redisport= "<Redis port number>"
`
```

Step 2: Publish each editing operation to a Redis channel

Publish each editing operation to Redis channel with the room name. This will send notifications to all the users(in different servers) subscribed to that specific channel. Refer to the publishToRedis() method in DocumentEditorHub.Java for details.

```
`java
try (Jedis jedis = RedisSubscriber.jedisPool.getResource()) {
jedis.publish("collaborativeediting", new
com.fasterxml.jackson.databind.ObjectMapper().writeValueAsString(payload));
break;
} catch (JedisConnectionException e) {
}
`
```

Step 3: Subscribe to the specific channel using the Redis Cache 'Subscribe'

Redis cache will be initialized and subscribe to the specific channel using the Redis Cache 'Subscribe' option. This ensures that users in any server will get notified when an editing operation is published to the Redis cache using the onMessage() API. Refer to the code snippet in RedisSubscriber.Java for details.

```
`java
@PostConstruct
public void subscribeToInstanceChannel() {
//Subscriber to collaborativeediting
String channel = "collaborativeediting";
new Thread(() -> {
JedisPoolConfig poolConfig = new JedisPoolConfig();
jedisPool = new JedisPool(poolConfig, REDISHOST, REDISPORT);
try (Jedis jedis = jedisPool.getResource()) {
jedis.subscribe(new JedisPubSub() {
@Override
public void onMessage(String channel, String message) {
// Message will be broadcasted to all the users connected to that room using sockjs
DocumentEditorHub.broadcastToRoom(action.getRoomName(), action, updateActionheaders);
} catch (JsonProcessingException e) {
e.printStackTrace();
}
}
@Override
public void onSubscribe(String channel, int subscribedChannels) {
System.out.println("Subscribed to channel: " + channel);
}
}, channel);
} catch (JedisConnectionException e) {
// Handle the connection exception
System.out.println("Connection failed. Retrying ...");
}
}).start();
}
,`
```

Full version of the code discussed about can be found in below GitHub location.

GitHub Example: [Collaborative editing examples](#)

View in Angular Document Editor Component

Web Layout

DocumentEditorContainer component allows you to change the view to web layout and print using the [layoutType](#) property with the supported [LayoutType](#).

```
`typescript
```

```
/
```

- Add below codes in app.component.ts file

```
*/
```

```
@Component({
```

```
selector: 'app-root',
```

```
templateUrl: '<ejs-documenteditorcontainer #documenteditor_default [enableToolbar]=true  
(created)="onCreate()" height="600px" style="display:block;"></ejs-documenteditorcontainer>',
```

```
encapsulation: ViewEncapsulation.None,
```

```
providers: [ToolbarService]
```

```
})
```

```
export class AppComponent {
```

```
@ViewChild('documenteditor_default')
```

```
public container: DocumentEditorContainerComponent;
```

```
onCreate(): void {
```

```
this.container.serviceUrl = 'https://ej2services.syncfusion.com/production/web-  
services/api/documenteditor/';
```

```
this.container.layoutType='Continuous';
```

```
}
```

```
}
```

```
,
```

Note: Default value of [layoutType](#) in DocumentEditorContainer component is [Pages](#).

Ruler

Using ruler we can refer to setting specific margins, tab stops, or indentations within a document to ensure consistent formatting in Document Editor.

The following example illustrates how to enable ruler in Document Editor

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { DocumentEditorAllModule } from '@syncfusion/ej2-angular-  
documenteditor'
```

```

import { Component, ViewEncapsulation, ViewChild, OnInit } from
'@angular/core';
import { DocumentEditorComponent } from '@syncfusion/ej2-angular-
documenteditor';
@Component({
  imports: [

    ButtonModule,
    DocumentEditorAllModule
  ],
  standalone: true,
  selector: "app-container",
  template: `<button
id='container_ruler_button'(click)="onClick(this)">Show/Hide Ruler</button>
<ejs-documenteditorcontainer #documenteditor
serviceUrl="https://ej2services.syncfusion.com/production/web-
services/api/documenteditor/" height="600px" isReadOnly: false
style="width:100%;display:block" [enableToolbar]=true
[documentEditorSettings]= "enableRuler"> </ejs-documenteditorcontainer>`,
  encapsulation: ViewEncapsulation.None,
  providers: []
})
export class AppComponent implements OnInit {
  @ViewChild("documenteditor")
  public documentEditor: DocumentEditorComponent;
  ngOnInit(): void {
    this.documentEditor.enableAllModules();
  }
  public enableRuler = { showRuler: true };
  onClick(args: any):void {
    this.documentEditor.documentEditorSettings.showRuler =
!this.documentEditor.documentEditorSettings.showRuler;
  }
}

```

MAIN.TS

```

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

```

Heading Navigation Pane

Using the heading navigation pane allows users to swiftly navigate documents by heading, enhancing their ability to move through the document efficiently.

The following example demonstrates how to enable the heading navigation pane in a document editor.

`typescript

```

import { Component, OnInit } from '@angular/core';
import { ToolbarService } from '@syncfusion/ej2-angular-documenteditor';
@Component({

```

```

selector: 'app-root',
// specifies the template string for the DocumentEditorContainer component
template: <ejs-documenteditorcontainer
serviceUrl="https://ej2services.syncfusion.com/production/web-
services/api/documenteditor/" height="600px" style="display:block"
[documentEditorSettings]= "settings" [enableToolbar]=true> </ejs-documenteditorcontainer>,
providers: [ToolbarService]
})
export class AppComponent implements OnInit {
public settings = {showNavigationPane : true};
ngOnInit(): void {
}
}
`

```

How To

Override the keyboard shortcuts in Angular Document editor component

Document Editor triggers the [keyDown](#) event every time when any key is entered and provides an instance of [DocumentEditorKeyDownEventArgs](#). You can use the [isHandled](#) property to override the keyboard shortcut behavior.

Preventing default keyboard shortcut

The following code shows how to prevent the **CTRL + C** keyboard shortcut for copying selected content in document editor.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { DocumentEditorAllModule } from '@syncfusion/ej2-angular-
documenteditor'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import {
    DocumentEditorComponent, SfdtExportService, SelectionService,
    EditorService, DocumentEditorKeyDownEventArgs
} from '@syncfusion/ej2-angular-documenteditor';
@Component({
imports: [
    ButtonModule,
    DocumentEditorAllModule
],
standalone: true,
selector: 'app-container',
template: `<div>

```



```

    <ejs-documenteditor #document_editor height="330px"
    style="width:100%;display:block" [isReadOnly]=false [enableSelection]=true
    [enableSfdtExport]=true [enableEditor]=true (keyDown)="onKeyDown($event)">
    </ejs-documenteditor>
  </div>`,
  encapsulation: ViewEncapsulation.None,
  //Inject require services.
  providers: [SelectionService, EditorService, SfdtExportService]
})
export class AppComponent {
  public onKeyDown(args: DocumentEditorKeyDownEventArgs): void {
    let keyCode: number = args.event.which || args.event.keyCode;
    let isCtrlKey: boolean = (args.event.ctrlKey ||
args.event.metaKey) ? true : ((keyCode === 17) ? true : false);
    //67 is the character code for 'C'
    if (isCtrlKey && keyCode === 67) {
      //To prevent copy operation set isHandled to true
      args.isHandled = true;
    }
  }
}

```

MAIN.TS

```

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

```

Override or define the keyboard shortcut

Override or define a new keyboard shortcut behavior instead of preventing the keyboard shortcut.

For example, **Ctrl + S** keyboard shortcut saves the document in SFDT format by default, and there is no behavior for **Ctrl + Alt + S**. The following code demonstrates how to override the **Ctrl + S** shortcut to save a document in DOCX format and define **Ctrl + Alt + S** to save the document in SFDT format.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { DocumentEditorAllModule } from '@syncfusion/ej2-angular-
documenteditor'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import {
  DocumentEditorComponent, SelectionService, EditorService,
  DocumentEditorKeyDownEventArgs, SfdtExportService, WordExportService
} from '@syncfusion/ej2-angular-documenteditor';
@Component({
  imports: [
    ButtonModule,
    DocumentEditorAllModule
  ],
  standalone: true,

```

```

        selector: 'app-container',
        template: `<div>
          <ejs-documenteditor #document_editor height="330px" style="width:
100%;display:block" [isReadOnly]=false [enableSelection]=true
[enableSfdtExport]=true [enableEditor]=true (keyDown)="onKeyDown($event)">
          </ejs-documenteditor>
        </div>`,
        encapsulation: ViewEncapsulation.None,
        providers: [SelectionService, EditorService, SfdtExportService,
WordExportService]
    })
    export class AppComponent {
        @ViewChild('document_editor')
        public documentEditor?: DocumentEditorComponent;
        public onKeyDown(args: DocumentEditorKeyDownEventArgs): void {
            let keyCode: number = args.event.which || args.event.keyCode;
            let isCtrlKey: boolean = (args.event.ctrlKey ||
args.event.metaKey) ? true : ((keyCode === 17) ? true : false);
            let isAltKey: boolean = args.event.altKey ? args.event.altKey :
((keyCode === 18) ? true : false);
            // 83 is the character code for 'S'
            if (isCtrlKey && !isAltKey && keyCode === 83) {
                //To prevent default save operation, set the isHandled
property to true
                args.isHandled = true;
                //Download the document in docx format.
                (this.documentEditor as
DocumentEditorComponent).save('sample', 'Docx');
                args.event.preventDefault();
            } else if (isCtrlKey && isAltKey && keyCode === 83) {
                //Download the document in sfdt format.
                (this.documentEditor as
DocumentEditorComponent).save('sample', 'Sfdt');
            }
        }
    }
}

```

MAIN.TS

```

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

```

Customize context menu in Angular Document editor component

[How to customize context menu in Document Editor](#)

Document Editor allows you to add custom option in context menu. It can be achieved by using the [addCustomMenu\(\)](#) method and custom action is defined using the [customContextMenuSelect](#)

Add Custom Option

The following code shows how to add custom option in context menu.

`typescript

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

```
import { ToolbarService ,DocumentEditorContainerComponent, CustomContentMenuEventArgs} from
 '@syncfusion/ej2-angular-documenteditor';
import { MenuItemModel } from '@syncfusion/ej2-navigations';
@Component({
  selector: 'app-root',
  // specifies the template string for the DocumentEditorContainer component
  template: <ejs-documenteditorcontainer #documenteditor_default
  serviceUrl="https://ej2services.syncfusion.com/production/web-
  services/api/documenteditor/" height="600px" style="display:block"
  [documentEditorSettings]= "fontFamilies" [enableToolbar]=true (created)="onCreate()"> </ejs-
  documenteditorcontainer>,
  providers: [ToolbarService]
})
export class AppComponent implements OnInit {
  @ViewChild('documenteditor_default')
  public container: DocumentEditorContainerComponent;
  public fontFamilies={fontFamilies :['Algerian', 'Arial', 'Calibri', 'Cambria', 'Windings']};
  ngOnInit(): void {
  }
  onCreate() {
    // creating Custom Options
    let menuItems: MenuItemModel[] = [
    {
      text: 'Search In Google',
      id: 'searchingoogle',
      iconCss: 'e-icons e-de-ctnr-find'
    }
  ];
    // adding Custom Options
    this.container.documentEditor.contextMenu.addCustomMenu(menuItems, false);
    // custom Options Select Event
    this.container.documentEditor.customContextMenuSelect = (args: any): void => {
    // custom Options Functionality
    let id: string = this.container.documentEditor.element.id;
    switch (args.id) {
```

```

case id + 'searchingoogle':
let searchContent: string = this.container.documentEditor.selection.text;
if (!this.container.documentEditor.selection.isEmpty && /\S/.test(searchContent)) {
window.open('http://google.com/search?q=' + searchContent);
}
break;
}
};
}
}
,

```

Customize custom option in context menu

Document Editor allows you to customize the added custom option and also to hide/show default context menu.

Hide default context menu items

Using [addCustomMenu\(\)](#) method, you can hide the default context menu. By setting second parameter as true.

The following code shows how to hide default context menu and add custom option in context menu.

```

`typescript
@Component({
selector: 'app-root',
// specifies the template string for the DocumentEditorContainer component
template: <ejs-documenteditorcontainer #documenteditor_default
serviceUrl="https://ej2services.syncfusion.com/production/web-
services/api/documenteditor/" height="600px" style="display:block"
[documentEditorSettings]= "fontFamilies" [enableToolbar]=true (created)="onCreate()"> </ejs-
documenteditorcontainer>,
providers: [ToolbarService]
})
export class AppComponent implements OnInit {
@ViewChild('documenteditor_default')
public container: DocumentEditorContainerComponent;
public fontFamilies={fontFamilies :['Algerian', 'Arial', 'Calibri', 'Cambria', 'Windings']};
ngOnInit(): void {
}
}

```

```

onCreate() {
// creating Custom Options
let menuItems: MenuItemModel[] = [
{
text: 'Search In Google',
id: 'searchingoogle',
iconCss: 'e-icons e-de-ctnr-find'
}];
// adding Custom Options
this.container.documentEditor.contextMenu.addCustomMenu(menuItems, true);
}
}
`

```

Customize added context menu items

The following code shows how to hide/show added custom option in context menu using the [customContextMenuBeforeOpen](#).

```

`typescript
@Component({
selector: 'app-root',
// specifies the template string for the DocumentEditorContainer component
template: <ejs-documenteditorcontainer #documenteditor_default
serviceUrl="https://ej2services.syncfusion.com/production/web-
services/api/documenteditor/" height="600px" style="display:block"
[documentEditorSettings]= "fontFamilies" [enableToolbar]=true (created)="onCreate()"> </ejs-
documenteditorcontainer>,
providers: [ToolbarService]
})
export class AppComponent implements OnInit {
@ViewChild('documenteditor_default')
public container: DocumentEditorContainerComponent;
public fontFamilies={fontFamilies :['Algerian', 'Arial', 'Calibri', 'Cambria', 'Windings']};
ngOnInit(): void {
}
onCreate() {
debugger;

```

```
// creating Custom Options
let menuItems: MenuItemModel[] = [
{
text: 'Search In Google',
id: 'searchingoogle',
iconCss: 'e-icons e-de-ctnr-find'
}
];

// adding Custom Options
this.container.documentEditor.contextMenu.addCustomMenu(menuItems, false);

// custom Options Select Event
this.container.documentEditor.customContextMenuSelect = (args: any): void => {
// custom Options Functionality
let id: string = this.container.documentEditor.element.id;
switch (args.id) {
case id + 'searchingoogle':
let searchContent: string = this.container.documentEditor.selection.text;
if (!this.container.documentEditor.selection.isEmpty && /\S/.test(searchContent)) {
window.open('http://google.com/search?q=' + searchContent);
}
break;
}
};

// custom options hide/show functionality
this.container.documentEditor.customContextMenuBeforeOpen = (args: any): void => {
let search: any = document.getElementById(args.ids[0]);
search.style.display = 'none';
let searchContent: string = this.container.documentEditor.selection.text;
if (!this.container.documentEditor.selection.isEmpty && /\S/.test(searchContent)) {
search.style.display = 'block';
}
};
}
```

The following is the output of custom context menu with customization.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DocumentEditorContainerModule } from '@syncfusion/ej2-angular-documenteditor'
import { Component, OnInit, ViewChild } from '@angular/core';
import { ToolbarService ,DocumentEditorContainerComponent, CustomContentMenuEventArgs} from '@syncfusion/ej2-angular-documenteditor';
import { MenuItemModel } from '@syncfusion/ej2-navigations';
@Component({
  imports: [

    DocumentEditorContainerModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the DocumentEditorContainer component
  template: `<ejs-documenteditorcontainer #documenteditor_default
serviceUrl="https://ej2services.syncfusion.com/production/web-services/api/documenteditor/" height="600px" style="display:block"
[documentEditorSettings]= "fontFamilies" [enableToolbar]=true
(created)="onCreate()"> </ejs-documenteditorcontainer>`,
  providers: [ToolbarService]
})
export class AppComponent implements OnInit {
  @ViewChild('documenteditor_default')
  public container?: DocumentEditorContainerComponent;
  public fontFamilies={fontFamilies :['Algerian', 'Arial', 'Calibri', 'Cambria', 'Windings']};
  ngOnInit(): void {
  }
  onCreate() {
    debugger;
    // creating Custom Options
    let menuItems: MenuItemModel[] = [
      {
        text: 'Search In Google',
        id: 'search_in_google',
        iconCss: 'e-icons e-de-ctnr-find'
      }
    ];
    // adding Custom Options
    (this.container as DocumentEditorContainerComponent)
    .documentEditor.contextMenu.addCustomMenu(menuItems, false);
    // custom Options Select Event
    (this.container as DocumentEditorContainerComponent)
    .documentEditor.customContextMenuSelect = (args: any): void => {
      // custom Options Functionality
      let id: string = (this.container as DocumentEditorContainerComponent)
    .documentEditor.element.id;
      switch (args.id) {
        case id + 'search in google':
```

```

        let searchContent: string = (this.container as
DocumentEditorContainerComponent ).documentEditor.selection.text;
        if (!(this.container as DocumentEditorContainerComponent
).documentEditor.selection.isEmpty && /\S/.test(searchContent)) {
            window.open('http://google.com/search?q=' +
searchContent);
        }
        break;
    }
};
// custom options hide/show functionality
(this.container as DocumentEditorContainerComponent
).documentEditor.customContextMenuBeforeOpen = (args: any): void => {
    let search: any = document.getElementById(args.ids[0]);
    search.style.display = 'none';
    let searchContent: string = (this.container as
DocumentEditorContainerComponent ).documentEditor.selection.text;
    if (!(this.container as DocumentEditorContainerComponent
).documentEditor.selection.isEmpty && /\S/.test(searchContent)) {
        search.style.display = 'block';
    }
};
}
}

```

MAIN.TS

```

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

```

Customize tool bar in Angular Document editor component

How to customize existing toolbar in DocumentEditorContainer

Document Editor Container allows you to customize(add, show, hide, enable, and disable) existing items in a toolbar.

- Add - New items can be defined by [CustomToolbarItemModel](#) and with existing items in [toolbarItems](#) property. Newly added item click action can be defined in [toolbarclick](#).
- Show, Hide - Existing items can be shown or hidden using the [toolbarItems](#) property. Pre-defined toolbar items are available with [ToolbarItem](#).
- Enable, Disable - Toolbar items can be enabled or disable using [enableItems](#)

`typescript

```

import { Component, ViewChild } from '@angular/core';
import { ClickEventArgs } from '@syncfusion/ej2-navigations'
import { DocumentEditorContainerComponent, ToolbarService, CustomToolbarItemModel } from
 '@syncfusion/ej2-angular-documenteditor';
@Component({

```



```

selector: 'app-root',

template: '<ejs-documenteditorcontainer [height]="600px" [toolbarItems]=items
(toolbarClick)="onToolbarClick($event)" #documenteditor_default style="display:block;"
[enableToolbar]=true></ejs-documenteditorcontainer>',

styleUrls: ['./app.component.css'],

providers: [ToolbarService]

})

export class AppComponent {
  @ViewChild('documenteditor_default', { static: true })
  container: DocumentEditorContainerComponent;

  //Custom toolbat item.
  public toolItem: CustomToolbarItemModel = {
    prefixIcon: "e-de-ctnr-lock",
    tooltipText: "Disable Image",
    text: "Disable Image",
    id: "Custom"
  };

  public items = [this.toolItem, 'Undo', 'Redo', 'Separator', 'Image', 'Table', 'Hyperlink', 'Bookmark',
    'Comments', 'TableOfContents', 'Separator', 'Header', 'Footer', 'PageSetup', 'PageNumber', 'Break',
    'Separator', 'Find', 'Separator', 'LocalClipboard', 'RestrictEditing'];

  public onToolbarClick(args: ClickEventArgs): void {
    switch (args.item.id) {
      case 'Custom':
        //Disable image toolbar item.
        this.container.toolbar.enableItems(4, false);
        break;
    }
  };
}

```

Note: Default value of toolbarItems is ['New', 'Open', 'Separator', 'Undo', 'Redo', 'Separator', 'Image', 'Table', 'Hyperlink', 'Bookmark', 'TableOfContents', 'Separator', 'Header', 'Footer', 'PageSetup', 'PageNumber', 'Break', 'Separator', 'Find', 'Separator', 'Comments', 'TrackChanges', 'Separator', 'LocalClipboard', 'RestrictEditing', 'Separator', 'FormFields', 'UpdateFields'].

Change document view in Angular Document editor component

How to change the document view in DocumentEditor component

DocumentEditor allows you to change the view to web layout and print using the [layoutType](#) property with the supported [LayoutType](#).

```
`typescript
import { Component, ViewChild, ViewEncapsulation } from '@angular/core';
import {
  DocumentEditorComponent, PrintService, SfdtExportService, WordExportService, TextExportService,
  SelectionService,
  SearchService, EditorService, ImageResizerService, EditorHistoryService, ContextMenuService,
  OptionsPaneService, HyperlinkDialogService, TableDialogService, BookmarkDialogService,
  TableOfContentsDialogService,
  PageSetupDialogService, StyleDialogService, ListDialogService, ParagraphDialogService,
  BulletsAndNumberingDialogService,
  FontDialogService, TablePropertiesDialogService, BordersAndShadingDialogService,
  TableOptionsDialogService,
  CellOptionsDialogService, StylesDialogService
} from '@syncfusion/ej2-angular-documenteditor';
@Component({
  selector: 'app-root',
  template: `<ejs-documenteditor id="container" #documenteditor_default
  serviceUrl="https://ej2services.syncfusion.com/production/web-services/api/documenteditor/"
  height="330px" style="display:block" [isReadOnly]=false [enableSelection]=true
  [enablePrint]=true [enableSfdtExport]=true [enableWordExport]=true [enableOptionsPane]=true
  [enableContextMenu]=true
  [enableHyperlinkDialog]=true [enableBookmarkDialog]=true [enableTableOfContentsDialog]=true
  [enableSearch]=true
  [enableParagraphDialog]=true [enableListDialog]=true [enableTablePropertiesDialog]=true
  [enableBordersAndShadingDialog]=true
  [enablePageSetupDialog]=true [enableStyleDialog]=true [enableFontDialog]=true
  [enableTableOptionsDialog]=true
  [enableTableDialog]=true [enableImageResizer]=true [enableEditor]=true [enableEditorHistory]=true
  (created)="onCreate()">
</ejs-documenteditor>`,
  encapsulation: ViewEncapsulation.None,
  providers: [PrintService, SfdtExportService, WordExportService, TextExportService, SelectionService,
  SearchService, EditorService,
```

```

ImageResizerService, EditorHistoryService, ContextMenuService, OptionsPaneService,
HyperlinkDialogService, TableDialogService,

BookmarkDialogService, TableOfContentsDialogService, PageSetupDialogService, StyleDialogService,
ListDialogService,

ParagraphDialogService, BulletsAndNumberingDialogService, FontDialogService,
TablePropertiesDialogService,

BordersAndShadingDialogService, TableOptionsDialogService, CellOptionsDialogService,
StylesDialogService]
})
export class AppComponent {
  @ViewChild('documenteditor_default')
  public container: DocumentEditorComponent;
  onCreate(): void {
    this.container.serviceUrl = 'https://ej2services.syncfusion.com/production/web-
services/api/documenteditor/';
    this.container.layoutType='Continuous';
  }
}
`

```

Note: Default value of [layoutType](#) in DocumentEditor component is [Pages](#).

How to change the document view in DocumentEditorContainer component

DocumentEditorContainer component allows you to change the view to web layout and print using the [layoutType](#) property with the supported [LayoutType](#).

`typescript

/

- Add below codes in app.component.ts file

*/

```

@Component({
  selector: 'app-root',
  templateUrl: '<ejs-documenteditorcontainer #documenteditor_default [enableToolbar]=true
(created)="onCreate()" height="600px" style="display:block;"></ejs-documenteditorcontainer>',
  encapsulation: ViewEncapsulation.None,
  providers: [ToolbarService]
})
export class AppComponent {

```

```

@ViewChild('documenteditor_default')

public container: DocumentEditorContainerComponent;

onCreate(): void {

this.container.serviceUrl = 'https://ej2services.syncfusion.com/production/web-
services/api/documenteditor/';

this.container.layoutType='Continuous';

}

}

`

```

Note: Default value of [layoutType](#) in DocumentEditorContainer component is [Pages](#).

Open default document in Angular Document editor component

In this article, we are going to see how to open a default document when Document Editor & Document Editor Container is initialized.

Opening a default document in DocumentEditor

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { DocumentEditorAllModule } from '@syncfusion/ej2-angular-
documenteditor'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import { DocumentEditorComponent } from '@syncfusion/ej2-angular-
documenteditor';
@Component({
imports: [

        ButtonModule,
        DocumentEditorAllModule
    ],
standalone: true,
    selector: "app-container",
    template: `<ejs-documenteditor #documenteditor height="330px"
style="width:100%;display:block" (created)="onCreate()"></ejs-
documenteditor>`,
    encapsulation: ViewEncapsulation.None,
    providers: []
})
export class AppComponent {
    @ViewChild("documenteditor")
    public documentEditor?: DocumentEditorComponent;
    // load your default document here
    onCreate(): any {
        let sfdt: string =
`{"sections":[{"sectionFormat":{"pageWidth":612,"pageHeight":792,"leftMargin
":72,"rightMargin":72,"topMargin":72,"bottomMargin":72,"differentFirstPage":
false,"differentOddAndEvenPages":false,"headerDistance":36,"footerDistance":
36,"bidi":false},"blocks":[{"paragraphFormat":{"afterSpacing":30,"styleName"
:"Heading

```

```
1","listFormat":{},"characterFormat":{},"inlines":[{"characterFormat":{},"text":
ext":"Adventure Works
Cycles"}]}]},"headersFooters":{"header":{"blocks":[{"paragraphFormat":{"listF
ormat":{},"characterFormat":{},"inlines":[]}}],"footer":{"blocks":[{"paragr
aphFormat":{"listFormat":{},"characterFormat":{},"inlines":[]}}]}]},"charac
terFormat":{"bold":false,"italic":false,"fontSize":11,"fontFamily":"Calibri"
,"underline":"None","strikethrough":"None","baselineAlignment":"Normal","hig
hlightColor":"NoColor","fontColor":"empty","fontSizeBidi":11,"fontFamilyBidi
":"Calibri","allCaps":false},"paragraphFormat":{"leftIndent":0,"rightIndent"
:0,"firstLineIndent":0,"textAlignment":"Left","beforeSpacing":0,"afterSpacin
g":0,"lineSpacing":1.0791666507720947,"lineSpacingType":"Multiple","listForm
at":{},"bidi":false},"defaultTabWidth":36,"trackChanges":false,"enforcement"
:false,"hashValue":"","saltValue":"","formatting":false,"protectionType":"No
Protection","dontUseHTMLParagraphAutoSpacing":false,"formFieldShading":true,
"styles":[{"name":"Normal","type":"Paragraph","paragraphFormat":{"lineSpacin
g":1.149999976158142,"lineSpacingType":"Multiple","listFormat":{},"characte
rFormat":{"fontFamily":"Calibri"},"next":"Normal"},{"name":"Default
Paragraph Font","type":"Character","characterFormat":{}},{name":"Heading 1
Char","type":"Character","characterFormat":{"fontSize":16,"fontFamily":"Cali
bri Light","fontColor":"#2F5496"},"basedOn":"Default Paragraph
Font"},{"name":"Heading
1","type":"Paragraph","paragraphFormat":{"beforeSpacing":12,"afterSpacing":0
,"outlineLevel":"Level1","listFormat":{},"characterFormat":{"fontSize":16,"f
ontFamily":"Calibri
Light","fontColor":"#2F5496"},"basedOn":"Normal","link":"Heading 1
Char","next":"Normal"},{"name":"Heading 2
Char","type":"Character","characterFormat":{"fontSize":13,"fontFamily":"Cali
bri Light","fontColor":"#2F5496"},"basedOn":"Default Paragraph
Font"},{"name":"Heading
2","type":"Paragraph","paragraphFormat":{"beforeSpacing":2,"afterSpacing":6,
"outlineLevel":"Level2","listFormat":{},"characterFormat":{"fontSize":13,"f
ontFamily":"Calibri
Light","fontColor":"#2F5496"},"basedOn":"Normal","link":"Heading 2
Char","next":"Normal"},{"name":"Heading
3","type":"Paragraph","paragraphFormat":{"leftIndent":0,"rightIndent":0,"fir
stLineIndent":0,"textAlignment":"Left","beforeSpacing":2,"afterSpacing":0,"l
ineSpacing":1.0791666507720947,"lineSpacingType":"Multiple","outlineLevel":"
Level3","listFormat":{},"characterFormat":{"fontSize":12,"fontFamily":"Cali
bri Light","fontColor":"#1F3763"},"basedOn":"Normal","link":"Heading 3
Char","next":"Normal"},{"name":"Heading 3
Char","type":"Character","characterFormat":{"fontSize":12,"fontFamily":"Cali
bri Light","fontColor":"#1F3763"},"basedOn":"Default Paragraph
Font"},{"name":"Heading
4","type":"Paragraph","paragraphFormat":{"leftIndent":0,"rightIndent":0,"fir
stLineIndent":0,"textAlignment":"Left","beforeSpacing":2,"afterSpacing":0,"l
ineSpacing":1.0791666507720947,"lineSpacingType":"Multiple","outlineLevel":"
Level4","listFormat":{},"characterFormat":{"italic":true,"fontFamily":"Cali
bri Light","fontColor":"#2F5496"},"basedOn":"Normal","link":"Heading 4
Char","next":"Normal"},{"name":"Heading 4
Char","type":"Character","characterFormat":{"italic":true,"fontFamily":"Cali
bri Light","fontColor":"#2F5496"},"basedOn":"Default Paragraph
Font"},{"name":"Heading
5","type":"Paragraph","paragraphFormat":{"leftIndent":0,"rightIndent":0,"fir
stLineIndent":0,"textAlignment":"Left","beforeSpacing":2,"afterSpacing":0,"l
ineSpacing":1.0791666507720947,"lineSpacingType":"Multiple","outlineLevel":"
Level5","listFormat":{},"characterFormat":{"fontFamily":"Calibri
Light","fontColor":"#2F5496"},"basedOn":"Normal","link":"Heading 5
```

```

Char", "next": "Normal"}, {"name": "Heading 5
Char", "type": "Character", "characterFormat": {"fontFamily": "Calibri
Light", "fontColor": "#2F5496"}, "basedOn": "Default Paragraph
Font"}, {"name": "Heading
6", "type": "Paragraph", "paragraphFormat": {"leftIndent": 0, "rightIndent": 0, "fir
stLineIndent": 0, "textAlignment": "Left", "beforeSpacing": 2, "afterSpacing": 0, "l
ineSpacing": 1.0791666507720947, "lineSpacingType": "Multiple", "outlineLevel": "
Level6", "listFormat": {}}, "characterFormat": {"fontFamily": "Calibri
Light", "fontColor": "#1F3763"}, "basedOn": "Normal", "link": "Heading 6
Char", "next": "Normal"}, {"name": "Heading 6
Char", "type": "Character", "characterFormat": {"fontFamily": "Calibri
Light", "fontColor": "#1F3763"}, "basedOn": "Default Paragraph
Font"}], "lists": [], "abstractLists": [], "comments": [], "revisions": [], "customXm
l": []}`;

    // open the default document.
    (this.documentEditor as DocumentEditorComponent).open(sfdd);
  }
}

```

MAIN.TS

```

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

```

Opening a default document in DocumentEditorContainer

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DocumentEditorContainerModule } from '@syncfusion/ej2-angular-
documenteditor'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import { ToolbarService, DocumentEditorContainerComponent } from
'@syncfusion/ej2-angular-documenteditor';
/**
 * Document Editor Component
 */
@Component({
  imports: [

    DocumentEditorContainerModule
  ],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-documenteditorcontainer #documenteditor_default
height="600px" style="width:100%;display:block" [enableToolbar]=true
(created)="onCreate()"></ejs-documenteditorcontainer>`,
  encapsulation: ViewEncapsulation.None,
  providers: [ToolbarService]
})
export class AppComponent {
  @ViewChild('documenteditor_default')
  public container?: DocumentEditorContainerComponent;
}

```

```

// load your default document here
onCreate(): any {
    let sfdt: string =
`{"sections":[{"sectionFormat":{"pageWidth":612,"pageHeight":792,"leftMargin":72,"rightMargin":72,"topMargin":72,"bottomMargin":72,"differentFirstPage":false,"differentOddAndEvenPages":false,"headerDistance":36,"footerDistance":36,"bidi":false},"blocks":[{"paragraphFormat":{"afterSpacing":30,"styleName":"Heading 1","listFormat":{},"characterFormat":{},"inlines":[{"characterFormat":{},"text":"Adventure Works Cycles"}]}],"headersFooters":{"header":{"blocks":[{"paragraphFormat":{"listFormat":{},"characterFormat":{},"inlines":[]}}],"footer":{"blocks":[{"paragraphFormat":{"listFormat":{},"characterFormat":{},"inlines":[]}}]},"characterFormat":{"bold":false,"italic":false,"fontSize":11,"fontFamily":"Calibri","underline":"None","strikethrough":"None","baselineAlignment":"Normal","highlightColor":"NoColor","fontColor":"empty","fontSizeBidi":11,"fontFamilyBidi":"Calibri","allCaps":false},"paragraphFormat":{"leftIndent":0,"rightIndent":0,"firstLineIndent":0,"textAlignment":"Left","beforeSpacing":0,"afterSpacing":0,"lineSpacing":1.0791666507720947,"lineSpacingType":"Multiple","listFormat":{},"bidi":false},"defaultTabWidth":36,"trackChanges":false,"enforcement":false,"hashValue":"","saltValue":"","formatting":false,"protectionType":"NoProtection","dontUseHTMLParagraphAutoSpacing":false,"formFieldShading":true,"styles":[{"name":"Normal","type":"Paragraph","paragraphFormat":{"lineSpacing":1.149999976158142,"lineSpacingType":"Multiple","listFormat":{},"characterFormat":{"fontFamily":"Calibri"},"next":"Normal"}, {"name":"Default Paragraph Font","type":"Character","characterFormat":{}}, {"name":"Heading 1 Char","type":"Character","characterFormat":{"fontSize":16,"fontFamily":"Calibri Light","fontColor":"#2F5496"},"basedOn":"Default Paragraph Font"}, {"name":"Heading 1","type":"Paragraph","paragraphFormat":{"beforeSpacing":12,"afterSpacing":0,"outlineLevel":"Level1","listFormat":{},"characterFormat":{"fontSize":16,"fontFamily":"Calibri Light","fontColor":"#2F5496"},"basedOn":"Normal","link":"Heading 1 Char","next":"Normal"}, {"name":"Heading 2 Char","type":"Character","characterFormat":{"fontSize":13,"fontFamily":"Calibri Light","fontColor":"#2F5496"},"basedOn":"Default Paragraph Font"}, {"name":"Heading 2","type":"Paragraph","paragraphFormat":{"beforeSpacing":2,"afterSpacing":6,"outlineLevel":"Level2","listFormat":{},"characterFormat":{"fontSize":13,"fontFamily":"Calibri Light","fontColor":"#2F5496"},"basedOn":"Normal","link":"Heading 2 Char","next":"Normal"}, {"name":"Heading 3","type":"Paragraph","paragraphFormat":{"leftIndent":0,"rightIndent":0,"firstLineIndent":0,"textAlignment":"Left","beforeSpacing":2,"afterSpacing":0,"lineSpacing":1.0791666507720947,"lineSpacingType":"Multiple","outlineLevel":"Level3","listFormat":{},"characterFormat":{"fontSize":12,"fontFamily":"Calibri Light","fontColor":"#1F3763"},"basedOn":"Normal","link":"Heading 3 Char","next":"Normal"}, {"name":"Heading 3 Char","type":"Character","characterFormat":{"fontSize":12,"fontFamily":"Calibri Light","fontColor":"#1F3763"},"basedOn":"Default Paragraph Font"}, {"name":"Heading 4","type":"Paragraph","paragraphFormat":{"leftIndent":0,"rightIndent":0,"firstLineIndent":0,"textAlignment":"Left","beforeSpacing":2,"afterSpacing":0,"lineSpacing":1.0791666507720947,"lineSpacingType":"Multiple","outlineLevel":"Level4","listFormat":{},"characterFormat":{"italic":true,"fontFamily":"Calibri Light","fontColor":"#2F5496"},"basedOn":"Normal","link":"Heading 4 Char","next":"Normal"}, {"name":"Heading 4

```

```

Char", "type": "Character", "characterFormat": { "italic": true, "fontFamily": "Calibri Light", "fontColor": "#2F5496" }, "basedOn": "Default Paragraph Font" }, { "name": "Heading 5", "type": "Paragraph", "paragraphFormat": { "leftIndent": 0, "rightIndent": 0, "firstLineIndent": 0, "textAlignment": "Left", "beforeSpacing": 2, "afterSpacing": 0, "lineSpacing": 1.0791666507720947, "lineSpacingType": "Multiple", "outlineLevel": "Level5", "listFormat": {} }, "characterFormat": { "fontFamily": "Calibri Light", "fontColor": "#2F5496" }, "basedOn": "Normal", "link": "Heading 5 Char", "next": "Normal" }, { "name": "Heading 5 Char", "type": "Character", "characterFormat": { "fontFamily": "Calibri Light", "fontColor": "#2F5496" }, "basedOn": "Default Paragraph Font" }, { "name": "Heading 6", "type": "Paragraph", "paragraphFormat": { "leftIndent": 0, "rightIndent": 0, "firstLineIndent": 0, "textAlignment": "Left", "beforeSpacing": 2, "afterSpacing": 0, "lineSpacing": 1.0791666507720947, "lineSpacingType": "Multiple", "outlineLevel": "Level6", "listFormat": {} }, "characterFormat": { "fontFamily": "Calibri Light", "fontColor": "#1F3763" }, "basedOn": "Normal", "link": "Heading 6 Char", "next": "Normal" }, { "name": "Heading 6 Char", "type": "Character", "characterFormat": { "fontFamily": "Calibri Light", "fontColor": "#1F3763" }, "basedOn": "Default Paragraph Font" } ], "lists": [], "abstractLists": [], "comments": [], "revisions": [], "customXml1": [] } `;

    // open the default document.
    (this.container as DocumentEditorContainerComponent)
    ).documentEditor.open(sfddt);
  }
}

```

MAIN.TS

```

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

```

Default read only in Angular Document editor component

In this article, we are going to see how to open a document in read only mode by default in Document Editor & Document Editor Container.

Opening a document in read only mode by default in DocumentEditor

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { DocumentEditorAllModule } from '@syncfusion/ej2-angular-documenteditor';
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import { DocumentEditorComponent, EditorService, SelectionService, EditorHistoryService, BookmarkDialogService } from '@syncfusion/ej2-angular-documenteditor';
@Component({
  imports: [

    DocumentEditorAllModule
  ],

```



```

standalone: true,
  selector: "app-container",
  template: `<ejs-documenteditor #documenteditor_readonly height="330px"
style="width:100%;display:block" (created)="onCreate()"
(documentChange)="onDocumentChange()" [isReadOnly]=false
[enableEditor]=true ></ejs-documenteditor>`,
  encapsulation: ViewEncapsulation.None,
  providers: [EditorService, SelectionService, EditorHistoryService,
BookmarkDialogService]
})
export class AppComponent {
  @ViewChild("documenteditor_readonly")
  public documentEditor?: DocumentEditorComponent;
  // load your default document here
  onCreate() {
    let sfdt: string =
`{"sections":[{"sectionFormat":{"pageWidth":612,"pageHeight":792,"leftMargin":72,"rightMargin":72,"topMargin":72,"bottomMargin":72,"differentFirstPage":false,"differentOddAndEvenPages":false,"headerDistance":36,"footerDistance":36,"bidi":false},"blocks":[{"paragraphFormat":{"afterSpacing":30,"styleName":"Heading 1","listFormat":{},"characterFormat":{},"inlines":[{"characterFormat":{},"text":"Adventure Works Cycles"}]}]}],"headersFooters":{"header":{"blocks":[{"paragraphFormat":{"listFormat":{},"characterFormat":{},"inlines":[]}}],"footer":{"blocks":[{"paragraphFormat":{"listFormat":{},"characterFormat":{},"inlines":[]}}]}],"characterFormat":{"bold":false,"italic":false,"fontSize":11,"fontFamily":"Calibri","underline":"None","strikethrough":"None","baselineAlignment":"Normal","highlightColor":"NoColor","fontColor":"empty","fontSizeBidi":11,"fontFamilyBidi":"Calibri","allCaps":false},"paragraphFormat":{"leftIndent":0,"rightIndent":0,"firstLineIndent":0,"textAlignment":"Left","beforeSpacing":0,"afterSpacing":0,"lineSpacing":1.0791666507720947,"lineSpacingType":"Multiple","listFormat":{},"bidi":false},"defaultTabWidth":36,"trackChanges":false,"enforcement":false,"hashValue":"","saltValue":"","formatting":false,"protectionType":"NoProtection","dontUseHTMLParagraphAutoSpacing":false,"formFieldShading":true,"styles":[{"name":"Normal","type":"Paragraph","paragraphFormat":{"lineSpacing":1.149999976158142,"lineSpacingType":"Multiple","listFormat":{},"characterFormat":{"fontFamily":"Calibri"},"next":"Normal"},{"name":"Default Paragraph Font","type":"Character","characterFormat":{}},{name":"Heading 1 Char","type":"Character","characterFormat":{"fontSize":16,"fontFamily":"Calibri Light","fontColor":"#2F5496"},"basedOn":"Default Paragraph Font"},{"name":"Heading 1","type":"Paragraph","paragraphFormat":{"beforeSpacing":12,"afterSpacing":0,"outlineLevel":"Level1","listFormat":{},"characterFormat":{"fontSize":16,"fontFamily":"Calibri Light","fontColor":"#2F5496"},"basedOn":"Normal","link":"Heading 1 Char","next":"Normal"},{"name":"Heading 2 Char","type":"Character","characterFormat":{"fontSize":13,"fontFamily":"Calibri Light","fontColor":"#2F5496"},"basedOn":"Default Paragraph Font"},{"name":"Heading 2","type":"Paragraph","paragraphFormat":{"beforeSpacing":2,"afterSpacing":6,"outlineLevel":"Level2","listFormat":{},"characterFormat":{"fontSize":13,"fontFamily":"Calibri Light","fontColor":"#2F5496"},"basedOn":"Normal","link":"Heading 2 Char","next":"Normal"},{"name":"Heading 3","type":"Paragraph","paragraphFormat":{"leftIndent":0,"rightIndent":0,"firstLineIndent":0,"textAlignment":"Left","beforeSpacing":2,"afterSpacing":0,"l

```

```

    "lineSpacing":1.0791666507720947,"lineSpacingType":"Multiple","outlineLevel":"
    Level3","listFormat":{},"characterFormat":{"fontSize":12,"fontFamily":"Cali
    bri Light","fontColor":"#1F3763"},"basedOn":"Normal","link":"Heading 3
    Char","next":"Normal"}, {"name":"Heading 3
    Char","type":"Character","characterFormat":{"fontSize":12,"fontFamily":"Cali
    bri Light","fontColor":"#1F3763"},"basedOn":"Default Paragraph
    Font"}, {"name":"Heading
    4","type":"Paragraph","paragraphFormat":{"leftIndent":0,"rightIndent":0,"fir
    stLineIndent":0,"textAlignment":"Left","beforeSpacing":2,"afterSpacing":0,"l
    ineSpacing":1.0791666507720947,"lineSpacingType":"Multiple","outlineLevel":"
    Level4","listFormat":{},"characterFormat":{"italic":true,"fontFamily":"Cali
    bri Light","fontColor":"#2F5496"},"basedOn":"Normal","link":"Heading 4
    Char","next":"Normal"}, {"name":"Heading 4
    Char","type":"Character","characterFormat":{"italic":true,"fontFamily":"Cali
    bri Light","fontColor":"#2F5496"},"basedOn":"Default Paragraph
    Font"}, {"name":"Heading
    5","type":"Paragraph","paragraphFormat":{"leftIndent":0,"rightIndent":0,"fir
    stLineIndent":0,"textAlignment":"Left","beforeSpacing":2,"afterSpacing":0,"l
    ineSpacing":1.0791666507720947,"lineSpacingType":"Multiple","outlineLevel":"
    Level5","listFormat":{},"characterFormat":{"fontFamily":"Calibri
    Light","fontColor":"#2F5496"},"basedOn":"Normal","link":"Heading 5
    Char","next":"Normal"}, {"name":"Heading 5
    Char","type":"Character","characterFormat":{"fontFamily":"Calibri
    Light","fontColor":"#2F5496"},"basedOn":"Default Paragraph
    Font"}, {"name":"Heading
    6","type":"Paragraph","paragraphFormat":{"leftIndent":0,"rightIndent":0,"fir
    stLineIndent":0,"textAlignment":"Left","beforeSpacing":2,"afterSpacing":0,"l
    ineSpacing":1.0791666507720947,"lineSpacingType":"Multiple","outlineLevel":"
    Level6","listFormat":{},"characterFormat":{"fontFamily":"Calibri
    Light","fontColor":"#1F3763"},"basedOn":"Normal","link":"Heading 6
    Char","next":"Normal"}, {"name":"Heading 6
    Char","type":"Character","characterFormat":{"fontFamily":"Calibri
    Light","fontColor":"#1F3763"},"basedOn":"Default Paragraph
    Font"}], "lists":[], "abstractLists":[], "comments":[], "revisions":[], "customXm
    l":[]}`;

    // open the default document.
    (this.documentEditor as DocumentEditorComponent).open(sfdd);
  }
  onDocumentChange() {
    //Enable read only mode.
    (this.documentEditor as DocumentEditorComponent).isReadOnly = true;
  }
}

```

MAIN.TS

```

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

```

Opening a document in ready only mode by default in DocumentEditorContainer

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'

```

```

import { BrowserModule } from '@angular/platform-browser'
import { DocumentEditorContainerModule } from '@syncfusion/ej2-angular-documenteditor'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import { ToolbarService, DocumentEditorContainerComponent } from '@syncfusion/ej2-angular-documenteditor';
/**
 * Document Editor Component
 */
@Component({
  imports: [

    DocumentEditorContainerModule
  ],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-documenteditorcontainer #documenteditor_readonly
height="600px" style="width:100%;display:block" [enableToolbar]=true
(created)="onCreate()" (documentChange)="onDocumentChange()" ></ejs-
documenteditorcontainer>`,
  encapsulation: ViewEncapsulation.None,
  providers: [ToolbarService]
})
export class AppComponent {
  @ViewChild('documenteditor_readonly')
  public container?: DocumentEditorContainerComponent;
  // load your default document here
  onCreate(): any {
    let sfdt: string =
`{"sections":[{"sectionFormat":{"pageWidth":612,"pageHeight":792,"leftMargin":72,"rightMargin":72,"topMargin":72,"bottomMargin":72,"differentFirstPage":false,"differentOddAndEvenPages":false,"headerDistance":36,"footerDistance":36,"bidi":false},"blocks":[{"paragraphFormat":{"afterSpacing":30,"styleName":"Heading 1","listFormat":{},"characterFormat":{},"inlines":[{"characterFormat":{},"text":"Adventure Works Cycles"}]}]},"headersFooters":{"header":{"blocks":[{"paragraphFormat":{"listFormat":{},"characterFormat":{},"inlines":[]}}],"footer":{"blocks":[{"paragraphFormat":{"listFormat":{},"characterFormat":{},"inlines":[]}}]},"characterFormat":{"bold":false,"italic":false,"fontSize":11,"fontFamily":"Calibri","underline":"None","strikethrough":"None","baselineAlignment":"Normal","highlightColor":"NoColor","fontColor":"empty","fontSizeBidi":11,"fontFamilyBidi":"Calibri","allCaps":false},"paragraphFormat":{"leftIndent":0,"rightIndent":0,"firstLineIndent":0,"textAlignment":"Left","beforeSpacing":0,"afterSpacing":0,"lineSpacing":1.0791666507720947,"lineSpacingType":"Multiple","listFormat":{},"bidi":false},"defaultTabWidth":36,"trackChanges":false,"enforcement":false,"hashValue":"","saltValue":"","formatting":false,"protectionType":"No Protection","dontUseHTMLParagraphAutoSpacing":false,"formFieldShading":true,"styles":[{"name":"Normal","type":"Paragraph","paragraphFormat":{"lineSpacing":1.149999976158142,"lineSpacingType":"Multiple","listFormat":{},"characterFormat":{"fontFamily":"Calibri"},"next":"Normal"},{"name":"Default Paragraph Font","type":"Character","characterFormat":{}},{name":"Heading 1 Char","type":"Character","characterFormat":{"fontSize":16,"fontFamily":"Calibri Light","fontColor":"#2F5496"},"basedOn":"Default Paragraph Font"},{"name":"Heading 1","type":"Paragraph","paragraphFormat":{"beforeSpacing":12,"afterSpacing":0,"outlineLevel":"Level1","listFormat":{},"characterFormat":{"fontSize":16,"`

```

```

fontFamily":"Calibri
Light","fontColor":"#2F5496"},"basedOn":"Normal","link":"Heading 1
Char","next":"Normal"},"{"name":"Heading 2
Char","type":"Character","characterFormat":{"fontSize":13,"fontFamily":"Cali
bri Light","fontColor":"#2F5496"},"basedOn":"Default Paragraph
Font"},"{"name":"Heading
2","type":"Paragraph","paragraphFormat":{"beforeSpacing":2,"afterSpacing":6,
"outlineLevel":"Level2","listFormat":{}},"characterFormat":{"fontSize":13,"f
ontFamily":"Calibri
Light","fontColor":"#2F5496"},"basedOn":"Normal","link":"Heading 2
Char","next":"Normal"},"{"name":"Heading
3","type":"Paragraph","paragraphFormat":{"leftIndent":0,"rightIndent":0,"fir
stLineIndent":0,"textAlignment":"Left","beforeSpacing":2,"afterSpacing":0,"l
ineSpacing":1.0791666507720947,"lineSpacingType":"Multiple","outlineLevel":"
Level3","listFormat":{}},"characterFormat":{"fontSize":12,"fontFamily":"Cali
bri Light","fontColor":"#1F3763"},"basedOn":"Normal","link":"Heading 3
Char","next":"Normal"},"{"name":"Heading 3
Char","type":"Character","characterFormat":{"fontSize":12,"fontFamily":"Cali
bri Light","fontColor":"#1F3763"},"basedOn":"Default Paragraph
Font"},"{"name":"Heading
4","type":"Paragraph","paragraphFormat":{"leftIndent":0,"rightIndent":0,"fir
stLineIndent":0,"textAlignment":"Left","beforeSpacing":2,"afterSpacing":0,"l
ineSpacing":1.0791666507720947,"lineSpacingType":"Multiple","outlineLevel":"
Level4","listFormat":{}},"characterFormat":{"italic":true,"fontFamily":"Cali
bri Light","fontColor":"#2F5496"},"basedOn":"Normal","link":"Heading 4
Char","next":"Normal"},"{"name":"Heading 4
Char","type":"Character","characterFormat":{"italic":true,"fontFamily":"Cali
bri Light","fontColor":"#2F5496"},"basedOn":"Default Paragraph
Font"},"{"name":"Heading
5","type":"Paragraph","paragraphFormat":{"leftIndent":0,"rightIndent":0,"fir
stLineIndent":0,"textAlignment":"Left","beforeSpacing":2,"afterSpacing":0,"l
ineSpacing":1.0791666507720947,"lineSpacingType":"Multiple","outlineLevel":"
Level5","listFormat":{}},"characterFormat":{"fontFamily":"Calibri
Light","fontColor":"#2F5496"},"basedOn":"Normal","link":"Heading 5
Char","next":"Normal"},"{"name":"Heading 5
Char","type":"Character","characterFormat":{"fontFamily":"Calibri
Light","fontColor":"#2F5496"},"basedOn":"Default Paragraph
Font"},"{"name":"Heading
6","type":"Paragraph","paragraphFormat":{"leftIndent":0,"rightIndent":0,"fir
stLineIndent":0,"textAlignment":"Left","beforeSpacing":2,"afterSpacing":0,"l
ineSpacing":1.0791666507720947,"lineSpacingType":"Multiple","outlineLevel":"
Level6","listFormat":{}},"characterFormat":{"fontFamily":"Calibri
Light","fontColor":"#1F3763"},"basedOn":"Normal","link":"Heading 6
Char","next":"Normal"},"{"name":"Heading 6
Char","type":"Character","characterFormat":{"fontFamily":"Calibri
Light","fontColor":"#1F3763"},"basedOn":"Default Paragraph
Font"}],"lists":[],"abstractLists":[],"comments":[],"revisions":[],"customXm
l":[]}`;

// open the default document.
(this.container as DocumentEditorContainerComponent
).documentEditor.open(sfddt);
}
onDocumentChange(): void {
    //Enable read only mode.
    (this.container as DocumentEditorContainerComponent).restrictEditing
= true;
}

```

```
}

```

MAIN.TS

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

Note: You can use the `restrictEditing` in `DocumentEditorContainer` and `isReadOnly` in `DocumentEditor` based on your requirement to change component to read only mode.

Open document by address in Angular Document editor component

[How to open a document from URL in Angular Document Editor](#)

In this article, we are going to see how to open a document from URL in DocumentEditor

please refer below example for client-side code

```
`typescript
```

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

```
import {
```

```
DocumentEditorContainerComponent,
```

```
ToolbarService,
```

```
} from '@syncfusion/ej2-angular-documenteditor';
```

```
@Component({
```

```
selector: 'app-root',
```

```
// specifies the template string for the DocumentEditorContainer component
```

```
template: <button id='export'(click)="onClick(this)">Export</button><ejs-  
documenteditorcontainer #documenteditor_default  
serviceUrl="https://ej2services.syncfusion.com/production/web-  
services/api/documenteditor/" height="600px" style="display:block" [enableToolbar]=true>  
</ejs-documenteditorcontainer>,
```

```
providers: [ToolbarService],
```

```
})
```

```
export class AppComponent implements OnInit {
```

```
@ViewChild('documenteditor_default')
```

```
public container: DocumentEditorContainerComponent;
```

```
ngOnInit(): void {}
```

```
onClick(): void {
```

```
let http: XMLHttpRequest = new XMLHttpRequest();
```

```
//add your url in which you want to open document inside the ""
```

```
let content = { fileUrl: " "};
let baseUrl: string = '/api/documenteditor/ImportFileURL';
http.open('POST', baseUrl, true);
http.setRequestHeader('Content-Type', 'application/json;charset=UTF-8');
http.onreadystatechange = () => {
if (http.readyState === 4) {
if (http.status === 200 || http.status === 304) {
//open the SFDT text in Document Editor
this.container.documentEditor.open(http.responseText);
}
}
};
http.send(JSON.stringify(content));
}
}
`
```

please refer below example for server-side code

```
`csharp
[AcceptVerbs("Post")]
public string ImportFileURL([FromBody]FileUrlInfo param)
{
try {
using(WebClient client = new WebClient())
{
MemoryStream stream = new MemoryStream(client.DownloadData(param.fileUrl));
WordDocument document = WordDocument.Load(stream, FormatType.Docx);
string json = Newtonsoft.Json.JsonConvert.SerializeObject(document);
document.Dispose();
stream.Dispose();
return json;
}
}
catch (Exception) {
```

```

return "";
}
}

public class FileUrlInfo {
public string fileUrl { get; set; }
public string Content { get; set; }
}
`

```

Deploy document editor component for mobile in Angular Document editor component *Document editor component for Mobile*

At present, Document editor component is not responsive for mobile, and we haven't ensured the editing functionalities in mobile browsers. Whereas it works properly as a document viewer in mobile browsers.

Hence, it is recommended to switch the Document editor component as read-only in mobile browsers. Also, invoke [fitPage](#) method with [FitPageWidth](#) parameter in document change event, such as to display one full page by adjusting the zoom factor.

The following example code illustrates how to deploy Document Editor component for Mobile.

```

`typescript
//Initialize Document Editor Container component.
import { Component } from '@angular/core';
import { ToolbarService } from '@syncfusion/ej2-angular-documentededitor';
@Component({
selector: 'app-container',
// specifies the template string for the DocumentEditorContainer component
template: <ejs-documenteditorcontainer #document_editor
(documentChange)="onDocumentChange()"
serviceUrl="https://ej2services.syncfusion.com/production/web-
services/api/documenteditor/" height="600px" style="display:block" [enableToolbar]=true>
</ejs-documenteditorcontainer>,
providers: [ToolbarService]
})
export class AppComponent {
@ViewChild('document_editor')
public container: DocumentEditorContainerComponent;
onDocumentChange(): void {
//To detect the device

```

```

let isMobileDevice: boolean = /Android|Windows Phone|webOS/i.test(navigator.userAgent);
if (isMobileDevice) {
  this.container.restrictEditing = true;
  setTimeout(() => {
    this.container.documentEditor.fitPage("FitPageWidth");
  }, 50);
}
else {
  this.container.restrictEditing = false;
}
}
}
`

```

You can download the complete working example from this [GitHub location](#)

Note: You can use the [restrictEditing](#) in DocumentEditorContainer and [isReadOnly](#) in DocumentEditor based on your requirement to change component to read only mode.

Disable optimized text measuring in Angular Document editor component

Starting from v19.3.0.x, the accuracy of text size measurements in Document editor is improved such as to match Microsoft Word pagination for most Word documents. This improvement is included as default behavior along with an optional API [enableOptimizedTextMeasuring](#) in Document editor settings.

If you want the Document editor component to retain the document pagination (display page-by-page) behavior like v19.2.0.x and older versions. Then you can disable this optimized text measuring improvement, by setting `false` to [enableOptimizedTextMeasuring](#) property of Angular Document Editor component.

Disable optimized text measuring in `DocumentEditorContainer` instance

The following example code illustrates how to disable optimized text measuring improvement in `DocumentEditorContainer` instance.

```

`typescript
import { Component, ViewEncapsulation } from '@angular/core';
import {
  DocumentEditorComponent, PrintService, SfdtExportService, WordExportService, TextExportService,
  SelectionService,
  SearchService, EditorService, ImageResizerService, EditorHistoryService, ContextMenuService,
  OptionsPaneService, HyperlinkDialogService, TableDialogService, BookmarkDialogService,
  TableOfContentsDialogService,
  PageSetupDialogService, StyleDialogService, ListDialogService, ParagraphDialogService,
  BulletsAndNumberingDialogService,

```



```

FontDialogService, TablePropertiesDialogService, BordersAndShadingDialogService,
TableOptionsDialogService,
CellOptionsDialogService, StylesDialogService
} from '@syncfusion/ej2-angular-documenteditor';

@Component({
  selector: 'app-container',
  template: `<ejs-documenteditor id="container" [documentEditorSettings]= "settings"
  serviceUrl="https://ej2services.syncfusion.com/production/web-services/api/documenteditor/"
  height="330px" style="display:block" [isReadOnly]=false [enableSelection]=true
  [enablePrint]=true [enableSfdtExport]=true [enableWordExport]=true [enableOptionsPane]=true
  [enableContextMenu]=true
  [enableHyperlinkDialog]=true [enableBookmarkDialog]=true [enableTableOfContentsDialog]=true
  [enableSearch]=true
  [enableParagraphDialog]=true [enableListDialog]=true [enableTablePropertiesDialog]=true
  [enableBordersAndShadingDialog]=true
  [enablePageSetupDialog]=true [enableStyleDialog]=true [enableFontDialog]=true
  [enableTableOptionsDialog]=true
  [enableTableDialog]=true [enableImageResizer]=true [enableEditor]=true [enableEditorHistory]=true>
  </ejs-documenteditor>`,
  encapsulation: ViewEncapsulation.None,
  providers: [PrintService, SfdtExportService, WordExportService, TextExportService, SelectionService,
  SearchService, EditorService,
  ImageResizerService, EditorHistoryService, ContextMenuService, OptionsPaneService,
  HyperlinkDialogService, TableDialogService,
  BookmarkDialogService, TableOfContentsDialogService, PageSetupDialogService, StyleDialogService,
  ListDialogService,
  ParagraphDialogService, BulletsAndNumberingDialogService, FontDialogService,
  TablePropertiesDialogService,
  BordersAndShadingDialogService, TableOptionsDialogService, CellOptionsDialogService,
  StylesDialogService]
})

export class AppComponent {
  public settings = { enableOptimizedTextMeasuring: false };
}

```

Disable optimized text measuring in `DocumentEditor` instance

The following example code illustrates how to disable optimized text measuring improvement in `DocumentEditor` instance.

```
`typescript
import { Component, OnInit } from '@angular/core';
import { ToolbarService } from '@syncfusion/ej2-angular-documenteditor';
@Component({
  selector: 'app-container',
  // specifies the template string for the DocumentEditorContainer component
  template: <ejs-documenteditorcontainer [documentEditorSettings]= "settings"
  serviceUrl="https://ej2services.syncfusion.com/production/web-
  services/api/documenteditor/" height="600px" style="display:block" [enableToolbar]=true>
  </ejs-documenteditorcontainer>,
  providers: [ToolbarService]
})
export class AppComponent {
  public settings = { enableOptimizedTextMeasuring: false };
}
```

Get the selected content in Angular Document editor component

You can get the selected content from the Angular Document Editor component as plain text and SFDT (rich text).

Get the selected content as plain text

You can use [text](#) API to get the selected content as plain text from Angular Document Editor component.

The following example code illustrates how to add search in google option in context menu for the selected text.

```
`typescript
import { Component, OnInit, ViewChild } from '@angular/core';
import { DocumentEditorContainerComponent, ToolbarService } from '@syncfusion/ej2-angular-
documenteditor';
import { MenuItemModel } from '@syncfusion/ej2-navigations';
@Component({
  selector: 'app-root',
  // specifies the template string for the DocumentEditorContainer component
  template: <ejs-documenteditorcontainer #documenteditor_default
  serviceUrl="https://ej2services.syncfusion.com/production/web-
```

```
services/api/documenteditor/" height="600px" style="display:block" [enableToolbar]=true
(created)="onCreate()"> </ejs-documenteditorcontainer>,
providers: [ToolbarService]
})
export class AppComponent implements OnInit {
  @ViewChild('documenteditor_default')
  public container: DocumentEditorContainerComponent;
  ngOnInit(): void {
  }
  onCreate():void {
    // creating Custom Options
    let menuItems: MenuItemModel[] = [
    {
      text: 'Search In Google',
      id: 'searchingoogle',
      iconCss: 'e-icons e-de-ctnr-find'
    }
    ];
    // adding Custom Options
    this.container.documentEditor.contextMenu.addCustomMenu(menuItems, false);
    // custom Options Select Event
    this.container.documentEditor.customContextMenuSelect = (args: any): void => {
    // custom Options Functionality
    let id: string = this.container.documentEditor.element.id;
    switch (args.id) {
      case id + 'searchingoogle':
        // To get the selected content as plain text
        let searchContent: string =this.container.documentEditor.selection.text;
        if (!this.container.documentEditor.selection.isEmpty && /\S/.test(searchContent)) {
          window.open('http://google.com/search?q=' + searchContent);
        }
        break;
      }
    };
  }
```

```
}
}
`
```

You can add the following custom options using this API,

- Save or export the selected text as text file.
- Search the selected text in Google or other search engines.
- Show synonyms for the selected word in context menu and replace with selected synonym using the setter method of same API.

Get the selected content as SFDT (rich text)

You can use [sfdt](#) API to get the selected content as plain text from Angular Document Editor component.

The following example code illustrates how to get the content of a bookmark and export it as SFDT.

```
`typescript
import { Component, OnInit, ViewChild } from '@angular/core';
import {
  DocumentEditorContainerComponent,
  ToolbarService,
} from '@syncfusion/ej2-angular-documenteditor';
@Component({
  selector: 'app-root',
  // specifies the template string for the DocumentEditorContainer component
  template: <ejs-documenteditorcontainer #documenteditor_default
    serviceUrl="https://ej2services.syncfusion.com/production/web-
    services/api/documenteditor/" height="600px" style="display:block" [enableToolbar]=true
    (created)="onCreate()"> </ejs-documenteditorcontainer>,
  providers: [ToolbarService],
})
export class AppComponent implements OnInit {
  @ViewChild('documenteditor_default')
  public container: DocumentEditorContainerComponent;
  ngOnInit(): void {}
  onCreate(): void {
    // To insert text in cursor position
    this.container.documentEditor.editor.insertText('Document editor');
    // To select all the content in document
```

```

this.container.documentEditor.selection.selectAll();
// Insert bookmark to selected content
this.container.documentEditor.editor.insertBookmark('Bookmark1');
//Select the bookmark
this.container.documentEditor.selection.selectBookmark('Bookmark1');
// To get the selected content as sfdt
let selectedContent: string = this.container.documentEditor.selection.sfdt;
// Insert the sfdt content in cursor position using paste API
this.container.documentEditor.editor.paste(selectedContent);
}
}
`

```

You can add the following custom options using this API,

- Save or export the selected content as SFDT file.
- Get the content of a bookmark in Word document as SFDT by selecting a bookmark using [selectbookmark](#) API.
- Create template content that can be inserted to multiple documents in cursor position using [paste](#) API.

Set default format in document editor in Angular Document editor component

You can set the default character format, paragraph format and section format in Document editor.

Set the default character format

You can use [setDefaultCharacterFormat](#) method to set the default character format. For example, Document editor default font size is 11 and you can change it as any valid value.

The following example code illustrates how to change the default font size in Document editor.

```

`typescript
import { Component, OnInit, ViewChild } from '@angular/core';
import { ToolbarService, DocumentEditorContainerComponent } from '@syncfusion/ej2-angular-documenteditor';
@Component({
  selector: 'app-root',
  // specifies the template string for the DocumentEditorContainer component
  template: <ejs-documenteditorcontainer #documenteditor_default
serviceUrl="https://ej2services.syncfusion.com/production/web-
services/api/documenteditor/" height="600px" style="display:block"
[documentEditorSettings]= "fontFamilies" [enableToolbar]=true (created)="onCreate()"> </ejs-
documenteditorcontainer>,

```

```

providers: [ToolbarService]
})
export class AppComponent implements OnInit {
  @ViewChild('documenteditor_default')
  public container: DocumentEditorContainerComponent;
  public fontFamilies={fontFamilies :['Algerian', 'Arial', 'Calibri', 'Cambria', 'Windings']};
  ngOnInit(): void {
  }
  onCreate() {
    this.container.documentEditor.setDefaultCharacterFormat({fontSize: 20});
  }
}

```

Similarly, you can change the required [CharacterFormatProperties](#) default value.

The following example code illustrates how to change other character format default value in Document editor.

```

`typescript
import { Component, OnInit,ViewChild } from '@angular/core';
import { ToolbarService ,DocumentEditorContainerComponent, CharacterFormatProperties} from
 '@syncfusion/ej2-angular-documenteditor';
@Component({
  selector: 'app-root',
  // specifies the template string for the DocumentEditorContainer component
  template: <ejs-documenteditorcontainer #documenteditor_default
  serviceUrl="https://ej2services.syncfusion.com/production/web-
  services/api/documenteditor/" height="600px" style="display:block"
  [documentEditorSettings]= "fontFamilies" [enableToolbar]=true (created)="onCreate()"> </ejs-
  documenteditorcontainer>,
  providers: [ToolbarService]
})
export class AppComponent implements OnInit {
  @ViewChild('documenteditor_default')
  public container: DocumentEditorContainerComponent;
  public fontFamilies={fontFamilies :['Algerian', 'Arial', 'Calibri', 'Cambria', 'Windings']};
  ngOnInit(): void {

```

```

}
onCreate() {
let defaultCharacterFormat:CharacterFormatProperties = {
bold: false,
italic: false,
baselineAlignment: 'Normal',
underline: 'None',
fontColor: "#000000",
fontFamily: 'Algerian',
fontSize: 12
};
this.container.documentEditor.setDefaultCharacterFormat(defaultCharacterFormat);
}
}
,

```

Set the default paragraph format

You can use [setDefaultParagraphFormat](#) API to set the default paragraph format. You can change the required [ParagraphFormatProperties](#) default value.

The following example code illustrates how to change the paragraph format(before spacing, line spacing etc.,) default value in Document editor.

```

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

import { ToolbarService ,DocumentEditorContainerComponent,ParagraphFormatProperties} from
'@syncfusion/ej2-angular-documenteditor';

@Component({
selector: 'app-root',
// specifies the template string for the DocumentEditorContainer component
template: <ejs-documenteditorcontainer #documenteditor_default
serviceUrl="https://ej2services.syncfusion.com/production/web-
services/api/documenteditor/" height="600px" style="display:block"
[documentEditorSettings]= "fontFamilies" [enableToolbar]=true (created)="onCreate()"> </ejs-
documenteditorcontainer>,
providers: [ToolbarService]
})

export class AppComponent implements OnInit {

```

```

@ViewChild('documenteditor_default')
public container: DocumentEditorContainerComponent;
public fontFamilies={fontFamilies:['Algerian', 'Arial', 'Calibri', 'Cambria', 'Windings']};
ngOnInit(): void {
}
onCreate() {
let defaultParagraphFormat:ParagraphFormatProperties = {
beforeSpacing: 8,
lineSpacing: 1.5,
leftIndent: 24,
textAlignment: "Center"
};
this.container.documentEditor.setDefaultParagraphFormat(defaultParagraphFormat);
}
}
`

```

Set the default section format

You can use [setDefaultSectionFormat](#) API to set the default section format. You can change the required [SectionFormatProperties](#) default value.

The following example code illustrates how to change the section format(header and footer distance, page width and height, etc.,) default value in Document editor.

```

`typescript
import { Component, OnInit,ViewChild } from '@angular/core';
import { ToolbarService ,DocumentEditorContainerComponent,SectionFormatProperties} from
 '@syncfusion/ej2-angular-documenteditor';
@Component({
selector: 'app-root',
// specifies the template string for the DocumentEditorContainer component
template: <ejs-documenteditorcontainer #documenteditor_default
serviceUrl="https://ej2services.syncfusion.com/production/web-
services/api/documenteditor/" height="600px" style="display:block"
[documentEditorSettings]= "fontFamilies" [enableToolbar]=true (created)="onCreate()"> </ejs-
documenteditorcontainer>,
providers: [ToolbarService]
})
`

```



```
export class AppComponent implements OnInit {
  @ViewChild('documenteditor_default')
  public container: DocumentEditorContainerComponent;
  public fontFamilies={fontFamilies:['Algerian', 'Arial', 'Calibri', 'Cambria', 'Windings']};
  ngOnInit(): void {
  }
  onCreate() {
    let defaultSectionFormat:SectionFormatProperties = {
      pageWidth: 500,
      pageHeight: 800,
      headerDistance: 56,
      footerDistance: 48,
      leftMargin: 12,
      rightMargin: 12,
      topMargin:0,
      bottomMargin:0
    };
    this.container.documentEditor.setDefaultSectionFormat(defaultSectionFormat);
  }
}
```

Show hide spinner in Angular Document editor component

Using [spinner](#) component, you can show/hide spinner while opening document in DocumentEditor .

Example code snippet to show/hide spinner

```
`typescript
// showSpinner() will make the spinner visible
showSpinner(document.getElementById('container'));
// hideSpinner() method used hide spinner
hideSpinner(document.getElementById('container'));
`
```

Refer to the following example.

APP.COMPONENT.TS

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

```

import { DocumentEditorContainerModule } from '@syncfusion/ej2-angular-
documenteditor'
import { Component, OnInit, ViewChild } from '@angular/core';
import { DocumentEditorContainerComponent, ImageFormat, ToolbarService }
from '@syncfusion/ej2-angular-documenteditor';
import { showSpinner,hideSpinner,createSpinner} from '@syncfusion/ej2-
popups';
@Component({
imports: [

    DocumentEditorContainerModule
],
standalone: true,
    selector: 'app-container',
    // specifies the template string for the DocumentEditorContainer
component
    template: `<button
id='export' (click)="onClick(this)">Export</button><ejs-
documenteditorcontainer id="container" #documenteditor_default
serviceUrl="https://ej2services.syncfusion.com/production/web-
services/api/documenteditor/" height="600px" style="display:block"
[enableToolbar]=true> </ejs-documenteditorcontainer>`,
    providers: [ToolbarService]
})
export class AppComponent implements OnInit {
    @ViewChild('documenteditor_default')
    public container?: DocumentEditorContainerComponent;
    ngOnInit(): void {
        createSpinner({
            // Specify the target for the spinner to show
            target: (document.getElementById('container') as HTMLElement)
as HTMLElement
        });
    }
    onClick( args: any):void {
        // load your default document here
        let data: string=
'{"sections":[{"sectionFormat":{"pageWidth":612,"pageHeight":792,"leftMargin":72,"rightMargin":72,"topMargin":72,"bottomMargin":72,"differentFirstPage":false,"differentOddAndEvenPages":false,"headerDistance":36,"footerDistance":36,"bidi":false},"blocks":[{"paragraphFormat":{"afterSpacing":30,"styleName":"Heading1","listFormat":{},"characterFormat":{},"inlines":[{"characterFormat":{},"text":"Adventure Works Cycles"}]}],"headersFooters":{"header":{"blocks":[{"paragraphFormat":{"listFormat":{},"characterFormat":{},"inlines":[]}}],"footer":{"blocks":[{"paragraphFormat":{"listFormat":{},"characterFormat":{},"inlines":[]}}]},"characterFormat":{"bold":false,"italic":false,"fontSize":11,"fontFamily":"Calibri","underline":"None","strikethrough":"None","baselineAlignment":"Normal","highlightColor":"NoColor","fontColor":"empty","fontSizeBidi":11,"fontFamilyBidi":"Calibri","allCaps":false},"paragraphFormat":{"leftIndent":0,"rightIndent":0,"firstLineIndent":0,"textAlignment":"Left","beforeSpacing":0,"afterSpacing":0,"lineSpacing":1.0791666507720947,"lineSpacingType":"Multiple","listFormat":{},"bidi":false},"defaultTabWidth":36,"trackChanges":false,"enforcement":false,"hashValue":"","saltValue":"","formatting":false,"protectionType":"NoProtection","dontUseHTMLParagraphAutoSpacing":false,"formFieldShading":true,"styles":[{"name":"Normal","type":"Paragraph","paragraphFormat":{"lineSpacin

```

```

g":1.149999976158142,"lineSpacingType":"Multiple","listFormat":{},"characterFormat":{"fontFamily":"Calibri"},"next":"Normal"},{"name":"Default Paragraph Font","type":"Character","characterFormat":{}}, {"name":"Heading 1 Char","type":"Character","characterFormat":{"fontSize":16,"fontFamily":"Calibri Light","fontColor":"#2F5496"},"basedOn":"Default Paragraph Font"}, {"name":"Heading 1","type":"Paragraph","paragraphFormat":{"beforeSpacing":12,"afterSpacing":0,"outlineLevel":"Level1","listFormat":{},"characterFormat":{"fontSize":16,"fontFamily":"Calibri Light","fontColor":"#2F5496"},"basedOn":"Normal","link":"Heading 1 Char","next":"Normal"}, {"name":"Heading 2 Char","type":"Character","characterFormat":{"fontSize":13,"fontFamily":"Calibri Light","fontColor":"#2F5496"},"basedOn":"Default Paragraph Font"}, {"name":"Heading 2","type":"Paragraph","paragraphFormat":{"beforeSpacing":2,"afterSpacing":6,"outlineLevel":"Level2","listFormat":{},"characterFormat":{"fontSize":13,"fontFamily":"Calibri Light","fontColor":"#2F5496"},"basedOn":"Normal","link":"Heading 2 Char","next":"Normal"}, {"name":"Heading 3 Char","type":"Paragraph","paragraphFormat":{"leftIndent":0,"rightIndent":0,"firstLineIndent":0,"textAlignment":"Left","beforeSpacing":2,"afterSpacing":0,"lineSpacing":1.0791666507720947,"lineSpacingType":"Multiple","outlineLevel":"Level3","listFormat":{},"characterFormat":{"fontSize":12,"fontFamily":"Calibri Light","fontColor":"#1F3763"},"basedOn":"Normal","link":"Heading 3 Char","next":"Normal"}, {"name":"Heading 3 Char","type":"Character","characterFormat":{"fontSize":12,"fontFamily":"Calibri Light","fontColor":"#1F3763"},"basedOn":"Default Paragraph Font"}, {"name":"Heading 4","type":"Paragraph","paragraphFormat":{"leftIndent":0,"rightIndent":0,"firstLineIndent":0,"textAlignment":"Left","beforeSpacing":2,"afterSpacing":0,"lineSpacing":1.0791666507720947,"lineSpacingType":"Multiple","outlineLevel":"Level4","listFormat":{},"characterFormat":{"italic":true,"fontFamily":"Calibri Light","fontColor":"#2F5496"},"basedOn":"Normal","link":"Heading 4 Char","next":"Normal"}, {"name":"Heading 4 Char","type":"Character","characterFormat":{"italic":true,"fontFamily":"Calibri Light","fontColor":"#2F5496"},"basedOn":"Default Paragraph Font"}, {"name":"Heading 5","type":"Paragraph","paragraphFormat":{"leftIndent":0,"rightIndent":0,"firstLineIndent":0,"textAlignment":"Left","beforeSpacing":2,"afterSpacing":0,"lineSpacing":1.0791666507720947,"lineSpacingType":"Multiple","outlineLevel":"Level5","listFormat":{},"characterFormat":{"fontFamily":"Calibri Light","fontColor":"#2F5496"},"basedOn":"Normal","link":"Heading 5 Char","next":"Normal"}, {"name":"Heading 5 Char","type":"Character","characterFormat":{"fontFamily":"Calibri Light","fontColor":"#2F5496"},"basedOn":"Default Paragraph Font"}, {"name":"Heading 6","type":"Paragraph","paragraphFormat":{"leftIndent":0,"rightIndent":0,"firstLineIndent":0,"textAlignment":"Left","beforeSpacing":2,"afterSpacing":0,"lineSpacing":1.0791666507720947,"lineSpacingType":"Multiple","outlineLevel":"Level6","listFormat":{},"characterFormat":{"fontFamily":"Calibri Light","fontColor":"#1F3763"},"basedOn":"Normal","link":"Heading 6 Char","next":"Normal"}, {"name":"Heading 6 Char","type":"Character","characterFormat":{"fontFamily":"Calibri Light","fontColor":"#1F3763"},"basedOn":"Default Paragraph Font"}], "lists":[], "abstractLists":[], "comments":[], "revisions":[], "customXml1":[]}'
// showSpinner() will make the spinner visible

```

```

        showSpinner((document.getElementById('container') as HTMLElement) as
HTMLElement);
        // Open the default document
        (this.container as DocumentEditorContainerComponent
).documentEditor.open(data);
        setInterval(function() {
            // hideSpinner() method used hide spinner
            hideSpinner((document.getElementById('container') as
HTMLElement));
        }, 5000);
    }
}

```

MAIN.TS

```

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

```

Note: In above example, we have used setInterval to hide spinner, just for demo purpose.

Resize document editor in Angular Document editor component

In this article, we are going to see how to change height and width of Documenteditor.

Change height of Document Editor

DocumentEditorContainer initially render with default height. You can change height of documenteditor using [height](#) property, the value which is in pixel.

The following example code illustrates how to change height of Document Editor.

`typescript

```

import { Component, OnInit } from '@angular/core';
import {
    ToolbarService,
} from '@syncfusion/ej2-angular-documenteditor';
@Component({
    selector: 'app-root',
    // specified the height as 600px
    template: <ejs-documenteditorcontainer #documenteditor_default
serviceUrl="https://ej2services.syncfusion.com/production/web-
services/api/documenteditor/" height="600px" style="display:block" [enableToolbar]=true>
</ejs-documenteditorcontainer>,
    providers: [ToolbarService],
})
export class AppComponent implements OnInit {

```

```
ngOnInit(): void {
}
}
```

Similarly, you can use [height](#) property for DocumentEditor also.

Change width of Document Editor

DocumentEditorContainer initially render with default width. You can change width of documenteditor using [width](#) property, the value which is in percent.

The following example code illustrates how to change width of Document Editor.

```
`typescript
import { Component, OnInit } from '@angular/core';
import {
  ToolbarService,
} from '@syncfusion/ej2-angular-documenteditor';
@Component({
  selector: 'app-root',
  // specified the height as 600px
  template: <ejs-documenteditorcontainer #documenteditor_default
  serviceUrl="https://ej2services.syncfusion.com/production/web-
  services/api/documenteditor/" width="100%" style="display:block" [enableToolbar]=true>
  </ejs-documenteditorcontainer>,
  providers: [ToolbarService],
})
export class AppComponent implements OnInit {
  ngOnInit(): void {
  }
}
`
```

Similarly, you can use [width](#) property for DocumentEditor also.

Resize Document Editor

Using [resize](#) method, you change height and width of Document editor.

The following example code illustrates how to fit Document Editor to browser window size.

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

```
DocumentEditorContainerComponent,
ToolbarService,
} from '@syncfusion/ej2-angular-documenteditor';
@Component({
  selector: 'app-root',
  // specifies the template string for the DocumentEditorContainer component
  template: <ejs-documenteditorcontainer #documenteditor_default
  serviceUrl="https://ej2services.syncfusion.com/production/web-
  services/api/documenteditor/" height="600px" style="display:block" [enableToolbar]=true
  (created)="onCreate()"> </ejs-documenteditorcontainer>,
  providers: [ToolbarService],
})
export class AppComponent implements OnInit {
  @ViewChild('documenteditor_default')
  public container: DocumentEditorContainerComponent;
  ngOnInit(): void {}
  onCreate(): void {
    setInterval(() => {
      this.updateDocumentEditorSize();
    }, 100);
    //Adds event listener for browser window resize event.
    window.addEventListener('resize', this.onWindowResize.bind(this));
  }
  onWindowResize() {
    //Resizes the document editor component to fit full browser window automatically whenever the
    browser resized.
    this.updateDocumentEditorSize();
  }
  updateDocumentEditorSize() {
    //Resizes the document editor component to fit full browser window.
    var windowWidth = window.innerWidth;
    var windowHeight = window.innerHeight;
    this.container.resize(windowWidth, windowHeight);
  }
}
```

```
}
,
```

Export document as pdf in Angular Document editor component

In this article, we are going to see how to export the document as PDF format. You can export the document as PDF in following ways:

Export the document as pdf in client-side

Use [pdf export component](#) in application level to export the document as pdf using [exportasimage](#) API. Here, all pages will be converted to image and inserted as pdf pages(works like print as PDF).

Note:

- You can install the pdf export packages from this [link](#).
- There is one limitation we can't search the text because we are exporting the pdf as image.

The following example code illustrates how to export the document as pdf in client-side.

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

import { DocumentEditorContainerComponent, ImageFormat, ToolbarService } from '@syncfusion/ej2-angular-documenteditor';

import {
  PdfBitmap,
  PdfDocument,
  PdfPageOrientation,
  PdfPageSettings,
  PdfSection,
  SizeF,
} from '@syncfusion/ej2-pdf-export';

@Component({
  selector: 'app-root',
  // specifies the template string for the DocumentEditorContainer component
  template: `<button id='export'(click)="onClick(this)">Export</button><ejs-documenteditorcontainer #documenteditor_default
    serviceUrl="https://ej2services.syncfusion.com/production/web-services/api/documenteditor/" height="600px" style="display:block" [enableToolbar]=true>
  </ejs-documenteditorcontainer>`,
  providers: [ToolbarService]
})

export class AppComponent implements OnInit {
```

```
@ViewChild('documenteditor_default')
public container: DocumentEditorContainerComponent;
ngOnInit(): void {
}
onClick():void {
let obj = this;
let pdfdocument: PdfDocument = new PdfDocument();
let count: number = obj.container.documentEditor.pageCount;
obj.container.documentEditor.documentEditorSettings.printDevicePixelRatio = 2;
let loadedPage = 0;
for (let i = 1; i <= count; i++) {
setTimeout(() => {
let format: ImageFormat = 'image/jpeg' as ImageFormat;
// Getting pages as image
let image = obj.container.documentEditor.exportAsImage(i, format);
image.onload = function () {
let imageHeight = parseInt(
image.style.height.toString().replace('px', ''))
);
let imageWidth = parseInt(
image.style.width.toString().replace('px', ''))
);
let section: PdfSection = pdfdocument.sections.add() as PdfSection;
let settings: PdfPageSettings = new PdfPageSettings(0);
if (imageWidth > imageHeight) {
settings.orientation = PdfPageOrientation.Landscape;
}
settings.size = new SizeF(imageWidth, imageHeight);
(section as PdfSection).setPageSettings(settings);
let page = section.pages.add();
let graphics = page.graphics;
let imageStr = image.src.replace('data:image/jpeg;base64,', '');
let pdfImage = new PdfBitmap(imageStr);
```



```

graphics.drawImage(pdfImage, 0, 0, imageWidth, imageHeight);
loadedPage++;
if (loadedPage == count) {
// Exporting the document as pdf
pdfdocument.save(
(obj.container.documentEditor.documentName === "
? 'sample'
: obj.container.documentEditor.documentName) + '.pdf'
);
}
};
}, 500);
}
}
}
,

```

Export document as pdf in server-side using Syncfusion DocIO

With the help of [Syncfusion DocIO](#), you can export the document as PDF in server-side. Here, you can search the text.

The following way illustrates how to convert the document as PDF:

- Using [serialize](#) API, convert the document as Sfdt and send it to server-side.

The following example code illustrates how to convert the document to sfdt and pass it to server-side.

```

`typescript
import { Component, OnInit, ViewChild } from '@angular/core';
import { DocumentEditorContainerComponent, ToolbarService } from '@syncfusion/ej2-angular-
documenteditor';
@Component({
selector: 'app-root',
// specifies the template string for the DocumentEditorContainer component
template: <button id='export'(click)="onClick(this)">Export</button><ejs-
documenteditorcontainer #documenteditor_default
serviceUrl="https://ej2services.syncfusion.com/production/web-
services/api/documenteditor/" height="600px" style="display:block" [enableToolbar]=true>
</ejs-documenteditorcontainer>,

```

```

providers: [ToolbarService]
})
export class AppComponent implements OnInit {
  @ViewChild('documenteditor_default')
  public container: DocumentEditorContainerComponent;
  ngOnInit(): void {
  }
  onClick():void {
    let obj=this;
    let http: XMLHttpRequest = new XMLHttpRequest();
    // Replace your running web service Url here
    http.open('POST', 'http://localhost:62869/api/documenteditor/ExportPdf');
    http.setRequestHeader('Content-Type', 'application/json;charset=UTF-8');
    http.responseType = 'json';
    //Serialize document content as SFDt.
    let sfdt: any = { content: obj.container.documentEditor.serialize() };
    //Send the sfdt content to server side.
    http.send(JSON.stringify(sfdt));
  }
}
`

```

- Using Save API in server-side, you can convert the sfdt to stream.
- Finally, convert the stream to PDF using [Syncfusion.DocIORenderer.Net.Core](#) library.

The following example code illustrates how to process the sfdt in server-side.

```

`c#
[AcceptVerbs("Post")]
[HttpPost]
[EnableCors("AllowAllOrigins")]
[Route("ExportPdf")]
public void ExportPdf([FromBody]SaveParameter data)
{
  // Converts the sfdt to stream
  Stream document = WordDocument.Save(data.content, FormatType.Docx);

```

```

Syncfusion.DocIO.DLS.WordDocument doc = new Syncfusion.DocIO.DLS.WordDocument(document,
Syncfusion.DocIO.FormatType.Docx);
//Instantiation of DocIORenderer for Word to PDF conversion
DocIORenderer render = new DocIORenderer();
//Converts Word document into PDF document
PdfDocument pdfDocument = render.ConvertToPDF(doc);
// Saves the document to server machine file system, you can customize here to save into databases or
file servers based on requirement.
FileStream fileStream = new FileStream("sample.pdf", FileMode.OpenOrCreate, FileAccess.ReadWrite);
//Saves the PDF file
pdfDocument.Save(fileStream);
pdfDocument.Close();
fileStream.Close();
document.Close();
}
`

```

Get the complete working sample in this [link](#).

Customize font family drop down in Angular Document editor component

Document editor provides an options to customize the font family drop down list values using [fontFamilies](#) in Document editor settings. Fonts which are added in fontFamilies of [documentEditorSettings](#) will be displayed on font drop down list of text properties pane and font dialog.

Similarly, you can use [documentEditorSettings](#) property for DocumentEditor also.

The following example code illustrates how to change the font families in Document editor container.

```

`typescript
import { Component, OnInit } from '@angular/core';
import { ToolbarService } from '@syncfusion/ej2-angular-documenteditor';
@Component({
  selector: 'app-root',
  // specifies the template string for the DocumentEditorContainer component
  template: <ejs-documenteditorcontainer
serviceUrl="https://ej2services.syncfusion.com/production/web-
services/api/documenteditor/" height="600px" style="display:block"
[documentEditorSettings]= "fontFamilies" [enableToolbar]=true> </ejs-
documenteditorcontainer>,
  providers: [ToolbarService]
})

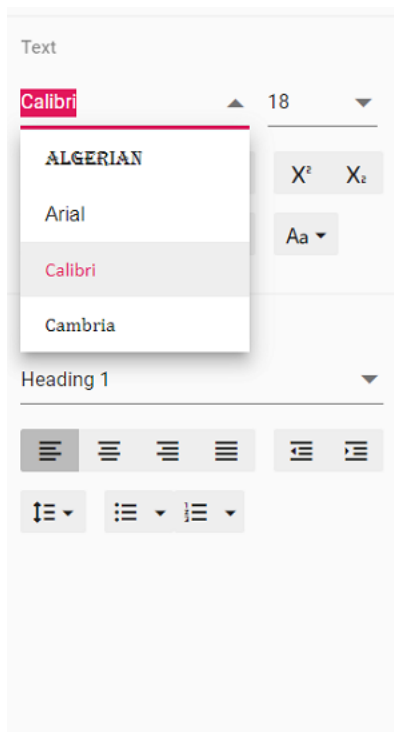
```

```

export class AppComponent implements OnInit {
  // Add required font families to list it in font drop down
  public fontFamilies={fontFamilies:['Algerian', 'Arial', 'Calibri', 'Cambria']};
  ngOnInit(): void {
  }
}

```

Output will be like below:



Auto save document in document editor in Angular Document editor component

In this article, we are going to see how to autosave the document in AWS S3. You can automatically save the edited content in regular intervals of time. It helps reduce the risk of data loss by saving an open document automatically at customized intervals.

The following example illustrates how to auto save the document in AWS S3.

- In the client-side, using content change event, we can automatically save the edited content in regular intervals of time. Based on `contentChanged` boolean, the document send as Docx format to server-side using [saveAsBlob](#) method.

`typescript

/

- Add below codes in app.component.html file

```
*/
<ejs-documenteditorcontainer #documenteditor_default [enableToolbar]=true (created)="onCreate()"
(contentChange)="onContentChange()" height="600px" style="display:block;"></ejs-
documenteditorcontainer>
/
```

- Add below codes in app.component.ts file

```
*/
@Component({
  selector: 'app-root',
  templateUrl: 'app.component.html',
  encapsulation: ViewEncapsulation.None,
  providers: [ToolbarService]
})
export class AppComponent {
  public hostUrl: string = 'https://ej2services.syncfusion.com/production/web-services/';
  @ViewChild('documenteditor_default')
  public container: DocumentEditorContainerComponent;
  contentChanged: boolean;
  onCreate(): void {
    this.container.serviceUrl = 'https://ej2services.syncfusion.com/production/web-
services/api/documenteditor/';
    setInterval(() => {
      if (this.contentChanged) {
        //You can save the document as below
        this.container.documentEditor.saveAsBlob('Docx').then((blob: Blob) => {
          console.log('Saved sucessfully');
          let exportedDocument: Blob = blob;
          //Now, save the document where ever you want.
          let formData: FormData = new FormData();
          formData.append('fileName', 'sample.docx');
          formData.append('data', exportedDocument);
          / tslint:disable /
          var req = new XMLHttpRequest();
```

```
// Replace your running Url here
req.open(
  'POST',
  'http://localhost:62869/api/documenteditor/SaveToS3',
  true
);
req.onreadystatechange = () => {
  if (req.readyState === 4) {
    if (req.status === 200 || req.status === 304) {
      console.log('Saved sucessfully');
    }
  }
};
req.send(formData);
});
this.contentChanged = false;
}
}, 1000);
}
onContentChange(): void {
  this.contentChanged = true;
}
}
`
```

- In server-side, configure the access key and secret key in `web.config` file and register profile in `startup.cs`.

In `web.config`, add key like below format:

```
`c#
<appSettings>
  <add key="AWSProfileName" value="sync_development" />
  <add key="AWSAccessKey" value="" />
  <add key="AWSSecretKey" value="" />
```

```
</appSettings>
```

```
,
```

In `startup.cs`, register profile in below format:

```
`c#
```

```
Amazon.Util.ProfileManager.RegisterProfile("sync_development", "", "");
```

```
,
```

- In server-side, Receives the stream content from client-side and process it to save the document in aws s3. Add Web API in controller file like below to save the document in aws s3.

```
`c#
```

```
[AcceptVerbs("Post")]
```

```
[HttpPost]
```

```
[EnableCors("AllowAllOrigins")]
```

```
[Route("SaveToS3")]
```

```
public string SaveToS3()
```

```
{
```

```
    IFormFile file = HttpContext.Request.Form.Files[0];
```

```
    Stream stream = new MemoryStream();
```

```
    file.CopyTo(stream);
```

```
    UploadFileStreamToS3(stream, "documenteditor", "", "GettingStarted.docx");
```

```
    stream.Close();
```

```
    return "Sucess";
```

```
}
```

```
public bool UploadFileStreamToS3(System.IO.Stream localFilePath, string bucketName, string  
subDirectoryInBucket, string fileNameInS3)
```

```
{
```

```
    AWSCredentials credentials = new StoredProfileAWSCredentials("sync_development");
```

```
    IAmazonS3 client = new AmazonS3Client(credentials, Amazon.RegionEndpoint.USEast1);
```

```
    TransferUtility utility = new TransferUtility(client);
```

```
    TransferUtilityUploadRequest request = new TransferUtilityUploadRequest();
```

```
    if (subDirectoryInBucket == "" || subDirectoryInBucket == null)
```

```
{
```

```
        request.BucketName = bucketName; //no subdirectory just bucket name
```

```
}
```

```

else
{ // subdirectory and bucket name
request.BucketName = bucketName + @"/" + subDirectoryInBucket;
}
request.Key = fileNameInS3; //file name up in S3
request.InputStream = localFilePath;
utility.Upload(request); //commencing the transfer
return true; //indicate that the file was sent
}
`

```

Get the complete working sample in this [link](#).

Retrieve the bookmark content as text in Angular Document editor component

You can get the bookmark or whole document content from the Angular Document Editor component as plain text and SFDT (rich text).

Get the bookmark content as plain text

You can [selectBookmark](#) API to navigate to the bookmark and use [text](#) API to get the bookmark content as plain text from Angular Document Editor component.

The following example code illustrates how to get the bookmark content as plain text.

```

`typescript
import { Component, OnInit, ViewChild } from '@angular/core';
import {
  ToolbarService,
  DocumentEditorContainerComponent,
} from '@syncfusion/ej2-angular-documenteditor';
@Component({
  selector: 'app-root',
  // specifies the template string for the DocumentEditorContainer component
  template: <ejs-documenteditorcontainer #documenteditor_default
  serviceUrl="https://ej2services.syncfusion.com/production/web-
  services/api/documenteditor/" height="600px" style="display:block" [enableToolbar]=true
  (created)="onCreated()"> </ejs-documenteditorcontainer>,
  providers: [ToolbarService],
})
export class AppComponent implements OnInit {
  @ViewChild('documenteditor_default')

```



```

public container: DocumentEditorContainerComponent;
ngOnInit(): void {}
onCreated() {
  // To insert text in cursor position
  this.container.documentEditor.editor.insertText('Document editor');
  // To select all the content in document
  this.container.documentEditor.selection.selectAll();
  // Insert bookmark to selected content
  this.container.documentEditor.editor.insertBookmark('Bookmark1');
  // Provide your bookmark name to navigate to specific bookmark
  this.container.documentEditor.selection.selectBookmark('Bookmark1');
  // To get the selected content as text
  let selectedContent = this.container.documentEditor.selection.text;
}
}
,

```

To get the bookmark content as SFDT (rich text), please check this [link](#)

[Get the whole document content as text](#)

You can use [text](#) API to get the whole document content as plain text from Angular Document Editor component.

The following example code illustrates how to get the whole document content as plain text.

```

`typescript
import { Component, OnInit, ViewChild } from '@angular/core';
import {
  ToolbarService,
  DocumentEditorContainerComponent,
} from '@syncfusion/ej2-angular-documenteditor';
@Component({
  selector: 'app-root',
  // specifies the template string for the DocumentEditorContainer component
  template: <ejs-documenteditorcontainer #documenteditor_default
  serviceUrl="https://ej2services.syncfusion.com/production/web-
  services/api/documenteditor/" height="600px" style="display:block" [enableToolbar]=true
  (created)="onCreated()"> </ejs-documenteditorcontainer>,

```

```

providers: [ToolbarService],
})
export class AppComponent implements OnInit {
  @ViewChild('documenteditor_default')
  public container: DocumentEditorContainerComponent;
  ngOnInit(): void {}
  onCreate() {
    // To insert text in cursor position
    this.container.documentEditor.editor.insertText('Document editor');
    // To select all the content in document
    this.container.documentEditor.selection.selectAll();
    // To get the content as text
    let selectedContent: string = this.container.documentEditor.selection.text;
  }
}

```

Get the whole document content as SFDT(rich text)

You can use [serialize](#) API to get the whole document content as SFDT string from Angular Document Editor component.

The following example code illustrates how to get the whole document content as SFDT.

```

`typescript
import { Component, OnInit, ViewChild } from '@angular/core';
import {
  ToolbarService,
  DocumentEditorContainerComponent,
} from '@syncfusion/ej2-angular-documenteditor';
@Component({
  selector: 'app-root',
  // specifies the template string for the DocumentEditorContainer component
  template: <ejs-documenteditorcontainer #documenteditor_default
  serviceUrl="https://ej2services.syncfusion.com/production/web-
  services/api/documenteditor/" height="600px" style="display:block" [enableToolbar]=true
  (created)="onCreated()"> </ejs-documenteditorcontainer>,
  providers: [ToolbarService],

```

```

})
export class AppComponent implements OnInit {
  @ViewChild('documenteditor_default')
  public container: DocumentEditorContainerComponent;
  ngOnInit(): void {}
  onCreate() {
    // To insert text in cursor position
    this.container.documentEditor.editor.insertText('Document editor');
    // To get the content as SFDT
    let selectedContent: string = this.container.documentEditor.serialize();
  }
}
`

```

Get the header content as text

You can use [goToHeader](#) API to navigate the selection to the header and then use [text](#) API to get the content as plain text.

The following example code illustrates how to get the header content as plain text.

```

`typescript
import { Component, OnInit, ViewChild } from '@angular/core';
import {
  ToolbarService,
  DocumentEditorContainerComponent,
} from '@syncfusion/ej2-angular-documenteditor';
@Component({
  selector: 'app-root',
  // specifies the template string for the DocumentEditorContainer component
  template: <ejs-documenteditorcontainer #documenteditor_default
    serviceUrl="https://ej2services.syncfusion.com/production/web-
    services/api/documenteditor/" height="600px" style="display:block" [enableToolbar]=true
    (created)="onCreate()"> </ejs-documenteditorcontainer>,
  providers: [ToolbarService],
})
export class AppComponent implements OnInit {
  @ViewChild('documenteditor_default')

```

```

public container: DocumentEditorContainerComponent;
ngOnInit(): void {}
onCreated() {
  // To navigate the selection to header
  this.container.documentEditor.selection.goToHeader();
  // To insert text in cursor position
  this.container.documentEditor.editor.insertText('Document editor');
  // To select all the content in document
  this.container.documentEditor.selection.selectAll();
  // To get the selected content as text
  let selectedContent: string = this.container.documentEditor.selection.text;
}
}
`

```

Similarly, you can use [goToFooter](#) API to navigate the selection to the footer and then use [text](#) API to get the content as plain text.

Get current word in Angular Document editor component

You can get the current word or paragraph content from the Angular Document Editor component as plain text and SFDT (rich text).

Select and get the word in current cursor position

You can use [selectCurrentWord](#) API in selection module to select the current word at cursor position and use [text](#) API to get the selected content as plain text from Angular Document Editor component.

The following example code illustrates how to select and get the current word as plain text.

```

`typescript
import { Component, OnInit, ViewChild } from '@angular/core';
import {
  ToolbarService,
  DocumentEditorContainerComponent,
} from '@syncfusion/ej2-angular-documenteditor';
@Component({
  selector: 'app-root',
  // specifies the template string for the DocumentEditorContainer component
  template: <ejs-documenteditorcontainer #documenteditor_default
  serviceUrl="https://ej2services.syncfusion.com/production/web-

```

```

services/api/documenteditor/" height="600px" style="display:block" [enableToolbar]=true
(created)="onCreated()"> </ejs-documenteditorcontainer>,
providers: [ToolbarService],
})
export class AppComponent implements OnInit {
  @ViewChild('documenteditor_default')
  public container: DocumentEditorContainerComponent;
  ngOnInit(): void {}
  onCreate() {
    // To insert text in cursor position
    this.container.documentEditor.editor.insertText('Document editor');
    // Move selection to previous character
    this.container.documentEditor.selection.moveToPreviousCharacter();
    // To select the current word in document
    this.container.documentEditor.selection.selectCurrentWord();
    // To get the selected content as text
    let selectedContent:string = this.container.documentEditor.selection.text;
  }
}
`

```

To get the bookmark content as SFDT (rich text), please check this [link](#)

Select and get the paragraph in current cursor position

You can use [selectParagraph](#) API in selection module to select the current paragraph at cursor position and use [text](#) API or [sfdt](#) API to get the selected content as plain text or SFDT from Angular Document Editor component.

The following example code illustrates how to select and get the current paragraph as SFDT.

```

`typescript
import { Component, OnInit, ViewChild } from '@angular/core';
import {
  ToolbarService,
  DocumentEditorContainerComponent,
} from '@syncfusion/ej2-angular-documenteditor';
@Component({
  selector: 'app-root',

```

```
// specifies the template string for the DocumentEditorContainer component
template: <ejs-documenteditorcontainer #documenteditor_default
serviceUrl="https://ej2services.syncfusion.com/production/web-
services/api/documenteditor/" height="600px" style="display:block" [enableToolbar]=true
(created)="onCreated()"> </ejs-documenteditorcontainer>,
providers: [ToolbarService],
})
export class AppComponent implements OnInit {
  @ViewChild('documenteditor_default')
  public container: DocumentEditorContainerComponent;
  ngOnInit(): void {}
  onCreated() {
    // To insert text in cursor position
    this.container.documentEditor.editor.insertText('Document editor');
    // To select the current paragraph in document
    this.container.documentEditor.selection.selectParagraph();
    // To get the selected content as SFDT
    let selectedContent: string = this.container.documentEditor.selection.sfdt;
  }
}
```

Insert page number and navigate to page in Angular Document editor component

You can insert page number and navigate to specific page in Angular Document Editor component by following ways.

Insert page number

You can use [insertPageNumber](#) API in editor module to insert the page number in current cursor position. By default, Page number will insert in Arabic number style. You can change it, by providing the number style in parameter.

Note: Currently, Documenteditor have options to insert page number at current cursor position.

The following example code illustrates how to insert page number in header.

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

```

} from '@syncfusion/ej2-angular-documenteditor';
@Component({
  selector: 'app-root',
  // specifies the template string for the DocumentEditorContainer component
  template: <ejs-documenteditorcontainer #documenteditor_default
  serviceUrl="https://ej2services.syncfusion.com/production/web-
  services/api/documenteditor/" height="600px" style="display:block" [enableToolbar]=true
  (created)="onCreated()"> </ejs-documenteditorcontainer>,
  providers: [ToolbarService],
})
export class AppComponent implements OnInit {
  @ViewChild('documenteditor_default')
  public container: DocumentEditorContainerComponent;
  ngOnInit(): void {}
  onCreated() {
    // To insert text in cursor position
    this.container.documentEditor.editor.insertText('Document editor');
    // To move the selection to header
    this.container.documentEditor.selection.goToHeader();
    // Insert page number in the current cursor position
    this.container.documentEditor.editor.insertPageNumber();
  }
}

```

Also, you use [insertField](#) API in Editor module to insert the Page number in current position

```

`typescript
//Current page number
this.container.documentEditor.editor.insertField('PAGE * MERGEFORMAT', '1');

```

Get page count

You can use [pageCount](#) API to gets the total number of pages in Document.

The following example code illustrates how to get the number of pages in Document.

```

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

```

```

import {
  ToolbarService,
  DocumentEditorContainerComponent,
} from '@syncfusion/ej2-angular-documenteditor';
@Component({
  selector: 'app-root',
  // specifies the template string for the DocumentEditorContainer component
  template: <ejs-documenteditorcontainer #documenteditor_default
  serviceUrl="https://ej2services.syncfusion.com/production/web-
  services/api/documenteditor/" height="600px" style="display:block" [enableToolbar]=true
  (created)="onCreated()"> </ejs-documenteditorcontainer>,
  providers: [ToolbarService],
})
export class AppComponent implements OnInit {
  @ViewChild('documenteditor_default')
  public container: DocumentEditorContainerComponent;
  ngOnInit(): void {}
  onCreated() {
    // To insert text in cursor position
    this.container.documentEditor.editor.insertText('Document editor');
    // To get the total number of pages
    let pageCount = this.container.documentEditor.pageCount;
  }
}

```

Navigate to specific page

You can use [goToPage](#) API in Selection module to move selection to the start of the specified page number.

The following example code illustrates how to move selection to specific page.

```

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

```



```

} from '@syncfusion/ej2-angular-documenteditor';
@Component({
  selector: 'app-root',
  // specifies the template string for the DocumentEditorContainer component
  template: <ejs-documenteditorcontainer #documenteditor_default
  serviceUrl="https://ej2services.syncfusion.com/production/web-
  services/api/documenteditor/" height="600px" style="display:block" [enableToolbar]=true
  (created)="onCreated()"> </ejs-documenteditorcontainer>,
  providers: [ToolbarService],
})
export class AppComponent implements OnInit {
  @ViewChild('documenteditor_default')
  public container: DocumentEditorContainerComponent;
  ngOnInit(): void {}
  onCreated() {
    // To move selection to page number 2
    this.container.documentEditor.selection.goToPage(2);
  }
}

```

Move selection to specific position in Angular Document editor component

Using [select](#) API in selection module, You can set cursor position to anywhere in the document.

Selects content based on start and end hierarchical index

Hierarchical index will be in below format.

sectionIndex;blockIndex;offset

The following code snippet illustrate how to select using hierarchical index.

```

`typescript
// Selection will occur between provided start and end offset
this.documentEdContainerIns.documentEditor.editor.insertText("Welcome");
// The below code will select the letters "We" from inserted text "Welcome"
this.documentEdContainerIns.documentEditor.selection.select("0;0;0", "0;0;2");

```

The following table illustrates about Hierarchical index in detail.

Element	Hierarchical Format	Explanation
---------	---------------------	-------------

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

|Move selection to Paragraph |sectionIndex;blockIndex;offset
Ex: 0;0;0|It moves the cursor to the start of paragraph. |

|Move selection to Table |sectionIndex;tableIndex;rowIndex;cellIndex;blockIndex;offset
Ex: 0;0;0;0;1;0|It moves the cursor to the second paragraph which is inside first row and cell of table. |

|Move selection to header |pageIndex;H;sectionIndex;blockIndex;offset
Ex: 1;H;0;0;0|It moves cursor to the header in second page. |

|Move selection to Footer |pageIndex;F;sectionIndex;blockIndex;offset
Ex: 1;F;0;0;0|It moves cursor to the footer in second page. |

Get the selection start and end hierarchical index

Using [startOffset](#), you can get start hierarchical index which denotes the start index of current selection.

Similarly, using [endOffset](#), you can get end hierarchical index which denotes the end index of current selection.

The following code snippet illustrate how to get the selection start and end offset on selection changes in document.

```
`typescript
import { Component, OnInit, ViewChild } from '@angular/core';
import {
  ToolbarService,
  DocumentEditorContainerComponent,
} from '@syncfusion/ej2-angular-documenteditor';
@Component({
  selector: 'app-root',
  // specifies the template string for the DocumentEditorContainer component
  template: <ejs-documenteditorcontainer #documenteditor_default
  serviceUrl="https://ej2services.syncfusion.com/production/web-
  services/api/documenteditor/" height="600px" style="display:block" [enableToolbar]=true
  (selectionChange)="selectionChanges()"> </ejs-documenteditorcontainer>,
  providers: [ToolbarService],
})
export class AppComponent implements OnInit {
  @ViewChild('documenteditor_default')
  public container: DocumentEditorContainerComponent;
  ngOnInit(): void {}
  selectionChanges() {
    //Get the start index of current selection
```

```

let startOffset: string =
this.container.documentEditor.selection.startOffset;
//Get the end index of current selection
let endOffset: string = this.container.documentEditor.selection.endOffset;
}
}
`

```

Document editor have [selectionChange](#) event which is triggered whenever the selection changes in Document.

Selects the content based on left and top position

Here, you can specify the [selection settings](#) to select the content based on left and top position.

x denotes the left position and y denotes the top position and extend denotes whether to extend or update selection.

Please check below code sample for reference.

```

`typescript
this.container.documentEditor.selection.select({ x: 188.4814208984375 , y: 662.00005, extend: true });
`

```

Disable header and footer edit in document editor in Angular Document editor component

Disable header and footer edit in DocumentEditorContainer instance

You can use [restrictEditing](#) property to disable header and footer editing based on selection context type.

RestrictEditing allows you to restrict the document modification and makes the Document read only mode. So, by using this property, and if selection inside header or footer, you can set this property as true.

The following example code illustrates how to header and footer edit in **DocumentEditorContainer** instance.

```

`typescript
import { Component, OnInit, ViewChild } from '@angular/core';
import {
  ToolbarService,
  DocumentEditorContainerComponent,
} from '@syncfusion/ej2-angular-documenteditor';
@Component({
  selector: 'app-root',
  // specifies the template string for the DocumentEditorContainer component

```

```

template: <ejs-documenteditorcontainer #documenteditor_default
serviceUrl="https://ej2services.syncfusion.com/production/web-
services/api/documenteditor/" height="600px" style="display:block" [enableToolbar]=true
(selectionChange)="selectionChanges()"> </ejs-documenteditorcontainer>,
providers: [ToolbarService],
})
export class AppComponent implements OnInit {
  @ViewChild('documenteditor_default')
  public container: DocumentEditorContainerComponent;
  ngOnInit(): void {}
  selectionChanges() {
    // Check whether selection is in header
    if (this.container.documentEditor.selection.contextType.indexOf('Header') > -1 ||
    // Check whether selection is in Footer
    this.container.documentEditor.selection.contextType.indexOf('Footer') > -1 ) {
      // Change the document to read only mode
      this.container.restrictEditing = true;
    } else {
      // Change the document to editable mode
      this.container.restrictEditing = false;
    }
  }
}

```

Otherwise, you can disable clicking inside Header or Footer by using [closeHeaderFooter](#) API in selection module.

The following example code illustrates how to close header and footer when selection is inside header or footer in `DocumentEditorContainer` instance.

```

`typescript
import { Component, OnInit, ViewChild } from '@angular/core';
import {
  ToolbarService,
  DocumentEditorContainerComponent,
} from '@syncfusion/ej2-angular-documenteditor';

```

```

@Component({
  selector: 'app-root',
  // specifies the template string for the DocumentEditorContainer component
  template: <ejs-documenteditorcontainer #documenteditor_default
  serviceUrl="https://ej2services.syncfusion.com/production/web-
  services/api/documenteditor/" height="600px" style="display:block" [enableToolbar]=true
  (selectionChange)="selectionChanges()"> </ejs-documenteditorcontainer>,
  providers: [ToolbarService],
})
export class AppComponent implements OnInit {
  @ViewChild('documenteditor_default')
  public container: DocumentEditorContainerComponent;
  ngOnInit(): void {}
  selectionChanges() {
    // Check whether selection is in header
    if (this.container.documentEditor.selection.contextType.indexOf('Header') > -1 ||
    // Check whether selection is in Footer
    this.container.documentEditor.selection.contextType.indexOf('Footer') > -1 ) {
    // Close header Footer
    this.container.documentEditor.selection.closeHeaderFooter();
    }
  }
}

```

Disable header and footer edit in DocumentEditor instance

Like restrictEditing, you can use [isReadOnly](#) property in Document editor to disable header and footer edit.

The following example code illustrates how to header and footer edit in **DocumentEditor** instance.

```

`typescript
// import { Component, OnInit, ViewChild } from '@angular/core';
// import {
//   ToolbarService,
//   DocumentEditorContainerComponent,
// } from '@syncfusion/ej2-angular-documenteditor';

```

```
// @Component({
//   selector: 'app-root',
//   // specifies the template string for the DocumentEditorContainer component
//   template: <ejs-documenteditorcontainer #documenteditor_default
//     serviceUrl="https://ej2services.syncfusion.com/production/web-
//     services/api/documenteditor/" height="600px" style="display:block" [enableToolbar]=false
//     (selectionChange)="selectionChanges()"> </ejs-documenteditorcontainer>
// })
// export class AppComponent implements OnInit {
//   @ViewChild('documenteditor_default')
//   public container: DocumentEditorContainerComponent;
//   ngOnInit(): void {}
// }
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import {
  DocumentEditorComponent, PrintService, SfdtExportService, WordExportService, TextExportService,
  SelectionService,
  SearchService, EditorService, ImageResizerService, EditorHistoryService, ContextMenuService,
  OptionsPaneService, HyperlinkDialogService, TableDialogService, BookmarkDialogService,
  TableOfContentsDialogService,
  PageSetupDialogService, StyleDialogService, ListDialogService, ParagraphDialogService,
  BulletsAndNumberingDialogService,
  FontDialogService, TablePropertiesDialogService, BordersAndShadingDialogService,
  TableOptionsDialogService,
  CellOptionsDialogService, StylesDialogService
} from '@syncfusion/ej2-angular-documenteditor';
@Component({
  selector: 'app-root',
  template: `<ejs-documenteditor #documenteditor_default
    serviceUrl="https://ej2services.syncfusion.com/production/web-services/api/documenteditor/"
    height="330px" style="display:block" [isReadOnly]=false [enableSelection]=true
    [enablePrint]=true [enableSfdtExport]=true [enableWordExport]=true [enableOptionsPane]=true
    [enableContextMenu]=true
    [enableHyperlinkDialog]=true [enableBookmarkDialog]=true [enableTableOfContentsDialog]=true
    [enableSearch]=true
    [enableParagraphDialog]=true [enableListDialog]=true [enableTablePropertiesDialog]=true
    [enableBordersAndShadingDialog]=true`
```

```

[enablePageSetupDialog]=true [enableStyleDialog]=true [enableFontDialog]=true
[enableTableOptionsDialog]=true

[enableTableDialog]=true [enableImageResizer]=true [enableEditor]=true [enableEditorHistory]=true
(selectionChange)="selectionChanges()">
</ejs-documenteditor>`,
encapsulation: ViewEncapsulation.None,
providers: [PrintService, SfdtExportService, WordExportService, TextExportService, SelectionService,
SearchService, EditorService,
ImageResizerService, EditorHistoryService, ContextMenuService, OptionsPaneService,
HyperlinkDialogService, TableDialogService,
BookmarkDialogService, TableOfContentsDialogService, PageSetupDialogService, StyleDialogService,
ListDialogService,
ParagraphDialogService, BulletsAndNumberingDialogService, FontDialogService,
TablePropertiesDialogService,
BordersAndShadingDialogService, TableOptionsDialogService, CellOptionsDialogService,
StylesDialogService]
})
export class AppComponent {
  @ViewChild('documenteditor_default')
  public documentEditor: DocumentEditorComponent;
  selectionChanges() {
    // Check whether selection is in header
    if (this.documentEditor.selection.contextType.indexOf('Header') > -1 ||
    // Check whether selection is in Footer
    this.documentEditor.selection.contextType.indexOf('Footer') > -1) {
      // Change the document to read only mode
      this.documentEditor.isReadOnly = true;
    } else {
      // Change the document to editable mode
      this.documentEditor.isReadOnly = false;
    }
  };
}
`

```

[Insert text in current position in Angular Document editor component](#)

You can insert the text, paragraph and rich-text content in Angular Document Editor component.

Insert text in current cursor position

You can use [insertText](#) API in editor module to insert the text in current cursor position.

The following example illustrates how to add the text in current selection.

```
`typescript
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import {
  DocumentEditorContainerComponent, ToolbarService
} from '@syncfusion/ej2-angular-documenteditor';
@Component({
  selector: 'app-container',
  // specifies the template string for the Document Editor container component
  template: `<div><button ej2-button (click)="insertText()" >Insert Text</button>
<ejs-documenteditorcontainer #documenteditor_default
  serviceUrl="https://ej2services.syncfusion.com/production/web-services/api/documenteditor/"
  height="600px" style="display:block" [enableToolbar]=true> </ejs-documenteditorcontainer></div>`,
  encapsulation: ViewEncapsulation.None,
  providers: [ToolbarService]
})
export class AppComponent {
  @ViewChild('documenteditor_default')
  public container: DocumentEditorContainerComponent;
  public insertText(): void {
    // It will insert the provided text in current selection
    this.container.documentEditor.editor.insertText('Syncfusion');
  }
}
```

Please check below gif which illustrates how to insert text in current cursor position on button click:

![Insert text in current cursor position in Javascript document editor](../images/insert_text.gif)

Insert paragraph in current cursor position

To insert new paragraph at current selection, you can use [insertText](#) API with parameter as `\r\n` or `\n`.

The following example code illustrates how to add the new paragraph in current selection.

```
`typescript
// It will add the new paragraph in current selection
```



```
this.container.documentEditor.editor.insertText('\n');
```

```
,
```

Insert the rich-text content

To insert the HTML content, you have to convert the HTML content to SFDT Format using [web service](#). Then use [paste](#) API to insert the sfdt at current cursor position.

Note: Html string should be wellformatted html. [DocIO](#) support only wellformatted XHTML.

The following example illustrates how to insert the HTML content at current cursor position.

- Send the HTML content to server side for SFDT conversion. Refer to the following example to send the HTML content to server side and inserting it in current cursor position.

```
`typescript
import { Component, OnInit, ViewChild } from '@angular/core';
import {
  ToolbarService,
  DocumentEditorContainerComponent,
} from '@syncfusion/ej2-angular-documenteditor';
@Component({
  selector: 'app-root',
  // specifies the template string for the DocumentEditorContainer component
  template: <ejs-documenteditorcontainer #documenteditor_default
  serviceUrl="https://ej2services.syncfusion.com/production/web-
  services/api/documenteditor/" height="600px" style="display:block"
  [documentEditorSettings]= "searchHighlightColor" [enableToolbar]=true
  (created)="onCreated()"> </ejs-documenteditorcontainer>,
  providers: [ToolbarService],
})
export class AppComponent implements OnInit {
  @ViewChild('documenteditor_default')
  public container: DocumentEditorContainerComponent;
  ngOnInit(): void {}
  onCreated() {
    let proxy = this;
    let htmltags: string =
    "<?xml version='1.0' encoding='utf - 8'?><!DOCTYPE html PUBLIC '-//W3C//DTD XHTML 1.0
    Strict//EN' 'http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd'><html xmlns
    ='http://www.w3.org/1999/xhtml' xml:lang='en' lang='en'><body><h1>The img element</h1><img
```

```

src='https://www.w3schools.com/images/lamp.jpg' alt='Lamp Image' width='500'
height='600'/></body></html>";

document.getElementById('export').addEventListener('click', () => {
let http: XMLHttpRequest = new XMLHttpRequest();
http.open('POST', 'http://localhost:5000/api/documenteditor/LoadString');
http.setRequestHeader('Content-Type', 'application/json;charset=UTF-8');
http.responseType = 'json';
http.onreadystatechange = function () {
if (http.readyState === 4) {
if (http.status === 200 || http.status === 304) {
// Insert the sfdt content in cursor position using paste API
proxy.container.documentEditor.editor.paste(http.response);
} else {
alert('failed;');
}
}
};
let htmlContent: any = { content: htmltags };
http.send(JSON.stringify(htmlContent));
});
}
}
`

```

- Please refer the following code example for server-side web implementation for HTML conversion using DocumentEditor.

```

`c#
//API controller for the conversion.
[HttpPost]
public string LoadString([FromBody]InputParameter data)
{
// You can also load HTML file/string from server side.
Syncfusion.EJ2.DocumentEditor.WordDocument document =
Syncfusion.EJ2.DocumentEditor.WordDocument.LoadString(data.content, FormatType.Html); // Convert
the HTML to SFDT format.

```

```

string json = Newtonsoft.Json.JsonConvert.SerializeObject(document);
document.Dispose();
return json;
}

public class InputParameter
{
    public string content {get; set; }
}
`

```

Note: The above example illustrates inserting HTML content. Similarly, you can insert any rich-text content by converting any of the supported file formats (DOCX, DOC, WordML, HTML, RTF) to SFDT.

[Change the cursor color in document editor in Angular Document editor component](#)

Document Editor default cursor color is black. The user can change the color by overriding the css property using class name. The Document editor cursor css have a class named `e-de-blink-cursor`.

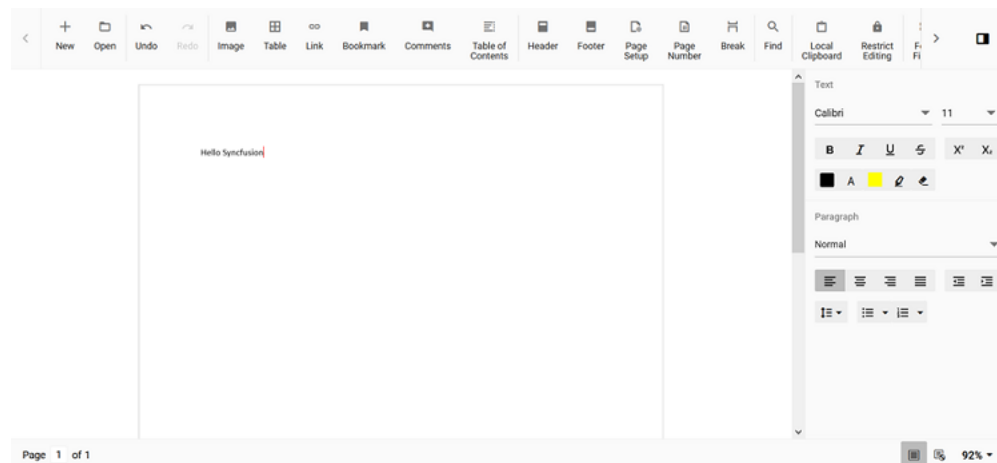
Please refer the below code snippet to change the cursor color to red.

```

`css
.e-de-blink-cursor {
    border-left: 1px solid red!important;
}
`

```

Output will be like below:



[Hide tool bar and properties pane in Angular Document editor component](#)

Document editor container provides the main document view area along with the built-in toolbar and properties pane.

Document editor provides just the main document view area. Here, the user can compose, view, and edit the Word documents. You may prefer to use this component when you want to design your own UI options for your application.

Hide the properties pane

By default, Document editor container has built-in properties pane which contains options for formatting text, table, image and header and footer. You can use [showPropertiesPane](#) API in [DocumentEditorContainer](#) to hide the properties pane.

The following example code illustrates how to hide the properties pane.

```
`typescript
import { Component, OnInit, ViewChild } from '@angular/core';
import {
  ToolbarService,
  DocumentEditorContainerComponent,
} from '@syncfusion/ej2-angular-documenteditor';
@Component({
  selector: 'app-root',
  // specifies the template string for the DocumentEditorContainer component
  template: <ejs-documenteditorcontainer #documenteditor_default
    serviceUrl="https://ej2services.syncfusion.com/production/web-
    services/api/documenteditor/" height="600px" style="display:block" [enableToolbar]=true
    [showPropertiesPane]=false (selectionChange)="selectionChanges()"> </ejs-
    documenteditorcontainer>,
  providers: [ToolbarService],
})
export class AppComponent implements OnInit {
  @ViewChild('documenteditor_default')
  public container: DocumentEditorContainerComponent;
  ngOnInit(): void {}
}
```

Note: Positioning and customizing the properties pane in Document editor container is not possible. Instead, you can hide the exiting properties pane and create your own pane using public API's.

Hide the toolbar

You can use [enableToolbar](#) API in [DocumentEditorContainer](#) to hide the existing toolbar.

The following example code illustrates how to hide the existing toolbar.

```
`typescript
```

```
import { Component, OnInit, ViewChild } from '@angular/core';
import {
  ToolbarService,
  DocumentEditorContainerComponent,
} from '@syncfusion/ej2-angular-documenteditor';
@Component({
  selector: 'app-root',
  // specifies the template string for the DocumentEditorContainer component
  template: <ejs-documenteditorcontainer #documenteditor_default
  serviceUrl="https://ej2services.syncfusion.com/production/web-
  services/api/documenteditor/" height="600px" style="display:block" [enableToolbar]=false
  (selectionChange)="selectionChanges()"> </ejs-documenteditorcontainer>
})
export class AppComponent implements OnInit {
  @ViewChild('documenteditor_default')
  public container: DocumentEditorContainerComponent;
  ngOnInit(): void {}
}
```

See Also

- [How to customize the toolbar](#)

Insert text or image in table programmatically in Angular Document editor component
Using Document editor API's, you can insert [text](#) or [image](#) in [table](#) programmatically based on your requirement.

Use [selection](#) API's to navigate between rows and cells.

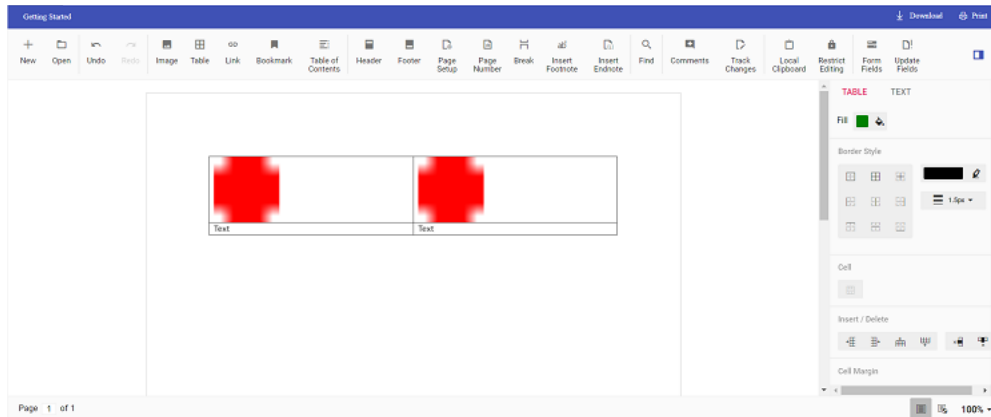
The following example illustrates how to create 2*2 table and then add text and image programmatically.

```
`typescript
import { Component, OnInit, ViewChild } from '@angular/core';
import {
  ToolbarService,
  DocumentEditorContainerComponent,
} from '@syncfusion/ej2-angular-documenteditor';
@Component({
```

```
selector: 'app-root',
// specifies the template string for the DocumentEditorContainer component
template: <ejs-documenteditorcontainer #documenteditor_default
serviceUrl="https://ej2services.syncfusion.com/production/web-
services/api/documenteditor/" height="600px" style="display:block" [enableToolbar]=true
(created)="onCreated()"> </ejs-documenteditorcontainer>,
providers: [ToolbarService],
})
export class AppComponent implements OnInit {
@ViewChild('documenteditor_default')
public container: DocumentEditorContainerComponent;
ngOnInit(): void {}
onCreated() {
// To insert the table in cursor position
this.container.documentEditor.editor.insertTable(2, 2);
// To insert the image at table first cell
this.container.documentEditor.editor.insertImage(
'data:image/png;base64,iVBORw0KGgoAAAANSUgAAAAUAAAAFCAYAAACNbybIAAAAHIEQVQI12P4
//8/w38GIAXDIBKE0DHxgljNBAAO9TXL0Y4OHwAAAABJRU5ErkJggg=='
);
// To move the cursor to next cell
this.moveCursorToNextCell();
// To insert the image at table second cell
this.container.documentEditor.editor.insertImage(
'data:image/png;base64,iVBORw0KGgoAAAANSUgAAAAUAAAAFCAYAAACNbybIAAAAHIEQVQI12P4
//8/w38GIAXDIBKE0DHxgljNBAAO9TXL0Y4OHwAAAABJRU5ErkJggg=='
);
// To move the cursor to next row
this.moveCursorToNextRow();
// To insert text in cursor position
this.container.documentEditor.editor.insertText('Text');
// To move the cursor to next cell
this.moveCursorToNextCell();
// To insert text in cursor position
```

```
this.container.documentEditor.editor.insertText('Text');
}
moveCursorToNextCell() {
// To get current selection start offset
var startOffset = container.documentEditor.selection.startOffset;
var offSet = startOffset.split(';');
// Increasing cell index to consider next cell
var cellIndex = parseInt(offSet[3]) + 1;
offSet[3] = cellIndex.toString();
// Changing start offset
startOffset = offSet.join(';');
// Navigating selection using select method
this.container.documentEditor.selection.select(startOffset, startOffset);
}
moveCursorToNextRow() {
// To get current selection start offset
var startOffset = container.documentEditor.selection.startOffset;
var offSet = startOffset.split(';');
// Increasing row index to consider next row
var rowIndex = parseInt(offSet[2]) + 1;
offSet[2] = rowIndex.toString();
var cellIndex = 0;
offSet[3] = cellIndex.toString();
// Changing start offset
startOffset = startOffset = offSet.join(';');
// Navigating selection using select method
this.container.documentEditor.selection.select(startOffset, startOffset);
}
}
`
```

The output will be like below.



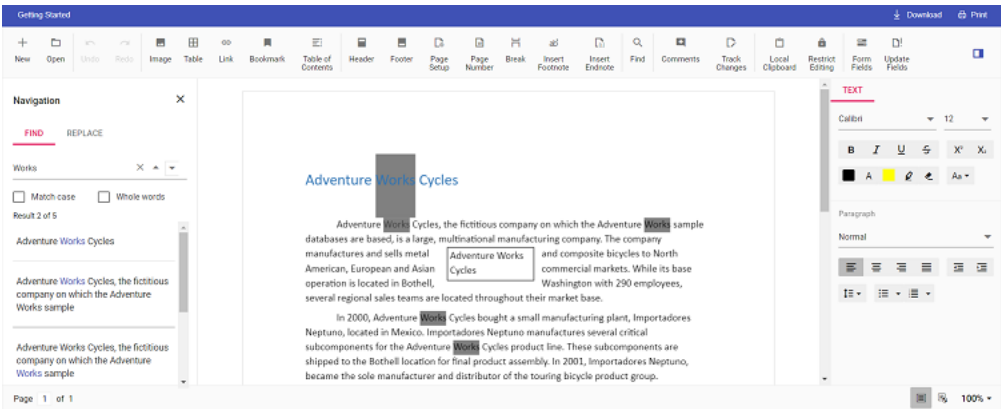
Change the default search highlight color in Angular Document editor component. Document editor provides an options to change the default search highlight color using [searchHighlightColor](#) in Document editor settings. The highlight color which is given in [documentEditorSettings](#) will be highlighted on the searched text. By default, search highlight color is yellow.

Similarly, you can use [documentEditorSettings](#) property for DocumentEditor also.

The following example code illustrates how to change the default search highlight color.

```
`typescript
import { Component, OnInit } from '@angular/core';
import { ToolbarService } from '@syncfusion/ej2-angular-documenteditor';
@Component({
  selector: 'app-root',
  // specifies the template string for the DocumentEditorContainer component
  template: <ejs-documenteditorcontainer
    serviceUrl="https://ej2services.syncfusion.com/production/web-
    services/api/documenteditor/" height="600px" style="display:block"
    [documentEditorSettings]= "searchHighlightColor" [enableToolbar]=true> </ejs-
    documenteditorcontainer>,
  providers: [ToolbarService],
})
export class AppComponent implements OnInit {
  // Add required color to change the default search highlight color
  public searchHighlightColor = { searchHighlightColor: 'Grey' };
  ngOnInit(): void {}
}
```

Output will be like below:



How to optimize the SFDT file

Starting from version v21.1.x, the SFDT file generated in Word Processor component is optimized by default to reduce the file size. All static keys are minified, and the final JSON string is compressed. This helps reduce the SFDT file size relative to a DOCX file and provides the following benefits,

- File transfer between client and server through the internet gets faster.
- The new optimized SFDT files require less storage space than the old SFDT files.

Hence, the optimized SFDT file can't be directly manipulated as JSON string.

This feature comes with a public API to switch between the old and new optimized SFDT format, allowing backward compatibility.

As a backward compatibility to create older format SFDT files, refer the following code changes,

Client/Server	Old Code	New Code from v21.1.x
Client-side	@Component({ template: })	@Component({ template: })export class AppComponent { public settings: DocumentEditorSettingsModel = { optimizeSfdt: false };}
Server-side C#	WordDocument sfdtDocument = WordDocument.Load(stream, formatType);string sfdt = Newtonsoft.Json.JsonConvert.SerializeObject(sfdtDocument);	WordDocument sfdtDocument = WordDocument.Load(stream, formatType);sfdtDocument.OptimizeSfdt = false;string sfdt = Newtonsoft.Json.JsonConvert.SerializeObject(sfdtDocument);
Server-side Java	String sfdtDocument = WordProcessorHelper.load(stream, formatType);	String sfdtDocument = WordProcessorHelper.load(stream, formatType, false);

To convert from older format SFDT from a new optimized SFDT file, refer the following code example,

Client/Server	Code example
---------------	--------------

Client-side	@Component({ template: `})export class AppComponent { public settings: DocumentEditorSettingsModel = { optimizeSfdt: false };}
Server-side C#	using(Syncfusion.DocIO.DLS.WordDocument docIODocument = WordDocument.Save(optimizedSfdt)) { sfdtDocument = WordDocument.Load(docIODocument); sfdtDocument.OptimizeSfdt = false; string oldSfdt = JsonSerializer.Serialize(sfdtDocument);}
Server-side Java	WordDocument docIODocument = WordProcessorHelper.save(optimizedSfdt);String oldSfdt = WordProcessorHelper.load(docIODocument, false);

How to disable auto focus in Syncfusion Angular Document Editor component

Document Editor gets focused automatically when the page loads. If you want the Document editor not to be focused automatically it can be customized.

The following example illustrates to disable the auto focus in DocumentEditorContainer.

```
`typescript
<ejs-documenteditorcontainer [enableAutoFocus]=false></ejs-documenteditorcontainer>
`
```

Note: Default value of [enableAutoFocus](#) property is true.

The following example illustrates to disable the auto focus in DocumentEditor.

```
`typescript
<ejs-documenteditor [enableAutoFocus]=false></ejs-documenteditor>
`
```

Note: Default value of [enableAutoFocus](#) property is true.

How to disable drag and drop in document editor in Angular Document editor component

Document Editor provides support to drag and drop contents within the component and it can be customized(enable and disable) using [allowDragAndDrop](#) property in Document editor settings.

The following example illustrates to customize the drag and drop option.

```
`typescript
@Component({
  template: `<ejs-documenteditorcontainer
    serviceUrl="https://ej2services.syncfusion.com/production/web-
    services/api/documenteditor/" height="600px" [enableToolbar]=true
    [documentEditorSettings]="settings"> </ejs-documenteditorcontainer>`,
})
export class AppComponent{
  public settings: DocumentEditorSettingsModel = { allowDragAndDrop : false };
}
```

Note: Default value of [allowDragAndDrop](#) property is true.

The following example illustrates to disable the drag and drop option in DocumentEditor.

```
`typescript
@Component({
  template: <ejs-documenteditor #document_editor height="330px" [enablePrint]=true
[documentEditorSettings]="settings"></ejs-documenteditor>,
})
export class AppComponent{
  public settings: DocumentEditorSettingsModel = { allowDragAndDrop : false };
}
```

Note: Default value of [allowDragAndDrop](#) property is true.

Enable ruler

How to enable ruler in Document Editor component

Using ruler we can refer to setting specific margins, tab stops, or indentations within a document to ensure consistent formatting in Document Editor.

The following example illustrates how to enable ruler in Document Editor

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { DocumentEditorAllModule } from '@syncfusion/ej2-angular-documenteditor'
import { Component, ViewEncapsulation, ViewChild, OnInit } from '@angular/core';
import { DocumentEditorComponent } from '@syncfusion/ej2-angular-documenteditor';
@Component({
  imports: [
    ButtonModule,
    DocumentEditorAllModule
  ],
  standalone: true,
  selector: "app-container",
  template: `<button
id='container_ruler_button' (click)="onClick(this)">Show/Hide Ruler</button>
<ejs-documenteditorcontainer #documenteditor
serviceUrl="https://ej2services.syncfusion.com/production/web-services/api/documenteditor/" height="600px" isReadOnly: false
style="width:100%;display:block" [enableToolbar]=true
[documentEditorSettings]= "enableRuler"> </ejs-documenteditorcontainer>`,
  encapsulation: ViewEncapsulation.None,
```

```

        providers: []
    })
    export class AppComponent implements OnInit {
        @ViewChild("documenteditor")
        public documentEditor: DocumentEditorComponent;
        ngOnInit(): void {
            this.documentEditor.enableAllModules();
        }
        public enableRuler = { showRuler: true };
        onClick(args: any):void {
            this.documentEditor.documentEditorSettings.showRuler =
!this.documentEditor.documentEditorSettings.showRuler;
        }
    }
}

```

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 enable ruler in Document Editor Container component

Using ruler we can refer to setting specific margins, tab stops, or indentations within a document to ensure consistent formatting in Document Editor Container.

The following example illustrates how to enable ruler in Document Editor Container.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { DocumentEditorAllModule } from '@syncfusion/ej2-angular-documenteditor'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import { ToolbarService, DocumentEditorContainerComponent } from
'@syncfusion/ej2-angular-documenteditor';
@Component({
    imports: [

        ButtonModule,
        DocumentEditorAllModule
    ],
    standalone: true,
    selector: 'app-container',
    template: `<button id='export' (click)="onClick(this)">Show/Hide
Ruler</button>
    <ejs-documenteditorcontainer #documenteditor default
serviceUrl="https://ej2services.syncfusion.com/production/web-
services/api/documenteditor/" height="600px"
style="width:100%;display:block" [enableToolbar]=true
[documentEditorSettings]= "enableRuler"> </ejs-documenteditorcontainer>`,
    encapsulation: ViewEncapsulation.None,
    providers: [ToolbarService]
})

```

```
export class AppComponent {
  @ViewChild('documenteditor_default')
  public container: DocumentEditorContainerComponent;
  public enableRuler = { showRuler: true };
  onClick():void {
    this.container.documentEditorSettings.showRuler =
    !this.container.documentEditorSettings.showRuler;
  }
}
```

MAIN.TS

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

Customize color picker in Angular Document editor component

Document editor provides an options to customize the color picker using [colorPickerSettings](#) in the document editor settings. The color picker offers customization options for default appearance, by allowing selection between Picker or Palette mode, for font and border colors.

Similarly, you can use [documentEditorSettings](#) property for DocumentEditor also.

The following example code illustrates how to customize the color picker in the document editor container.

`typescript

```
import { Component, OnInit } from '@angular/core';
import { ToolbarService } from '@syncfusion/ej2-angular-documenteditor';

@Component({
  selector: 'app-root',
  // specifies the template string for the DocumentEditorContainer component
  template: <ejs-documenteditorcontainer
serviceUrl="https://ej2services.syncfusion.com/production/web-
services/api/documenteditor/" height="600px" style="display:block"
[documentEditorSettings]= "colorPickerSettings" [enableToolbar]=true> </ejs-
documenteditorcontainer>,
  providers: [ToolbarService]
})
export class AppComponent implements OnInit {
  public colorPickerSettings = {colorPickerSettings: { mode: 'Palette', modeSwitcher: true, showButtons:
true }};
  ngOnInit(): void {
  }
}
```

```
}
```

```
,
```

```
| Property | Behaviour |
```

```
|---|---|
```

```
| columns | It is used to render the ColorPicker palette with specified columns. Defaults to 10 |
```

```
| disabled | It is used to enable / disable ColorPicker component. If it is disabled the ColorPicker popup won't open. Defaults to false |
```

```
| mode | It is used to render the ColorPicker with the specified mode. Defaults to 'Picker' |
```

```
| modeSwitcher | It is used to show / hide the mode switcher button of ColorPicker component. Defaults to true |
```

```
| showButtons | It is used to show / hide the control buttons (apply / cancel) of ColorPicker component. Defaults to true |
```

DropDownButton

Getting started with Angular Drop down button component

This section explains how to create a simple DropDownButton and demonstrate the basic usage of the DropDownButton component in an Angular environment.

Dependencies

The list of dependencies required to use the DropDownButton component in your application is given below:

```
`typescript
```

```
|-- @syncfusion/ej2-angular-splitbuttons
```

```
|-- @syncfusion/ej2-angular-base
```

```
|-- @syncfusion/ej2-base
```

```
|-- @syncfusion/ej2-splitbuttons
```

```
|-- @syncfusion/ej2-popups
```

```
|-- @syncfusion/ej2-buttons
```

```
,
```

Setup Angular environment

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

```
,
```

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

```
,
```

Create an Angular application

Start a new Angular application using below Angular CLI command.

```
,
```

```
ng new my-app
```

```
cd my-app
```

```
,
```

Installing Syncfusion DropDownButton 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-splitbuttons](#) package to the application.

```
`bash
```

```
npm install @syncfusion/ej2-angular-splitbuttons --save
```

```
,
```

Angular compatibility compiled package(ngcc)

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

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

```
`bash
```

```
npm install @syncfusion/ej2-angular-splitbuttons@ngcc --save
```

```
,
```

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

```
`bash
```

```
@syncfusion/ej2-angular-splitbuttons:"20.2.38-ngcc"
```

```
,
```

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

Adding DropDownButton module

Import DropDownButton module into Angular application(`app.module.ts`) from the package

`@syncfusion/ej2-angular-splitbuttons`.

```
`typescript
```

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
// Imported Syncfusion dropdown button module from splitbuttons package.
import { DropDownButtonModule } from '@syncfusion/ej2-angular-splitbuttons';
import { AppComponent } from './app.component';
@NgModule({
  imports: [ BrowserModule, DropDownButtonModule ], // Registering EJ2 Dropdownbutton Module.
  declarations: [ AppComponent ],
  bootstrap: [ AppComponent ]
})
export class AppModule { }
```

[Adding Syncfusion DropDownButton component](#)

Modify the template in `app.component.ts` file to render the DropDownButton component.

```
`typescript
import { Component } from '@angular/core';
import { ItemModel, MenuEventArgs } from '@syncfusion/ej2-angular-splitbuttons';
@Component({
  selector: 'app-root',
  template: `<!-- To render DropDownButton. -->
<button ej2-dropdownbutton [items]='items' content='Clipboard'></button>`
})
export class AppComponent {
  // Initialize action items.
  public items: ItemModel[] = [
    {
      text: 'Cut'
    },
    {
      text: 'Copy'
    },
    {
      text: 'Paste'
    }
  ]
}
```



```

});
}

```

Adding CSS reference

Add DropDownButton component's styles as given below in `style.css`.

```

`css

@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css';

```

Running the application

Run the application in the browser using the following command:

```

ng serve

```

The following example shows a basic Button component.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DropDownButtonModule } from '@syncfusion/ej2-angular-splitbuttons'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component } from '@angular/core';
import { ItemModel } from '@syncfusion/ej2-angular-splitbuttons';
@Component({
  imports: [

    DropDownButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<div class="e-section-control">
    <!-- To render DropDownButton. -->
    <button ej2-dropdownbutton [items]='items'
content='Clipboard'></button></div>`
})
export class AppComponent {
  // Initialize action items.
  public items: ItemModel[] = [
    {
      text: 'Cut'
    },
    {
      text: 'Copy'
    }
  ]
}

```

```

    },
    {
        text: 'Paste'
    }
];
}

```

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 Drop down button component

DropDownButton icons

DropDownButton can have an icon to provide the visual representation of the action. To place the icon on a DropdownButton, set the [iconCss](#) property to e-icons with the required icon CSS. By default, the icon is positioned to the left side of the DropdownButton. You can customize the icon's position using the [iconPosition](#) property.

In the following example, the DropdownButton with default iconPosition and iconPosition as **Top** is showcased.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { DropDownButtonModule } from '@syncfusion/ej2-angular-splitbuttons';
import { enableRipple } from '@syncfusion/ej2-base';
import { Component } from '@angular/core';
import { ItemModel } from '@syncfusion/ej2-angular-splitbuttons';
@Component({
    imports: [
        DropDownButtonModule
    ],
    standalone: true,
    selector: 'app-root',
    template: `<div class="e-section-control">
        <!-- To render DropDownButton. -->
        <button ejs-dropdownbutton [items]='items' content='Message'
iconCss='ddb-icons e-message'></button>
        <!-- To render DropDownButton with iconposition as 'top'. -->
        <button ejs-dropdownbutton [items]='items' content='Message'
iconCss='ddb-icons e-message' iconPosition='Top'></button>
    </div>`
})
export class AppComponent {
    // Initialize action items.
    public items: ItemModel[] = [
        {
            text: 'Edit'
        },
        {

```

```

        text: 'Delete'
      },
      {
        text: 'Mark as Read'
      },
      {
        text: 'Like Message'
      }
    ]];
  }

```

MAIN.TS

```

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

```

Icon only DropDownButton

Icon only DropDownButton can be achieved by using [iconCss](#) property and to hide drop down arrow `e-caret-hide` class is added using [cssClass](#) property.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { DropDownButtonModule } from '@syncfusion/ej2-angular-splitbuttons';
import { enableRipple } from '@syncfusion/ej2-base';
import { Component } from '@angular/core';
import { ItemModel } from '@syncfusion/ej2-angular-splitbuttons';
@Component({
  imports: [
    DropDownButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<div class="e-section-control">
    <!-- To render DropDownButton without down arrow. -->
    <button ej2-dropdownbutton [items]='items' iconCss='e-icons e-
menu' cssClass='e-caret-hide'></button></div>`
})
export class AppComponent {
  // Initialize action items.
  public items: ItemModel[] = [
    {
      text: 'New tab'
    },
    {
      text: 'New window'
    },
    {
      text: 'New incognito window'
    },
    {
      separator: true
    }
  ]
}

```

```

    },
    {
      text: 'Print'
    },
    {
      text: 'Cast'
    },
    {
      text: 'Find'
    }
  ]];
}

```

MAIN.TS

```

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

```

The Essential JS 2 provides a set of icons that can be loaded by applying **e-icons** class name to the element.

You can also use third party icons on the DropDownButton using the [iconCss](#) property.

DropDownButton with sprite image

Sprite images can be loaded in DropDownButton instead of font icons using [iconCss](#) property.

In this following example, **e-image** class is added with background url of the sprite image along with X and Y positions. The **width** and **height** of the element set as **32px**.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { DropDownButtonModule } from '@syncfusion/ej2-angular-splitbuttons';
import { enableRipple } from '@syncfusion/ej2-base';
import { Component } from '@angular/core';
import { ItemModel } from '@syncfusion/ej2-angular-splitbuttons';
@Component({
  imports: [
    DropDownButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<div class="e-section-control">
    <!-- To render DropDownButton with sprite image. -->
    <button ej2-dropdownbutton [items]='items' iconCss='e-image'
cssClass='e-caret-hide'></button></div>`
})
export class AppComponent {
  // Initialize action items.
  public items: ItemModel[] = [
    {
      text: 'Display Settings'
    },
  ],

```

```

    {
      text: 'System Settings'
    },
    {
      text: 'Additional Settings'
    }
  ];
}

```

MAIN.TS

```

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

```

The Essential JS 2 provides a set of icons that can be loaded by applying **e-icons** class name to the element.

You can also use third party icons on the DropDownButton using the [iconCss](#) property.

Vertical button

Vertical button in DropDownButton can be achieved by adding **e-vertical** class using [cssClass](#) property.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { DropDownButtonModule } from '@syncfusion/ej2-angular-splitbuttons';
import { enableRipple } from '@syncfusion/ej2-base';
import { Component } from '@angular/core';
import { ItemModel } from '@syncfusion/ej2-angular-splitbuttons';
@Component({
  imports: [
    DropDownButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<div class="e-section-control">
    <!-- To render Vertical DropDownButton. -->
    <button ej2-dropdownbutton [items]='items' content='Message'
    iconCss='ddb-icons e-message' cssClass='e-vertical'
    iconPosition='Top'></button></div>`
})
export class AppComponent {
  // Initialize action items.
  public items: ItemModel[] = [
    {
      text: 'Edit'
    },
    {
      text: 'Delete'
    }
  ]
}

```

```

        text: 'Mark as Read'
    },
    {
        text: 'Like Message'
    }
];
}

```

MAIN.TS

```

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

```

See Also

- [Dropdown popup with icons](#)
- [Customized icon size](#)

Popup items in Angular Drop down button component

Icons

The popup action item have an icon or image to provide visual representation of the action. To place the icon on a popup item, set the [iconCss](#) property to `e-icons` with the required icon CSS. By default, the icon is positioned to the left side of the popup action item.

In the following sample, the icons for edit, delete, mark as read and like message menu items are added using the `iconCss` property.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { DropDownButtonModule } from '@syncfusion/ej2-angular-splitbuttons';
import { enableRipple } from '@syncfusion/ej2-base';
import { Component } from '@angular/core';
import { ItemModel } from '@syncfusion/ej2-angular-splitbuttons';
@Component({
    imports: [
        DropDownButtonModule
    ],
    standalone: true,
    selector: 'app-root',
    template: `<div class="e-section-control">
        <!-- To render DropDownButton. -->
        <button ej2-dropdownbutton [items]='items' content='Message'
            iconCss='ddb-icons e-message'></button></div>`
})
export class AppComponent {
    // Initialize action items.
    public items: ItemModel[] = [
        {
            text: 'Edit',

```

```

        iconCss: 'ddb-icons e-edit'
      },
      {
        text: 'Delete',
        iconCss: 'ddb-icons e-delete'
      },
      {
        text: 'Mark As Read',
        iconCss: 'ddb-icons e-read'
      },
      {
        text: 'Like Message',
        iconCss: 'ddb-icons e-like'
      }
    ]];
  }
}

```

MAIN.TS

```

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

```

Navigations

Actions in DropDownButton can be used to navigate to the other web page when action item is clicked. This can be achieved by providing link to the action item using `url` property.

In the following sample, navigation URL for Flipkart, Amazon, and Snapdeal action items are added using the `url` property:

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { DropDownButtonModule } from '@syncfusion/ej2-angular-splitbuttons';
import { enableRipple } from '@syncfusion/ej2-base';
import { Component } from '@angular/core';
import { ItemModel, MenuEventArgs } from '@syncfusion/ej2-angular-splitbuttons';
@Component({
  imports: [
    DropDownButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<div class="e-section-control">
    <!-- To render DropDownButton. -->
    <button ejs-dropdownbutton [items]='items' content='Shop By'
    iconCss='e-cart-icon e-shopping'
    (beforeItemRender)='itemBeforeEvent($event)'></button></div>`
})
export class AppComponent {
  // Initialize action items.
  public items: ItemModel[] = [

```

```

    {
      text: 'Flipkart',
      iconCss: 'e-cart-icon e-link',
      url: 'https://www.google.co.in/search?q=flipkart'
    },
    {
      text: 'Amazon',
      iconCss: 'e-cart-icon e-link',
      url: 'https://www.google.co.in/search?q=amazon'
    },
    {
      text: 'Snapdeal',
      iconCss: 'e-cart-icon e-link',
      url: 'https://www.google.co.in/search?q=snapdeal'
    }
  ];
  // To open the url in the blank page.
  public itemBeforeEvent (args: MenuEventArgs) {
    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));

```

Template

Item templating

Popup items can be customized using the [beforeItemRender](#) event. The item render event triggers while rendering each popup action item. The event argument will be used to identify the action item and customize based on the requirement.

The following popup template is customized using [beforeItemRender](#) event by appending `span` and `div` element on each `li` rendering:

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DropDownButtonModule } from '@syncfusion/ej2-angular-splitbuttons'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component } from '@angular/core';
import { ItemModel, MenuEventArgs } from '@syncfusion/ej2-angular-splitbuttons';
@Component({
  imports: [
    DropDownButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<div class="e-section-control">

```



```

        <!-- To render DropDownButton. -->
        <button ejs-dropdownbutton [items]='items' content='Paste'
iconCss='e-ddb-icons e-paste' iconPosition='Top' cssClass='e-vertical'
(beforeItemRender)='itemBeforeEvent($event)'></button></div>`
    })
    export class AppComponent {
        // Initialize action items.
        public items: ItemModel[] = [
            {
                text: 'Edit'
            },
            {
                text: 'Cut'
            }
        ];
        public itemBeforeEvent (args: MenuEventArgs) {
            // To append span and div element in each li rendering.
            if (args.item.text === 'Edit') {
                args.element.innerHTML = '<span></span><div><b>Paste
Text</b><div>Provides option to paste only the<br>selected
text.</div></div>';
                args.element.style.height = '80px';
                let span: Element = args.element.children[0];
                span.setAttribute('class', 'e-cm-icons e-pastetext e-align');
                let div: Element = args.element.children[1];
                div.setAttribute('class', 'e-div-align');
            } else {
                args.element.innerHTML = '<span></span><div><b>Paste
Special</b><div>Provides options to paste formulas,<br> values, comments,
validations etc...</div></div>';
                args.element.style.height = '80px';
                let span: Element = args.element.children[0];
                span.setAttribute('class', 'e-cm-icons e-pastespecial e-align');
                let div: Element = args.element.children[1];
                div.setAttribute('class', 'e-div-align');
            }
        }
    }
}

```

MAIN.TS

```

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

```

Popup templating

The whole popup can be customized as per the requirement. In the following example, the popup can be customized by handling it in [target](#) property.

In the following sample, the whole popup item is customized as table template by giving `div` as target and it can be achieved using `target` property.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'

```

```

import { BrowserModule } from '@angular/platform-browser'
import { DropDownButtonModule } from '@syncfusion/ej2-angular-splitbuttons'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component } from '@angular/core';
@Component({
  imports: [

    DropDownButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<div class="e-section-control">
    <!-- Target element to DropDownButton . -->
    <div id="target" style='border: 1px solid #999;'>
    <!-- To create table. -->
    <table>
      <caption style='height: 18px; background-color:
#e0e0e0;'><b>Insert Table</b></caption>
      <tr class='e-row'>
        <td class='e-data'></td>
        <td class='e-data'></td>
        <td class='e-data'></td>
        <td class='e-data'></td>
        <td class='e-data'></td>
        <td class='e-data'></td>
      </tr>
      <tr class='e-row'>
        <td class='e-data'></td>
        <td class='e-data'></td>
        <td class='e-data'></td>
        <td class='e-data'></td>
        <td class='e-data'></td>
        <td class='e-data'></td>
      </tr>
      <tr class='e-row'>
        <td class='e-data'></td>
        <td class='e-data'></td>
        <td class='e-data'></td>
        <td class='e-data'></td>
        <td class='e-data'></td>
        <td class='e-data'></td>
      </tr>
      <tr class='e-row'>
        <td class='e-data'></td>
        <td class='e-data'></td>
        <td class='e-data'></td>
        <td class='e-data'></td>
        <td class='e-data'></td>
        <td class='e-data'></td>
      </tr>
      <tr class='e-row'>
        <td class='e-data'></td>
        <td class='e-data'></td>
        <td class='e-data'></td>
        <td class='e-data'></td>
        <td class='e-data'></td>
        <td class='e-data'></td>
      </tr>
    </table>
  `;
})

```

```

        </tr>
        <tr class='e-row'>
            <td class='e-data'></td>
            <td class='e-data'></td>
            <td class='e-data'></td>
            <td class='e-data'></td>
            <td class='e-data'></td>
            <td class='e-data'></td>
        </tr>
    </table>
</div>
<!-- To render DropDownButton. -->
<button ejs-dropdownbutton target='#target' content='Table'
iconCss='e-icons e-table' iconPosition='Top' cssClass='e-vertical'></button>
</div>`
    ))
    export class AppComponent {
    }

```

MAIN.TS

```

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

```

Separator

The Separators are the horizontal lines that are used to separate the popup items. You cannot select the separators.

You can enable separators to group the popup items using the `separator` property.

In the following sample, cut, copy, and paste popup items are grouped using the separator property:

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DropDownButtonModule } from '@syncfusion/ej2-angular-splitbuttons'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component } from '@angular/core';
import { ItemModel } from '@syncfusion/ej2-angular-splitbuttons';
@Component({
  imports: [

    DropDownButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<div class="e-section-control">
    <!-- To render DropDownButton. -->
    <button ejs-dropdownbutton [items]='items' content='Clipboard'
iconCss='e-icons e-edit'></button></div>`
})
export class AppComponent {
  // Initialize action items.

```

```
public items: ItemModel[] = [
  {
    text: 'Cut',
    iconCss: 'e-db-icons e-cut'
  },
  {
    text: 'Copy',
    iconCss: 'e-icons e-copy'
  },
  {
    text: 'Paste',
    iconCss: 'e-db-icons e-paste'
  },
  {
    separator: true
  },
  {
    text: 'Font',
    iconCss: 'e-db-icons e-font'
  },
  {
    text: 'Paragraph',
    iconCss: 'e-db-icons e-paragraph'
  }
];
```

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

- [Integration with ListView component](#)

Accessibility in Angular Drop down button component

The Drop down button component followed the accessibility guidelines and standards, including [ADA](#), [Section 508](#), [WCAG 2.2](#) standards, and [WCAG roles](#) that are commonly used to evaluate accessibility.

The accessibility compliance for the Drop down button component is outlined below.

| Accessibility Criteria | Compatibility |

| -- | -- |

| [WCAG 2.2](#) Support | |

| [Section 508](#) Support | |

| Screen Reader Support | |

| Right-To-Left Support | |

| Color Contrast | |

| Mobile Device Support | |

| Keyboard Navigation Support | |

| [Accessibility Checker](#) Validation | |

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

```
<style>
.post .post-content img {
display: inline-block;
margin: 0.5em 0;
}
</style>

<div> - All
features of the component meet the requirement.</div>

<div> - Some features of the component do not meet the requirement.</div>

<div> - The component does not meet the requirement.</div>
```

WAI-ARIA attributes

The Drop down button component followed the [WAI-ARIA] patterns to meet the accessibility. The following ARIA attributes are used in the Drop down button component:

| Attributes | Purpose |

| --- | --- |

| **role** | Indicates the Drop down button component as **button**, Drop down popup as **menu**, and the dropdown popup action items as **menuitem**. |

| **aria-haspopup** | Indicates the availability of the popup element. |

| **aria-expanded** | Indicates whether the popup can be expanded or collapsed, as well as indicates whether its current state is expanded or collapsed. |

| **aria-owns** | Identifies an elements in order to define a visual, functional, or contextual parent/child relationship between DOM elements where the DOM hierarchy cannot be used to represent the relationship. |

| **aria-disabled** | Indicates that the element is perceivable but disabled, so it is not editable or otherwise operable. |

Keyboard interaction

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

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

| --- | --- |

| **Esc** | Closes the popup. |

| **Enter** | Opens the popup, or activates the highlighted item and closes the popup. |

| **Space** | Opens the popup. |

| **Up** | Navigates up or to the previous action item. |

| **Alt + Up Arrow** | Closes the popup. |

| **Alt + Down Arrow** | Opens the popup. |

Ensuring accessibility

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

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

See also

- [Accessibility in Syncfusion Angular components](#)

How To

Change caret icon in Angular Drop down button component

Dropdown arrow can be customized on popup open and close. It can be handled in

[beforeOpen](#) and [beforeClose](#) event.

In the following example, the up arrow is updated on popup close and down arrow is updated on popup open using **beforeOpen** and **beforeClose** event by adding and removing **e-caret-up** class.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DropDownButtonModule } from '@syncfusion/ej2-angular-splitbuttons'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component, ViewChild } from '@angular/core';
import { ItemModel, BeforeOpenCloseMenuEventArgs, DropDownButtonComponent }
from '@syncfusion/ej2-angular-splitbuttons';
@Component({
```

```

imports: [
    DropDownButtonModule
],
standalone: true,
selector: 'app-root',
template: `<div class="e-section-control">
    <!-- To render DropDownButton. -->
    <button ejs-dropdownbutton #dropdownbutton [items]='items'
content='Clipboard' (beforeOpen)='beforeOpen($event)'
(beforeClose)='beforeClose($event)'></button></div>`
})
export class AppComponent {
    @ViewChild('dropdownbutton')
    public dropdownbutton: DropDownButtonComponent | any;
    // Initialize action items.
    public items: ItemModel[] = [
        {
            text: 'Cut'
        },
        {
            text: 'Copy'
        },
        {
            text: 'Paste'
        }
    ];
    // To update up arrow with `e-caret-up` class.
    public beforeOpen (args: BeforeOpenCloseMenuEventArgs) {
        this.dropdownbutton.cssClass = 'e-caret-up';
    }
    // To remove `e-caret-up` class.
    public beforeClose (args: BeforeOpenCloseMenuEventArgs) {
        this.dropdownbutton.cssClass = '';
    }
}

```

MAIN.TS

```

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

```

Create dropdownbutton with rounded corner in Angular Drop down button component
 DropDownButton with rounded corner can be achieved by adding `border-radius` CSS property to button element.

In the following example, `e-round-corner` class is defined with `5px border-radius` property and added that class to button element using `cssClass` property.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DropDownButtonModule } from '@syncfusion/ej2-angular-splitbuttons'

```

```

import { enableRipple } from '@syncfusion/ej2-base'
import { Component } from '@angular/core';
import { ItemModel } from '@syncfusion/ej2-angular-splitbuttons';
@Component({
  imports: [

    DropDownButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<div class="e-section-control">
    <!-- To render DropDownButton with 'e-round-corner' class. -->
  >
    <button ejjs-dropdownbutton [items]='items' content='Clipboard'
cssClass='e-round-corner'></button></div>`
})
export class AppComponent {
  // Initialize action items.
  public items: ItemModel[] = [
    {
      text: 'Cut'
    },
    {
      text: 'Copy'
    },
    {
      text: 'Paste'
    }
  ];
}

```

MAIN.TS

```

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

```

Create right to left dropdownbutton in Angular Drop down button component

DropDownButton component has RTL support. This can be achieved by setting [enableRtl](#) as true.

The following example illustrates how to enable right-to-left support in DropDownButton component.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DropDownButtonModule } from '@syncfusion/ej2-angular-splitbuttons'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component } from '@angular/core';
import { ItemModel } from '@syncfusion/ej2-angular-splitbuttons';
@Component({
  imports: [

    DropDownButtonModule
  ],
  standalone: true,

```



```

    selector: 'app-root',
    template: `<div class="e-section-control">
        <!-- To render DropDownButton. -->
        <button ejs-dropdownbutton [items]='items' content='Message'
iconCss='ddb-icons e-message' enableRtl='true'></button>
        </div>`
  })
  export class AppComponent {
    // Initialize action items.
    public items: ItemModel[] = [
      {
        text: 'Edit'
      },
      {
        text: 'Delete'
      },
      {
        text: 'Mark as Read'
      },
      {
        text: 'Like Message'
      }
    ];
  }
}

```

MAIN.TS

```

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

```

Customize icon and width in Angular Drop down button component

Width of the DropDownButton can be customized by setting required width to the dropdown element.

The following UI can be achieved by setting [iconPosition](#) as **Top**, width as **85px**

and size of the font icon as **40px** by adding **e-custom** class.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DropDownButtonModule } from '@syncfusion/ej2-angular-splitbuttons'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component } from '@angular/core';
import { ItemModel } from '@syncfusion/ej2-angular-splitbuttons';
@Component({
  imports: [
    DropDownButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<div class="e-section-control">
    <!-- To render DropDownButton. -->

```

```

        <button ej-dropdownbutton [items]='items' content='Find &
Select' iconCss='e-icons e-search' iconPosition='Top' cssClass='e-
custom'></button></div>`
    })
    export class AppComponent {
        // Initialize action items.
        public items: ItemModel[] = [
            {
                text: 'Find'
            },
            {
                text: 'Replace'
            },
            {
                text: 'Go To'
            },
            {
                text: 'Go To Special'
            }
        ];
    }

```

MAIN.TS

```

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

```

Disable a dropdownbutton in Angular Drop down button component

DropDownButton component can be enabled/disabled by giving [disabled](#) property.

To disable DropDownButton component, the disabled property can be set as `true`.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DropDownButtonModule } from '@syncfusion/ej2-angular-splitbuttons'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component } from '@angular/core';
import { ItemModel } from '@syncfusion/ej2-angular-splitbuttons';
@Component({
    imports: [
        DropDownButtonModule
    ],
    standalone: true,
    selector: 'app-root',
    template: `<div class="e-section-control">
        <!-- To render DropDownButton. -->
        <button ej-dropdownbutton [items]='items' content='Message'
iconCss='ddb-icons e-message' disabled='true'></button>
        </div>`
})
export class AppComponent {
    // Initialize action items.

```

```

public items: ItemModel[] = [
  {
    text: 'Edit'
  },
  {
    text: 'Delete'
  },
  {
    text: 'Mark as Read'
  },
  {
    text: 'Like Message'
  }
];
}

```

MAIN.TS

```

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

```

Group popup items with listview component in Angular Drop down button component

Header in popup items is possible in DropdownButton by templating entire popup with ListView. Create ListView with id `listview` and provide it as a [target](#) for DropDownButton.

In the following example, ListView element is given as `target` to DropDownButton and header can be achieved by [groupBy](#) property.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { DropDownButtonModule } from '@syncfusion/ej2-angular-splitbuttons';
import { enableRipple } from '@syncfusion/ej2-base';
import { ListViewModule } from '@syncfusion/ej2-angular-lists';
import { Component } from '@angular/core';
import { ItemModel } from '@syncfusion/ej2-angular-splitbuttons';
@Component({
  imports: [
    ListViewModule,
    DropDownButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<div class="e-section-control">
    <!-- To render ListView. -->
    <ejs-listview id='listview' [dataSource]='data'
[fields]='field' showCheckBox='true'></ejs-listview>
    <!-- To render DropDownButton. -->
    <button ej2-dropdownbutton target='#listview' iconCss='e-icons
e-down' cssClass='e-caret-hide'></button>
  </div>`
})

```

```
export class AppComponent {  
  // Initialize action items.  
  public items: ItemModel[] = [  
    {  
      text: 'Cut'  
    },  
    {  
      text: 'Copy'  
    },  
    {  
      text: 'Paste'  
    }  
  ];  
  // Datasource for listview.  
  public data: Object = [  
    { class: 'data', text: 'Print', id: 'data1', category: 'Customize Quick  
Access Toolbar' },  
    { class: 'data', text: 'Save As', id: 'data2', category: 'Customize Quick  
Access Toolbar' },  
    { class: 'data', text: 'Update Folder', id: 'data3', category: 'Customize  
Quick Access Toolbar' },  
    { class: 'data', text: 'Reply', id: 'data4', category: 'Customize Quick  
Access Toolbar' }  
  ];  
  public field: Object = { text: 'text', 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));
```

Hide dropdown arrow in Angular Drop down button component

You can hide the dropdown arrow from the DropDownButton by adding class `e-caret-hide`

to DropDownButton element using [cssClass](#) property.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'  
import { BrowserModule } from '@angular/platform-browser'  
import { DropDownButtonModule } from '@syncfusion/ej2-angular-splitbuttons'  
import { enableRipple } from '@syncfusion/ej2-base'  
import { Component } from '@angular/core';  
import { ItemModel } from '@syncfusion/ej2-angular-splitbuttons';  
@Component({  
  imports: [  
    DropDownButtonModule  
  ],  
  standalone: true,  
  selector: 'app-root',  
  template: `<div class="e-section-control">  
    <!-- To render DropDownButton with cssClass. -->
```

```

        <button ej-dropdownbutton [items]='items' content='Clipboard'
cssClass='e-caret-hide'></button></div>`
    })
    export class AppComponent {
        // Initialize action items.
        public items: ItemModel[] = [
            {
                text: 'Cut'
            },
            {
                text: 'Copy'
            },
            {
                text: 'Paste'
            }
        ];
    }
}

```

MAIN.TS

```

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

```

Open a dialog on popup item click in Angular Drop down button component

This section explains about how to open a dialog on DropdownButton popup item click.

This can be achieved by handling dialog open in [select](#) event of the DropdownButton.

In the following example, Dialog will open while selecting **Other Folder...** item.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DropDownButtonModule } from '@syncfusion/ej2-angular-splitbuttons'
import { DialogModule } from '@syncfusion/ej2-angular-popups'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component, ViewChild } from '@angular/core';
import { ItemModel, MenuEventArgs, DropDownButtonComponent } from '@syncfusion/ej2-angular-splitbuttons';
import { DialogComponent } from '@syncfusion/ej2-angular-popups';
@Component({
    imports: [
        DialogModule,
        DropDownButtonModule
    ],
    standalone: true,
    selector: 'app-root',
    template: `<div class="e-section-control">
        <!-- To render Dialog. -->
        <ejs-dialog #dialog [buttons]='alertDlgButtons'
[visible]='false' content='Move Items To "Web Team"' width='250px'
[position]='position'>
            </ejs-dialog>
    `
})

```

```

        <!-- To render DropDownButton. -->
        <button ejs-dropdownbutton #dropdownbutton [items]='items'
content='Move' iconCss='ddb-icons e-folder' cssClass='e-vertical'
iconPosition='Top' (select)='select($event)'></button>
        </div>`
    })
    export class AppComponent {
        @ViewChild('dialog')
        public alertDialog?: DialogComponent;
        @ViewChild('dropdownbutton')
        public dropdownbutton?: DropDownButtonComponent;
        public position: any = {X: 100, Y: 100};
        public alertDlgButtons: Object[] = [{
            buttonModel: {
                isPrimary: true,
                content: 'Submit',
                cssClass: 'e-flat',
            },
            click: function () {
                (this as any).hide();
            }
        }];
        // Initialize action items.
        public items: ItemModel[] = [
            {
                text: 'Archive'
            },
            {
                text: 'Inbox'
            },
            {
                text: 'HR Portal'
            },
            {
                separator: true
            },
            {
                text: 'Other Folder...'
            },
            {
                text: 'Copy to Folder'
            }
        ];
        // To open dialog on selecting `Other Folder...` item.
        public select (args: MenuEventArgs) {
            if (args.item.text === 'Other Folder...') {
                this.alertDialog!.show();
            }
        }
    }
}

```

MAIN.TS

```

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

```

Position popup open in Angular Drop down button component

Popup open position can be changed according to the requirement. Popup open position can be changed in [open](#) event by setting **top** and **left** for the popup element.

In the following example, the **top** position of the popup element is changed in **open** event.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DropDownButtonModule } from '@syncfusion/ej2-angular-splitbuttons'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component, ViewChild } from '@angular/core';
import { ItemModel, OpenCloseMenuEventArgs, DropDownButtonComponent } from '@syncfusion/ej2-angular-splitbuttons';
@Component({
  imports: [
    DropDownButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<div class="e-section-control">
    <!-- To render DropDownButton. -->
    <button ej2-dropdownbutton #dropdownbutton [items]='items'
content='Clipboard' (open)='onOpen($event)'></button></div>`
})
export class AppComponent {
  @ViewChild('dropdownbutton')
  public dropdownbutton?: DropDownButtonComponent;
  // Initialize action items.
  public items: ItemModel[] = [
    {
      text: 'Cut'
    },
    {
      text: 'Copy'
    },
    {
      text: 'Paste'
    }
  ];
  // To open popup in particular position.
  public onOpen (args: OpenCloseMenuEventArgs) {
    args.element.parentElement!.style.top =
    this.dropdownbutton!.element.getBoundingClientRect().top -
    args.element.parentElement!.offsetHeight + 'px';
  }
}
```

MAIN.TS

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

Underline a character in the item text in Angular Drop down button component

Underline a particular character in a text can be handled in [beforeItemRender](#) event by adding `<u>` tag in between the text and given as innerHTML in `li` rendering.

In the following example, **C** is underlined in the text **Copy**.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DropDownButtonModule } from '@syncfusion/ej2-angular-splitbuttons'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component } from '@angular/core';
import { ItemModel, MenuEventArgs } from '@syncfusion/ej2-angular-splitbuttons';
@Component({
  imports: [

    DropDownButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<div class="e-section-control">
    <!-- To render DropDownButton. -->
    <button ej2-dropdownbutton [items]='items' content='Clipboard'
(beforeItemRender)="itemRender($event)"></button></div>`
})
export class AppComponent {
  // Initialize action items.
  public items: ItemModel[] = [
    {
      text: 'Cut'
    },
    {
      text: 'Copy'
    },
    {
      text: 'Paste'
    }
  ];
  public itemRender(args: MenuEventArgs) {
    if (args.item.text === 'Copy') {
      //To underline a particular text.
      args.element.innerHTML = '<u>C</u>opy';
    }
  }
}
```

MAIN.TS

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


DropDownList

Getting started with Angular Drop down list component

This section briefly explains how to create a simple **DropDownList** component and configure its available functionalities in Angular.

Dependencies

The following list of dependencies are required to use the DropDownList 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

Angular provides the easiest way to set angular CLI projects using [Link to the Video](#) tool.

To get started quickly with angular DropDownList component using angular CLI, you can check the video below.

Install the CLI application globally to your machine.

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

Create a new application

```
`bash
ng new syncfusion-angular-dropdownlist
`,`
```

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-dropdownlist --style=scss
```

Navigate to the created project folder.

```
`bash
```

```
cd syncfusion-angular-dropdownlist
```

Installing Syncfusion DropDownList 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 DropDownList module

Import `DropDownList` 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 DropDownListModule for the DropDownList component
import { DropDownListModule } from '@syncfusion/ej2-angular-dropdowns';
import { AppComponent } from './app.component';
@NgModule({
  //declaration of ej2-angular-dropdowns module into NgModule
  imports: [ BrowserModule, DropDownListModule ],
  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 DropDownList component

Modify the template in `[src/app/app.component.ts]` file to render the DropDownList component. Add the Angular DropDownList by using `<ejs-dropdownlist>` 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 DropDownList component
template: <ejs-dropdownlist id='ddlelement'></ejs-dropdownlist>
})
export class AppComponent { }
```

Binding data source

After initialization, populate the DropDownList with data using the `dataSource` property. Here, an array of string values passed to DropDownList component.

```
`typescript
import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
  // specifies the template string for the DropDownList component
  template: <ejs-dropdownlist id='ddlelement' [dataSource]='data'></ejs-dropdownlist>
})
export class AppComponent {
  constructor() {
  }
  // defined the array of data
  public data: string[] = ['Snooker', 'Tennis', 'Cricket', 'Football', 'Rugby'];
}
```

Running the application

After completing the configuration required to render a basic DropDownList, 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 { DropDownListModule } from '@syncfusion/ej2-angular-dropdowns'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component } from '@angular/core';
@Component({
```

```
imports: [
    FormsModule, ReactiveFormsModule, DropDownListModule, ButtonModule
],
standalone: true,
selector: 'app-root',
// specifies the template string for the DropDownList component with
// dataSource
template: `<ejs-dropdownlist id='ddlelement' [dataSource]='data'
placeholder = 'Select a game'></ejs-dropdownlist>`
})
export class AppComponent {
    constructor() {
    }
    // defined the array of data
    public data: string[] = ['Snooker', 'Tennis', 'Cricket', 'Football',
'Rugby'];
}
```

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 DropDownList input element's width, and the height of the popup list has '300px'.

The height and width of the popup list can also be customized using the [popupHeight](#) and [popupWidth](#) property 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 { DropDownListModule } from '@syncfusion/ej2-angular-dropdowns'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component } from '@angular/core';
@Component({
    imports: [
        FormsModule, ReactiveFormsModule, DropDownListModule, ButtonModule
    ],
    standalone: true,
    selector: 'app-root',
    // specifies the template url path
    template: `<ejs-dropdownlist id='ddlelement' [dataSource]='data'
placeholder='Select a game' popupHeight='200px' popupWidth='250px' ></ejs-
dropdownlist>`
})
export class AppComponent {
    constructor() {
    }
}
```

```
// defined the array of data
public data: string[] = ['Badminton', 'Basketball', 'Cricket',
'Football', 'Golf', 'Hockey', 'Rugby', 'Snooker', 'Tennis'];
}
```

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 DropDownList, the **value** property supports two-way binding functionality. The following example demonstrates how to work the two-way binding functionality in DropDownList.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule, ReactiveFormsModule } from '@angular/forms'
import { DropDownListModule } from '@syncfusion/ej2-angular-dropdowns'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component } from '@angular/core';
@Component({
  imports: [
    FormsModule, ReactiveFormsModule, DropDownListModule, ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the DropDownList component and
  // input element for checking the two-way binding support using value
  property
  template: `
    <ejs-dropdownlist id='ddlelement' [dataSource]='data' [(value)]='value'
placeholder='Select a game'></ejs-dropdownlist>
    <div style='margin-top: 50px'>
      <input type="text" [(ngModel)]="value"
style="width:245px;height:25px" />
    </div>
  `
})
export class AppComponent {
  constructor() {
  }
  // defined the array of complex data
  public data: string[] = [ 'Badminton', 'Basketball', 'Cricket',
'Football' ];
  // set a value to pre-select
  public value: string = 'Badminton';
}
```

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](#)

Data binding in Angular Drop down list component

The DropDownList loads the data either from local data sources or remote data services using the [dataSource](#) property. It supports the data type of array or **DataManager**.

remote data services using the [dataSource](#) property. It supports the data type of array or **DataManager**.

The DropDownList also supports different kinds of data services such as OData, OData V4, and Web API, and data formats such as XML, JSON, and JSONP with the help of **DataManager**adaptors.

Fields	Type	Description
text	string	Specifies the display text of each list item.
value	number or string	Specifies the hidden data value mapped to each list item that should contain a unique value.
groupBy	string	Specifies the category under which the list item has to be grouped.
iconCss	string	Specifies the icon class of each list item.

When binding complex data to the DropDownList, fields should be mapped correctly. Otherwise, the selected item remains undefined.

Binding local data

Local data can be represented in two ways as described below.

1. Array of simple data

The DropDownList has support to load array of primitive data such as strings and numbers. Here, both value and text field act the same.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule, ReactiveFormsModule } from '@angular/forms'
import { DropDownListModule } from '@syncfusion/ej2-angular-dropdowns'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component } from '@angular/core';
@Component({
  imports: [
    FormsModule, ReactiveFormsModule, DropDownListModule, ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
```

```

    // specifies the template string for the DropDownList component with
    change event
    template: `<ejs-dropdownlist id='ddlelement' #samples [dataSource]='data'
    [placeholder]='placeholder'></ejs-dropdownlist>`
  })
  export class AppComponent {
    constructor() {
    }
    // defined the array of data
    public data: string[] = ['Badminton', 'Basketball', 'Cricket', 'Golf',
    'Hockey', 'Rugby'];
    // set placeholder text to DropDownList input element
    public placeholder: 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));

```

2. Array of JSON data

The DropDownList can generate its list items through an array of complex data. For this, the appropriate columns should be mapped to the [fields](#) property.

In the following example, **Id** column and **Game** column from complex data have been mapped to the **value** field and **text** field, respectively.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule, ReactiveFormsModule } from '@angular/forms'
import { DropDownListModule } from '@syncfusion/ej2-angular-dropdowns'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component } from '@angular/core';
@Component({
  imports: [
    FormsModule, ReactiveFormsModule, DropDownListModule, ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the DropDownList component with
  change event
  template: `<ejs-dropdownlist id='ddlelement' #samples [dataSource]='data'
  [fields]='fields' [placeholder]='text'></ejs-dropdownlist>`
})
export class AppComponent {
  constructor() {
  }
  // defined the array of data

```



```

    public data: { [key: string]: Object }[] = [ { Id: 'game1', Game:
'Badminton' },
        { Id: 'game2', Game: 'Football' }, { Id: 'game3', Game:
'Tennis' }];
    // maps the appropriate column to fields property
    public fields: Object = { text: 'Game', value: 'Id' };
    //set the placeholder to DropDownList input
    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));

```

3. Array of Complex data

The DropDownList can generate its list items through an array of complex data. For this, the appropriate columns should be mapped to the [fields](#) property.

In the following example, **Code.Id** column and **Country.Name** column from complex data have been mapped to the **value** field and **text** field, respectively.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule, ReactiveFormsModule } from '@angular/forms'
import { DropDownListModule } from '@syncfusion/ej2-angular-dropdowns'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component } from '@angular/core';
@Component({
  imports: [
    FormsModule, ReactiveFormsModule, DropDownListModule, ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the DropDownList component with
  change event
  template: `<ejs-dropdownlist id='ddlelement' #samples [dataSource]='data'
[fields]='fields' [placeholder]='text'></ejs-dropdownlist>`
})
export class AppComponent {
  constructor() {
  }
  // defined the array of data
  public data: { [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' } }
  ];
}

```

```
// maps the appropriate column to fields property
public fields: Object = { text: 'Country.Name', value: 'Code.Id' };
//set the placeholder to DropDownList input
public text: 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 DropDownList supports retrieval of data from remote data services with the help

of **DataManager** component. The **Query** property is used to fetch data from the database and bind it to the DropDownList.

The following sample displays the first 6 contacts from “Customers” table of the **Northwind** Data Service.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule, ReactiveFormsModule } from '@angular/forms'
import { DropDownListModule } from '@syncfusion/ej2-angular-dropdowns'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component } from '@angular/core';
import { Query, DataManager, ODataV4Adaptor } from '@syncfusion/ej2-data'
@Component({
  imports: [
    FormsModule, ReactiveFormsModule, DropDownListModule, ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the DropDownList component with
  // change event
  template: `<ejs-dropdownlist id='ddlelement' #samples [dataSource]='data'
[fields]='fields' [placeholder]='text' [query]='query'
[sortOrder]='sorting'></ejs-dropdownlist>`
})
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(6);
    //set the placeholder to DropDownList input
    public text: string = "Select a customer";
    //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.

DropDownList 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, ViewChild } 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 DropDownList component with dataSource
```

```
  template: <ejs-dropdownlist id='customers2' formControlName="skillname" name="skillname"
#remote2 [dataSource]='data | async' [fields]='remoteFields'
[placeholder]='remoteWaterMark' ></ejs-dropdownlist >,

```

```
})
```

```
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 DropDownList 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, AutoCompleteModule } 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,
    AutoCompleteModule,
    DialogModule,
    HttpClientModule,
    ReactiveFormsModule
  ],
  declarations: [ AppComponent ],
  bootstrap: [ AppComponent ]
})
```

```
export class AppModule { }  
`ts  
[main.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](#)
- [How to get the count of the data when using remote data](#)
- [How to achieve cascading](#)
- [How to add item in between the options](#)
- [How to remove an item](#)

Value binding in DropDownList

Value binding in the DropDownList control allows you to associate data values with each list item. This facilitates managing and retrieving selected values efficiently. The DropDownList component provides flexibility in binding both primitive data types and complex objects.

Primitive Data Types

The DropDownList 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 DropDownList.

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 { DropDownListModule } from '@syncfusion/ej2-angular-dropdowns'
import { Component } from '@angular/core';
import { DropDownListComponent } from '@syncfusion/ej2-angular-dropdowns';
@Component({
  imports: [
    FormsModule, DropDownListModule
  ],
  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"
  // 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));

```

Object Data Types

In the DropDownList control, object binding allows you to bind to a dataset of objects. When [Link to the Value](#) 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 { DropDownListModule } from '@syncfusion/ej2-angular-dropdowns'
import { Component } from '@angular/core';
import { DropDownListComponent } from '@syncfusion/ej2-angular-dropdowns';
@Component({
  imports: [
    FormsModule, DropDownListModule
  ],
  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' };
    // 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));

```

Templates in Angular Drop down list component

The DropDownList has been provided with several options to customize each list items, group title, selected value, header, and footer elements.

To get started quickly with templates in angular DropDownList component, you can check the video below.

Item template

The content of each list item within the DropDownList can be customized with the help of [itemTemplate](#) property.

In the following sample, each list item is split into two columns to display relevant data's.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule } from '@angular/forms'
import { DropDownListModule } from '@syncfusion/ej2-angular-dropdowns'
import { Component } from '@angular/core';
import { Query, DataManager, ODataV4Adaptor } from '@syncfusion/ej2-data'
@Component({
  imports: [
    FormsModule, DropDownListModule
  ],
  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 DropDownList 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 DropDownList component-->
    <ejs-dropdownlist id='dropdownlist-template' [dataSource]='data'
    [fields]='fields' [sortOrder]='sorting' [query]='query'
    [popupHeight]='height' [placeholder]='text' [itemTemplate]='itemTemplate'>
      <ng-template #itemTemplate="" let-data="">
        <!--set the value to itemTemplate property-->
        <span><span class='name'> {{data.FirstName}}</span><span
        class ='city'>{{data.City}}</span></span>
      </ng-template>
    </ejs-dropdownlist>
  </div>
</div>

```

Value template

The currently selected value that is displayed by default on the DropDownList input element can be customized using the [valueTemplate](#) property.

In the following sample, the selected value is displayed as a combined text of both **FirstName** and **City** in the DropDownList input, which is separated by a hyphen.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule } from '@angular/forms'
import { DropDownListModule } from '@syncfusion/ej2-angular-dropdowns'
import { Component } from '@angular/core';
import { Query, DataManager, ODataV4Adaptor } from '@syncfusion/ej2-data'
@Component({
  imports: [
    FormsModule, DropDownListModule
  ],
  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 DropDownList 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">
  <div id='content' style="margin: 0px auto; width:300px;">
    <!-- specifies the template string for the DropDownList component-->
    <ejs-dropdownlist id='ddlelement' #samples [dataSource]='data'
[fields]='fields' [sortOrder]='sorting' [placeholder]='text' [query]='query'
[itemTemplate]='itemTemplate' [valueTemplate]='valueTemplate'>
      <ng-template #itemTemplate="" let-data="">
        <!--set the value to itemTemplate property-->
        <span><span class='name'> {{data.FirstName}}</span><span
class ='city'>{{data.City}}</span></span>

```

```

        </ng-template>
        <ng-template #valueTemplate="" let-data="">
            <!--set the value to valueTemplate property-->
            <span class='value'>{{data.FirstName}} - {{data.City}}</span>
        </ng-template>
    </ejs-dropdownlist>
</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 { DropDownListModule } from '@syncfusion/ej2-angular-dropdowns'
import { Component } from '@angular/core';
import { Query, Predicate, DataManager, ODataV4Adaptor } from
 '@syncfusion/ej2-data'
@Component({
  imports: [
    FormsModule, DropDownListModule
  ],
  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 DropDownList 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 DropDownList component-->
    <ejs-dropdownlist id='ddlelement' #samples [dataSource]='data'
[fields]='fields' [sortOrder]='sorting' [placeholder]='text'
[groupTemplate]='groupTemplate' [query]='query'>
      <ng-template #groupTemplate="" let-data="">
        <!--set the value to groupTemplate property-->
        <strong>{{data.City}}</strong>
      </ng-template>
    </ejs-dropdownlist>
  </div>
</div>
```

Header template

The header element is shown statically at the top of the popup list items within the DropDownList, and any custom element can be placed as a header element using the [headerTemplate](#) property.

In the following sample, the list items and its headers are designed and displayed as two columns similar to multiple columns of the grid.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule } from '@angular/forms'
import { DropDownListModule } from '@syncfusion/ej2-angular-dropdowns'
import { Component } from '@angular/core';
import { Query, DataManager, ODataV4Adaptor } from '@syncfusion/ej2-data';
@Component({
  imports: [
    FormsModule, DropDownListModule
  ],
  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 DropDownList 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 DropDownList component-->
    <ejs-dropdownlist id='ddlelement' #samples [dataSource]='data'
[fields]='fields' [sortOrder]='sorting' [placeholder]='text' [query]='query'
[headerTemplate]='headerTemplate' [itemTemplate]='itemTemplate'>
      <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-dropdownlist>
  </div>
</div>

```

Footer template

The DropDownList has options to show a footer element at the bottom of the list items in the popup list. Here, you can place any custom element as a footer element using

the [footerTemplate](#) property.

In the following sample, footer element displays the total number of list items present in the DropDownList

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule } from '@angular/forms'
import { DropDownListModule } from '@syncfusion/ej2-angular-dropdowns'
import { Component } from '@angular/core';
@Component({
  imports: [
    FormsModule, DropDownListModule
  ],
  standalone: true,
  selector: 'app-root',
  // specifies the template url path
  templateUrl: 'template.html'
})
export class AppComponent {
  constructor() {
  }
  // defined the array of data
  public data: Object[] = ['Badminton', 'Basketball', 'Cricket', 'Golf', 'Hockey'];
  // set placeholder text to DropDownList 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 DropDownList component-->
    <ejs-dropdownlist id='ddlelement' #samples [dataSource]='data'
[placeholder]='text' [footerTemplate]='footerTemplate'>
      <ng-template #footerTemplate="" let-data="">
        <!--set the value to footerTemplate property-->
        <span class='foot'> Total list item: 5</span>
      </ng-template>
    </ejs-dropdownlist>
  </div>
</div>
```

No records template

The DropDownList is provided with support to custom design the popup list content when no data is found and no matches found on search with the help of

[noRecordsTemplate](#) property.

In the following sample, popup list content displays the notification of no data available.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule } from '@angular/forms'
import { DropDownListModule } from '@syncfusion/ej2-angular-dropdowns'
import { Component } from '@angular/core';
@Component({
  imports: [
    FormsModule, DropDownListModule
  ],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the DropDownList component
  template: `<ejs-dropdownlist id='ddlelement' [dataSource]='data'
placeholder='Find a item'>
      <ng-template #noRecordsTemplate>
        <span class='norecord'> NO DATA AVAILABLE</span>
      </ng-template>
    </ejs-dropdownlist>`
})
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 DropDownList displays the notification.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule } from '@angular/forms'
import { DropDownListModule } 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, DropDownListModule
  ],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the DropDownList component
  template: `<ejs-dropdownlist id='ddlelement' [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-dropdownlist>`
})
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 achieve filtering](#)
- [How to group the data using header](#)
- [How to show the list items with icon](#)
- [How to render tooltip for the options](#)

Virtualization in DropDown List

Dropdown list virtualization is a technique used to efficiently render extensive lists of items while minimizing the impact on performance. This method is particularly advantageous when dealing with large datasets because it ensures that only a fixed number of DOM (Document Object Model) elements are created. When scrolling through the list, existing DOM elements are reused to display relevant data instead of generating new elements for each item. This recycling process is managed internally.

During virtual scrolling, the data retrieved from the data source depends on the popup height and the calculation of the list item height. Enabling the [enableVirtualization](#) option in a dropdown list activates this virtualization technique.

When fetching data from the data source, the [actionBegin](#) event is triggered before data retrieval begins. Then, the [actionComplete](#) event is triggered once the data is successfully fetched.

Furthermore, Incremental Search is supported with virtualization in the DropDownList component. When a key is typed, the focus is moved to the respective element, and the value is updated in the component in the open popup state. In the closed popup state, the respective value is updated in the

component based on the typed key. The Incremental Search functionality is well-suited for scenarios involving remote data binding.

When the `enableVirtualization` property is enabled, the `skip` and `take` properties provided by the user within the Query class at the initial state or during the `actionBegin` or `actionComplete` events will not be considered, since it is internally managed and calculated based on certain dimensions with respect to the popup height.

Binding local data

The DropDownList can generate its list items through an array of complex data. For this, the appropriate columns should be mapped to the `fields` property. When using virtual scrolling, the list updates based on the scroll offset value, triggering a request to fetch more data from the server. As the data is being fetched, the `actionBegin` event occurs before the data retrieval starts. Once the data retrieval is successful, the `actionComplete` event is triggered, indicating that the data fetch process is complete.

In the following example, `id` column and `text` column from complex data have been mapped to the `value` field and `text` field, respectively.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule } from '@angular/forms'
import { DropDownListModule } from '@syncfusion/ej2-angular-dropdowns'
import { Component } from '@angular/core';
import { DropDownListComponent, VirtualScroll } from '@syncfusion/ej2-angular-dropdowns';
DropDownListComponent.Inject(VirtualScroll);
@Component({
  imports: [
    FormsModule, DropDownListModule
  ],
  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 DropDownList component-->
    <ejs-dropdownlist id='dropdownlist-virtualization'
[dataSource]='records' [fields]='fields' popupHeight='200px'
[enableVirtualization]='true' [allowFiltering]='false'
[placeholder]='waterMark'>
      </ejs-dropdownlist>
    </div>
  </div>
```

Binding remote data

The DropDownList supports retrieval of data from remote data services with the help of **DataManager** component. When using remote data, it initially fetches all the data from the server, triggering the **actionBegin** and **actionComplete** events, and then stores the data locally. During virtual scrolling, additional data is retrieved from the locally stored data, triggering the **actionBegin** and **actionComplete** events at that time as well.

The following sample displays the OrderId from the **Orders** Data Service.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule } from '@angular/forms'
import { DropDownListModule } from '@syncfusion/ej2-angular-dropdowns'
import { Component } from '@angular/core';
import { DropDownListComponent, VirtualScroll } from '@syncfusion/ej2-angular-dropdowns';
import { Query, DataManager, WebApiAdaptor } from '@syncfusion/ej2-data';
DropDownListComponent.Inject(VirtualScroll);
@Component({
  imports: [
    FormsModule, DropDownListModule
  ],
  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/Orders',
    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 DropDownList component-->
    <ejs-dropdownlist id='dropdownlist-virtualization'
[dataSource]='customerData' [fields]='customerField' popupHeight='200px'
[enableVirtualization]='true' [allowFiltering]='true'
[placeholder]='waterMark'>
      </ejs-dropdownlist>
    </div>
  </div>

```

Grouping

The DropDownList component supports grouping with Virtualization. It allows you to organize elements into groups based on different categories. Each item in the list can be classified using the [groupBy](#) field in the data table. After grouping, virtualization works similarly to local data binding, providing a seamless user experience. When the data source is bound to remote data, an initial request is made to retrieve all data for the purpose of grouping. Subsequently, the grouped data works in the same way as local data binding on virtualization.

The following sample shows the example for Grouping with Virtualization.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule } from '@angular/forms'
import { DropDownListModule } from '@syncfusion/ej2-angular-dropdowns'
import { Component } from '@angular/core';
import { DropDownListComponent, VirtualScroll } from '@syncfusion/ej2-
angular-dropdowns';
DropDownListComponent.Inject(VirtualScroll);
@Component({
  imports: [
    FormsModule, DropDownListModule
  ],
  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 DropDownList component-->
        <ejs-dropdownlist id='dropdownlist-virtualization-grouping'
[dataSource]='records' [fields]='fields' popupHeight='200px'
[enableVirtualization]='true' [allowFiltering]='true'
[placeholder]='waterMark'>
            </ejs-dropdownlist>
        </div>
    </div>

```

Filtering with Virtualization

The DropDownList component supports Filtering with Virtualization. The DropDownList includes a built-in feature that enables data filtering when the [allowFiltering](#) option is enabled. In the context of Virtual Scrolling, the filtering process operates in response to the typed characters. Specifically, the DropDownList sends a request to the server, utilizing the full data source, to achieve filtering. Before initiating the request, an action event is triggered. Upon successful retrieval of data from the server, an action complete event is triggered. The initial data is loaded when the popup is opened. Whether the filter list has a selection or not, the popup closes.

The following sample shows the example for Filtering with Virtualization.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule } from '@angular/forms'
import { DropDownListModule } from '@syncfusion/ej2-angular-dropdowns'
import { Component } from '@angular/core';
import { DropDownListComponent, VirtualScroll } from '@syncfusion/ej2-angular-dropdowns';
DropDownListComponent.Inject(VirtualScroll);
@Component({
  imports: [
    FormsModule, DropDownListModule
  ],
  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 DropDownList component-->
    <ejs-dropdownlist id='dropdownlist-virtualization'
[dataSource]='records' [fields]='fields' popupHeight='200px'
[enableVirtualization]='true' [allowFiltering]='true'
[placeholder]='waterMark'>
      </ejs-dropdownlist>
    </div>
  </div>
</div>

```

Grouping in Angular Drop down list component

The DropDownList supports wrapping nested elements into a group based on different categories. The category of each list item can be mapped through the [groupByLink to the Video](#) field in the data table. The group header is displayed both as inline and fixed headers. The fixed group header content is updated dynamically on scrolling the popup list with its category value.

To get started quickly with grouping in angular DropDownList component, you can check the video below.

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 { DropDownListModule } from '@syncfusion/ej2-angular-dropdowns'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component } from '@angular/core';
@Component({
  imports: [
    FormsModule, ReactiveFormsModule, DropDownListModule, ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the DropDownList component
  template: `<ejs-dropdownlist id='ddlelement' #samples [dataSource]='data'
[fields]='fields' [placeholder]='text' [popupHeight]='height'></ejs-
dropdownlist>`
})
export class AppComponent {
  constructor() {
  }
  // defined the array of data
  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' }]];
    // maps the appropriate column to fields property
    public fields: Object = { groupBy: 'Category', text: 'Vegetable', value:
    'Id' };
    // set the placeholder to the DropDownList input
    public text: string = "Select a vegetable";
    // Set the popup list height
    public height: string = '200px';
}

```

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.

See Also

- [How to limit the search result while filtering](#)
- [How to highlight the matched characters in filtering](#)
- [How to modify the result data using remote data source](#)

Filtering in Angular Drop down list component

The DropDownList has built-in support to filter data items when `allowFiltering` is enabled. The filter operation starts as soon as you start typing characters in the search box.

To display filtered items in the popup, filter the required data and return it to the DropDownList via [updateData](#) method by using the [filtering](#) event.

The following sample illustrates how to query the data source and pass the data to the DropDownList through the `updateData` method in `filtering` event.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule, ReactiveFormsModule } from '@angular/forms'
import { DropDownListModule } from '@syncfusion/ej2-angular-dropdowns'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component } from '@angular/core';
import { FilteringEventArgs } from '@syncfusion/ej2-dropdowns';
import { EmitType } from '@syncfusion/ej2-base';
import { Query } from '@syncfusion/ej2-data';
@Component({
  imports: [
    FormsModule, ReactiveFormsModule, DropDownListModule, ButtonModule

```

```

    ],
    standalone: true,
    selector: 'app-root',
    // specifies the template string for the DropDownList component with
    // change event
    template: `<ejs-dropdownlist id='ddlelement' #samples [dataSource]='data'
[fields]='fields' [placeholder]='text' [allowFiltering]='true'
(filtering)='onFiltering($event)'></ejs-dropdownlist>`
  })
  export class AppComponent {
    constructor() {
    }
    // defined the array of data
    public data: { [key: string]: Object }[] = [
      { Id: "s3", Country: "Alaska" },
      { Id: "s1", Country: "California" },
      { Id: "s2", Country: "Florida" },
      { Id: "s4", Country: "Georgia" }];
    // maps the appropriate column to fields property
    public fields: Object = { text: "Country", value: "Id" };
    // set the placeholder to the DropDownList input
    public text: string = "Select a country";
    //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.data, 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 DropDownList. This can be done by manual validation within the filter event handler.

In the following example, the remote request does not fetch the search data until the search key contains three characters.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule, ReactiveFormsModule } from '@angular/forms'
import { DropDownListModule } from '@syncfusion/ej2-angular-dropdowns'

```

```

import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component } from '@angular/core';
import { Query, DataManager, ODataV4Adaptor } from '@syncfusion/ej2-data';
import { FilteringEventArgs } from '@syncfusion/ej2-dropdowns';
import { EmitType } from '@syncfusion/ej2-base';
@Component({
  imports: [
    FormsModule, ReactiveFormsModule, DropDownListModule, ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the DropDownList component with
  // change event
  template: `<ejs-dropdownlist id='ddlelement' #samples [dataSource]='data'
[query]='query' [fields]='fields' [placeholder]='text'
[allowFiltering]='true' [sortOrder]='sorting'
(filtering)='onFiltering($event)'></ejs-dropdownlist>`
})
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
    });
    public query: Query = new
    Query().from('Customers').select(['ContactName', 'CustomerID']).take(6);
    // maps the appropriate column to fields property
    public fields: Object = { text: 'ContactName', value: 'CustomerID' };
    // set the placeholder to the DropDownList input
    public text: string = "Select a customer";
    //sort the result 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.data);
      } else {
        // restrict the remote request until search key contains 3
        // characters.
        if (e.text.length < 3) { return; }
        let query: Query = new
        Query().from('Customers').select(['ContactName', 'CustomerID']);
        query = (e.text !== '') ? query.where('ContactName', 'startswith',
        e.text, true) : query;
        e.updateData(this.data, 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 { DropDownListModule } from '@syncfusion/ej2-angular-dropdowns'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component } from '@angular/core';
import { Query, DataManager, ODataV4Adaptor } from '@syncfusion/ej2-data';
import { FilteringEventArgs } from '@syncfusion/ej2-dropdowns';
import { EmitType } from '@syncfusion/ej2-base';
@Component({
  imports: [
    FormsModule, ReactiveFormsModule, DropDownListModule, ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the DropDownList component with
  // change event
  template: `<ejs-dropdownlist id='ddlelement' #samples [dataSource]='data'
[query]='query' [fields]='fields' [placeholder]='text' [popupHeight]='height'
[sortOrder]='sorting' [allowFiltering]='true'
(filtering)='onFiltering($event)'></ejs-dropdownlist>`
})
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
  });
  public query: Query = new
  Query().from('Customers').select(['ContactName', 'CustomerID']).take(7);
  // maps the appropriate column to fields property
  public fields: Object = { text: 'ContactName', value: 'CustomerID' };
  // set the placeholder to the DropDownList input
  public text: string = "Select a customer";
  // set the height of the popup
  public height: string = '250px';
  //sort the result 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.data);
    } else {
        let query: Query = new
Query().from('Customers').select(['ContactName', 'CustomerID']);
        // change the type of filtering
        query = (e.text !== '') ? query.where('ContactName', 'endswith',
e.text, true) : query;
        e.updateData(this.data, 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 { DropDownListModule } from '@syncfusion/ej2-angular-dropdowns'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component } from '@angular/core';
import { Query, DataManager, ODataV4Adaptor } from '@syncfusion/ej2-data';
import { FilteringEventArgs } from '@syncfusion/ej2-dropdowns';
import { EmitType } from '@syncfusion/ej2-base';
@Component({
  imports: [
    FormsModule, ReactiveFormsModule, DropDownListModule, ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the DropDownList component with
  // change event
  template: `<ejs-dropdownlist id='ddlelement' #samples [dataSource]='data'
[query]='query' [fields]='fields' [placeholder]='text' [popupHeight]='height'
[sortOrder]='sorting' [allowFiltering]='true'
(filtering)='onFiltering($event)'></ejs-dropdownlist>`
})
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
});

public query: Query = new
Query().from('Customers').select(['ContactName', 'CustomerID']).take(7);
// maps the appropriate column to fields property
public fields: Object = { text: 'ContactName', value: 'CustomerID' };
// set the placeholder to the DropDownList input
public text: string = "Select a customer";
// set the height of the popup
public height?: '250px';
//sort the result 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.data);
    } else {
        let query: Query = new
Query().from('Customers').select(['ContactName', 'CustomerID']);
        //enable the case sensitive filtering by passing false to 4th
parameter.
        query = (e.text !== '') ? query.where('ContactName', 'startswith',
e.text, false) : query;
        e.updateData(this.data, 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

The DropDownList supports diacritics filtering which will ignore the [diacritics](#) and makes it easier to filter the results in international characters lists when the [ignoreAccent](#) is enabled.

In the following sample, data with diacritics are bound as dataSource for DropDownList.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule, ReactiveFormsModule } from '@angular/forms'
import { DropDownListModule } from '@syncfusion/ej2-angular-dropdowns'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component } from '@angular/core';
@Component({

```

```

imports: [
    FormsModule, ReactiveFormsModule, DropDownListModule, ButtonModule
],
standalone: true,
selector: 'app-root',
// specifies the template string for the DropDownList component with
change event
template: `<ejs-dropdownlist id='diacritics' [dataSource]='data'
[allowFiltering]='true' [ignoreAccent]='true' placeholder='Select a value'
    filterBarPlaceholder='e.g: aero'>
    </ejs-dropdownlist>`
})
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 limit the search result while filtering](#)
- [How to highlight the matched characters in filtering](#)
- [How to modify the result data using remote data source](#)

Localization in Angular Drop down list component

The Localization library allows you to localize static text content of the

[noRecordsTemplate](#) and [actionFailureTemplate](#) properties according to the culture currently assigned to the DropDownList.

| Locale key | en-US (default) |

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

| noRecordsTemplate | No records found |

| actionFailureTemplate | The request failed |

Loading translations

To load translation object to your application, use load function of the **L10n** class.

In the following sample, French culture is set to the DropDownList and no data is loaded. Hence, the [noRecordsTemplate](#) property displays its text in French culture initially, and if the sample is run offline, the [actionFailureTemplate](#) property displays its text appropriately.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule, ReactiveFormsModule } from '@angular/forms'
import { DropDownListModule } 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 } from '@syncfusion/ej2-base';
@Component({
  imports: [
    FormsModule, ReactiveFormsModule, DropDownListModule, ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the DropDownList component
  template: `<ejs-dropdownlist id='ddlelement' #samples [dataSource]='data'
[query]='query' [fields]='fields' [placeholder]='text'
[locale]='locale'></ejs-dropdownlist>`
})
export class AppComponent implements OnInit {
  constructor() {
  }
  //set the placeholder text in french to DropDownList input
  public text: string = "Sélectionnez un élément";
  // bind remotedata to showcase actionFailureTemplate in offline
  public data: 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 DropDownList component
  public locale: string = 'fr-BE';
  ngOnInit(): void {
    L10n.load({
      'fr-BE': {
        'dropdowns': {
          'noRecordsTemplate': "Aucun enregistrement trouvé",
          'actionFailureTemplate': "Modèle d'échec d'action"
        }
      }
    });
  }
}
```

```
}
```

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 Drop down list component

The following content provides the exact CSS structure that can be used to modify the control's appearance based on the user preference.

Customizing the appearance of wrapper element

Use the following CSS to customize the appearance of wrapper element.

```
`css
.e-ddl.e-input-group.e-control-wrapper .e-input {
font-size: 20px;
font-family: emoji;
color: #ab3243;
background: #32a5ab;
}
`
```

Customizing the dropdown icon's color

Use the following CSS to customize the dropdown icon's color.

```
`css
.e-ddl.e-input-group .e-input-group-icon,.e-ddl.e-input-group.e-control-wrapper .e-input-group-
icon:hover {
color: #bb233d;
font-size: 13px;
}
`
```

Customizing the focus color

Use the following CSS to customize the focusing color of input element.

```
`css
```

```
.e-ddl.e-input-group.e-control-wrapper.e-input-focus::before, .e-ddl.e-input-group.e-control-wrapper.e-input-focus::after {
background: #c000ff;
}
`css
```

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;
}
`css
```

Customizing the disabled component's text color

Use the following CSS to customize the text color when the component is disabled.

```
`css
.e-input-group.e-control-wrapper .e-input[disabled] {
-webkit-text-fill-color: #0d9133;
}
`css
```

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-ddl.e-lib.e-input-group.e-control-wrapper.e-float-input.e-input-focus .e-float-text.e-label-top {
color: #2319b8;
}
`css
```

,

Customizing the color of the placeholder text

Use the following CSS to customize the text color of placeholder.

```
`css
.e-ddl.e-input-group input.e-input::placeholder {
color: red;
}
```

,

Customizing the background color of focus, hover, and active item's

Use the following CSS to customize the background color of focus, hover and active item's.

```
`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;
}
```

,

Adding mandatory asterisk to placeholder and float label

You can add a mandatory asterisk(*) to placeholder and float label using `.e-input-group.e-control-wrapper.e-float-input .e-float-text::after` class.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule, ReactiveFormsModule } from '@angular/forms'
import { DropDownListModule } from '@syncfusion/ej2-angular-dropdowns'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component } from '@angular/core';
```



```
@Component({
  imports: [
    FormsModule, ReactiveFormsModule, DropDownListModule, ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the DropDownList component with
  // dataSource
  template: `<ejs-dropdownlist id='ddlelement' [dataSource]='data'
placeholder = 'Select a game' floatLabelType="Auto"></ejs-dropdownlist>`
})
export class AppComponent {
  constructor() {
  }
  // defined the array of data
  public data: string[] = ['Snooker', 'Tennis', 'Cricket', 'Football',
'Rugby'];
}
```

MAIN.TS

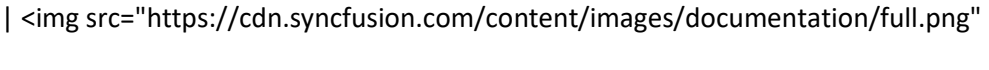
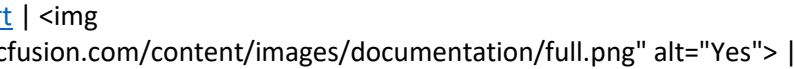
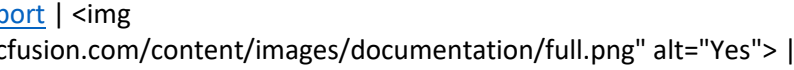
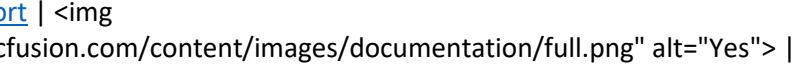
```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Accessibility in Angular Drop down list component

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

The DropDownList component followed the accessibility guidelines and standards, including [ADA](#), [Section 508](#), [WCAG 2.2](#) standards, and [WCAG roles](#) that are commonly used to evaluate accessibility.

The accessibility compliance for the DropDownList component is outlined below.

Accessibility Criteria Compatibility
-- --
WCAG 2.2 Support 
Section 508 Support 
Screen Reader Support 
Right-To-Left Support 

| [Color Contrast](#) | |

| [Mobile Device Support](#) | |

| [Keyboard Navigation Support](#) | |

| [Accessibility Checker Validation](#) | |

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

<style>

.post .post-content img {

display: inline-block;

margin: 0.5em 0;

}

</style>

<div> - All features of the component meet the requirement.</div>

<div> - Some features of the component do not meet the requirement.</div>

<div> - The component does not meet the requirement.</div>

WAI-ARIA attributes

The DropDownList component uses the **Listbox** role, and each list item has an **option** role. The following **ARIA attributes** denote the DropDownList state.

| **Properties** | **Functionalities** |

| --- | --- |

| aria-haspopup | Indicates whether the DropDownList input element has a popup list or not. |

| aria-expanded | Indicates whether the popup list has expanded or not. |

| aria-selected | Indicates the selected option. |

| aria-readonly | Indicates the readonly state of the DropDownList element. |

| aria-disabled | Indicates whether the DropDownList component is in a disabled state or not. |

| aria-activedescendent | This attribute holds the ID of the active list item to focus its descendant child element. |

| aria-owns | This attribute contains the ID of the popup list to indicate popup as a child element. |

Keyboard Interaction

You can use the following key shortcuts to access the DropDownList without interruptions.

| **Keyboard shortcuts** | **Actions** |

| --- | --- |

| **Arrow Down** | Selects the first item in the DropDownList when no item selected. Otherwise, selects the item next to the currently selected item. |

| **Arrow Up** | Selects the item previous to the currently selected one. |

| **Page Down** | Scrolls down to the next page and selects the first item when popup list opens. |

| **Page Up** | Scrolls up to the previous page and selects the first item when popup list opens. |

| **Enter** | Selects the focused item, and when it is in an open state the popup list closes. Otherwise, toggles the popup list. |

| **Tab** | Focuses on the next TabIndex element on the page when the popup is closed. Otherwise, closes the popup list and remains the focus of the component. |

| **Shift + tab** | Focuses on the previous TabIndex element on the page when the popup is closed. Otherwise, closes the popup list and remains the focus of the component. |

| **Alt + Down** | Opens the popup list. |

| **Alt + Up** | Closes the popup list. |

| **Esc(Escape)** | Closes the popup list when it is in an open state and the currently selected item remains the same. |

| **Home** | Selects the first item. |

| **End** | Selects the last item. |

In the following sample, alt+t keys are used to focus the DropDownList component.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule, ReactiveFormsModule } from '@angular/forms'
import { DropDownListModule } from '@syncfusion/ej2-angular-dropdowns'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component, HostListener, ViewChild } from '@angular/core';
import { DropDownListComponent } from '@syncfusion/ej2-angular-dropdowns';
@Component({
  imports: [
    FormsModule, ReactiveFormsModule, DropDownListModule, ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the DropDownList component with
  // change event
  template: `<ejs-dropdownlist id='ddlelement' #samples [dataSource]='data'
[placeholder]='text' [popupHeight]='height'></ejs-dropdownlist>`
})
export class AppComponent {
  @ViewChild('samples')
  public sports?: DropDownListComponent;
```

```

    constructor() {
    }
    // defined the array of data
    public data: string[] = ['Badminton', 'Basketball', 'Cricket',
'FootBall', 'Golf', 'Hockey', 'Rugby', 'Snooker', 'Tennis'];
    // set placeholder to DropDownList input element
    public text: string = "Select a game";
    // set the popup list height
    public height: string = '200px';
    @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 DropDownList component's accessibility levels are ensured through an [accessibility-checker](#) and [axe-core](#) software tools during automated testing.

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

See also

- [Accessibility in Syncfusion Angular components](#)

Form support in Angular Drop down list component

The DropDownList 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 DropDownList element which will be used to identify the form element. To register an DropDownList element to ngForm, give the `ngModel` to it so the FormsModule will automatically detect the DropDownList as a form element. After that, the DropDownList 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 { DropDownListModule } 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, DropDownListModule, 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 `DropDownList` 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 { DropDownListModule } 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, DropDownListModule, ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  templateUrl: 'reactive-form.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

Add item in Angular Drop down list component

You can add item in between based on item [Link to the Video](#). If you add new item without item index, item will be added as last item in list.

To get started quickly with adding items in angular DropDownList component, you can check the video below.

The following example demonstrate how to add item in between in DropDownList.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule } from '@angular/forms'
import { DropDownListModule } from '@syncfusion/ej2-angular-dropdowns'
import { Component, ViewChild } from '@angular/core';
import { DropDownListComponent } from '@syncfusion/ej2-angular-dropdowns';
@Component({
  imports: [
    FormsModule, DropDownListModule
  ],
  standalone: true,
  selector: 'control-content',
  // specifies the template path for DropDownList component
  templateUrl: `add.html`
})
export class AppComponent {
  constructor() {
  }
  ngAfterViewInit() {
    // add item at first
    document.getElementById('first')!.onclick = () => {
      (this.dropDownListObject as any).addItem({Id: 'game0', Game:
'Hockey'}, 0);
    }
    // add item in between
    document.getElementById('between')!.onclick = () => {
      (this.dropDownListObject as any).addItem({Id: 'game4', Game:
'Golf'}, 2);
    }
    // add item at last
    document.getElementById('last')!.onclick = () => {
      (this.dropDownListObject as any).addItem({Id: 'game5', Game:
'Cricket'});
    }
  }
  // defined the array of data
  public data: { [key: string]: Object }[] = [ { Id: 'game1', Game:
'Badminton' },
    { Id: 'game2', Game: 'Football' }, { Id: 'game3', Game:
'Tennis' }];
  // maps the appropriate column to fields property
  public fields: Object = { text: 'Game', value: 'Id' };
  //set the placeholder to DropDownList input
  public text: string = "Select a game";
  @ViewChild('ddlelement')
```

```
public dropDownListObject = DropDownListComponent;
}
```

MAIN.TS

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

Cascading in Angular Drop down list component

The cascading DropDownList is a series of DropDownList, where the value of one DropDownList depends upon another's value. This can be configured by using the [Link to the Video](#) event of the parent DropDownList. Within that change event handler, data has to be loaded to the child DropDownList based on the selected value of the parent DropDownList.

To get started quickly with cascading in angular DropDownList component, you can check the video below.

The following example, shows the cascade behavior of country, state, and city

DropDownList. Here, the `dataBind` method is used to reflect the property changes immediately to the DropDownList.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule } from '@angular/forms'
import { DropDownListModule } from '@syncfusion/ej2-angular-dropdowns'
import { Component, ViewChild } from '@angular/core';
import { DropDownListComponent } from '@syncfusion/ej2-angular-dropdowns';
import { Query, DataManager } from '@syncfusion/ej2-data';
@Component({
  imports: [
    FormsModule, DropDownListModule
  ],
  standalone: true,
  selector: 'app-root',
  // specifies the template path for DropDownList component
  templateUrl: `cascading.html`
})
export class AppComponent {
  constructor() {
  }
  //define the country DropDownList data
  public countryData: { [key: string]: Object }[] = [
    { CountryName: 'Australia', CountryId: '2' },
    { CountryName: 'United States', CountryId: '1' }
  ];
  //define the state DropDownList data
  public stateData: { [key: string]: Object }[] = [
    { StateName: 'New York', CountryId: '1', SateId: '101' },
    { StateName: 'Virginia ', CountryId: '1', SateId: '102' },
    { StateName: 'Tasmania ', CountryId: '2', SateId: '105' }
  ]
}
```



```

];
//define the city DropDownList data
public cityData: { [key: string]: Object }[] = [
    { CityName: 'Albany', SateId: '101', CityId: 201 },
    { CityName: 'Beacon ', SateId: '101', CityId: 202 },
    { CityName: 'Emporia', SateId: '102', CityId: 206 },
    { CityName: 'Hampton ', SateId: '102', CityId: 205 },
    { CityName: 'Hobart', SateId: '105', CityId: 213 },
    { CityName: 'Launceston ', SateId: '105', CityId: 214 }
];
// maps the appropriate column to fields property for country
DropDownList
public countryFields: Object = { text: 'CountryName', value: 'CountryId'
};
// maps the appropriate column to fields property for state DropDownList
public stateFields: Object = { text: 'StateName', value: 'SateId' };
// maps the appropriate column to fields property for city DropDownList
public cityFields: Object = { text: 'CityName', value: 'CityId' };
//set the placeholder to country DropDownList input
public countryWatermark: string = "Select a country";
//set the placeholder to state DropDownList input
public stateWatermark: string = "Select a state";
//set the placeholder to city DropDownList input
public cityWatermark: string = "Select a city";
@ViewChild('country')
public countryObj?: DropDownListComponent | any;
@ViewChild('state')
public stateObj?: DropDownListComponent | any;
@ViewChild('city')
public cityObj?: DropDownListComponent | any;
public countryChange(): void {
    let tempQuery: Query = new Query().where('CountryId', 'equal',
this.countryObj.value);
    //Query the data source based on country DropDownList selected value
    this.stateObj.query = tempQuery;
    // enable the state DropDownList
    this.stateObj.enabled = true;
    //clear the existing selection.
    this.stateObj.text = null;
    // bind the property changes to state DropDownList
    this.stateObj.dataBind();
    //clear the existing selection in city DropDownList
    this.cityObj.text = null;
    //disabe the city DropDownList
    this.cityObj.enabled = false;
    //bind the property cahnges to City DropDownList
    this.cityObj.dataBind();
}
public stateChange(): void {
    // Query the data source based on state DropDownList selected value
    this.cityObj.query = new Query().where('SateId', 'equal',
this.stateObj.value);
    // enable the city DropDownList
    this.cityObj.enabled = true;
    //clear the existing selection
    this.cityObj.text = null;
    // bind the property change to city DropDownList

```

```

        this.cityObj.dataBind();
    }
}

```

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-dropdownlist #country id="country"
[dataSource]="countryData" [fields]="countryFields"
(change)="countryChange()" [placeholder]="countryWatermark"></ejs-
dropdownlist>
        <div class="padding-top">
            <ejs-dropdownlist #state id="state" [dataSource]="stateData"
[fields]="stateFields" (change)="stateChange()"
[placeholder]="stateWatermark" [enabled]="false"></ejs-dropdownlist>
        </div>
        <div class="padding-top">
            <ejs-dropdownlist #city id="city" [dataSource]="cityData"
[fields]="cityFields" [placeholder]="cityWatermark" [enabled]="false"></ejs-
dropdownlist>
        </div>
    </div>
</div>

```

Clear item in Angular Drop down list component

You can clear the selected item in the below two different ways.

By clicking on the **clear icon** which is shown in DropDownList element, you can clear the selected item in DropDownList through **interaction**. By using [showClearButton](#)

property, you can enable the clear icon in DropDownList element.

Through **programmatic** you can set **null** value to anyone of the index, text or value property to clear the selected item in DropDownList.

The following example demonstrate about how to clear the selected item in DropDownList.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule } from '@angular/forms'
import { DropDownListModule } from '@syncfusion/ej2-angular-dropdowns'
import { Component, ViewChild } from '@angular/core';
import { DropDownListComponent } from '@syncfusion/ej2-angular-dropdowns';
import { Button } from '@syncfusion/ej2-buttons';
@Component({

```

```

imports: [
    FormsModule, DropDownListModule
],
standalone: true,
selector: 'control-content',
// specifies the template string for the DropDownList component with
change event
templateUrl: `clear.html`
})
export class AppComponent {
    constructor() {
    }
    ngAfterViewInit() {
        // Set null value to value property for clear the selected item
        document.getElementById('btn')!.onclick = () => {
            (this.dropDownListObject as any).value = null;
        }
    }
    // defined the array of data
    public data: string[] = ['Badminton', 'Basketball', 'Cricket', 'Golf',
'Hockey', 'Rugby'];
    // set placeholder text to DropDownList input element
    public placeholder: string = 'Select a game';
    @ViewChild('ddlelement')
    public dropDownListObject?: DropDownListComponent;
}

```

MAIN.TS

```

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

```

Close popup in Angular Drop down list component

By using the `hidePopup` method in `DropDownList`, you can close the popup on scroll when triggered the windows scroll event.

The following example demonstrate about how to close the popup on scroll.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule, ReactiveFormsModule } from '@angular/forms'
import { DropDownListModule } from '@syncfusion/ej2-angular-dropdowns'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component, ViewChild } from '@angular/core';
import { DropDownListComponent } from '@syncfusion/ej2-angular-dropdowns';
@Component({
    imports: [
        FormsModule, ReactiveFormsModule, DropDownListModule, ButtonModule
    ],
    standalone: true,
    selector: 'app-root',

```

```

    // specifies the template string for the DropDownList component with
    change event
    template: `<ejs-dropdownlist #ddlelement id='ddlelement' #samples
[dataSource]='data' [placeholder]='placeholder'></ejs-dropdownlist>`
  })
  export class AppComponent {
    constructor() {
      // bind the onscroll event to window
      window.onscroll = () => {
        this.dropDownListObject!.hidePopup();
      }
    }
    // defined the array of data
    public data: string[] = ['Badminton', 'Basketball', 'Cricket', 'Golf',
'Hockey', 'Rugby'];
    // set placeholder text to DropDownList input element
    public placeholder: string = 'Select a game';
    @ViewChild('ddlelement')
    public dropDownListObject?: DropDownListComponent;
  }

```

MAIN.TS

```

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

```

Group header in Angular Drop down list component

The following example demonstrate about how to disable the Fixed group header in DropDownList through CSS by using visibility attribute.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule } from '@angular/forms'
import { DropDownListModule } from '@syncfusion/ej2-angular-dropdowns'
import { Component } from '@angular/core';
@Component({
  imports: [
    FormsModule, DropDownListModule
  ],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the DropDownList component
  template: `<ejs-dropdownlist id='ddlelement' #samples [dataSource]='data'
[fields]='fields' [placeholder]='text' [popupHeight]='height'></ejs-
dropdownlist>`,
})
export class AppComponent {
  constructor() {
  }
  // defined the array of data

```

```

    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' }];
    // maps the appropriate column to fields property
    public fields: Object = { groupBy: 'Category', text: 'Vegetable', value:
    'Id' };
    // set the placeholder to the DropDownList input
    public text: string = "Select a vegetable";
    // Set the popup list height
    public height: string = '200px';
  }

```

MAIN.TS

```

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

```

Highlight filtering in Angular Drop down list component

By using the `highlightSearch` method, you can highlight the matched character in DropDownList filtering.

The following example demonstrates about how to highlight the matched character in filtering.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule, ReactiveFormsModule } from '@angular/forms'
import { DropDownListModule } from '@syncfusion/ej2-angular-dropdowns'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component } from '@angular/core';
import { Query, DataManager, ODataV4Adaptor } from '@syncfusion/ej2-data';
import { FilteringEventArgs, highlightSearch } from '@syncfusion/ej2-
dropdowns';
@Component({
  imports: [
    FormsModule, ReactiveFormsModule, DropDownListModule, ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the DropDownList component with
  change event
  template: `<ejs-dropdownlist id='ddlelement' [dataSource]='data'
[query]='query' [fields]='fields' [placeholder]='text'

```

```

[allowFiltering]='true' [sortOrder]='sorting'
(filtering)='onFiltering($event)'></ejs-dropdownlist>`
})
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
  });
  public query: Query = new
  Query().from('Customers').select(['ContactName', 'CustomerID']).take(6);
  // maps the appropriate column to fields property
  public fields: Object = {
    text: 'ContactName', value: 'CustomerID', itemCreated: (e: any) => {
      highlightSearch(e.item, (this as any).queryString, true,
'StartsWith');
    }
  };
  // set the placeholder to the DropDownList input
  public text: string = "Select a customer";
  //sort the result items
  public sorting: string = 'Ascending';
  public queryString?: string;
  //Bind the filter event
  public onFiltering: any = (e: FilteringEventArgs) => {
    // take text for highlight the character in list items.
    this.queryString = e.text;
    let query: Query = new
  Query().from('Customers').select(['ContactName', 'CustomerID']);
    query = (e.text !== '') ? query.where('ContactName', 'startswith',
e.text, true) : query;
    e.updateData(this.data, query);
  };
}

```

MAIN.TS

```

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

```

Icons support in Angular Drop down list 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.

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 { DropDownListModule } from '@syncfusion/ej2-angular-dropdowns'
import { Component } from '@angular/core';
@Component({
  imports: [
    FormsModule, DropDownListModule
  ],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the DropDownList component
  template: `<ejs-dropdownlist id='ddlelement' #samples [dataSource]='data'
[fields]='fields' [placeholder]='text'></ejs-dropdownlist>`
})
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 DropDownList 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));
```

Incremental search in Angular Drop down list component

DropDownList supports incremental search, by default. You can search the list item by focusing the DropDownList and typing the characters in it. The closely matched items are selected sequentially.

If the same key is searched once again, the next matched item is selected.

Modify data in Angular Drop down list component

When binding the remote data source, by using the [actionComplete](#) event, you can modify the result data before passing it to DropDownList.

The following sample demonstrate how to modify the result data.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule, ReactiveFormsModule } from '@angular/forms'
import { DropDownListModule } from '@syncfusion/ej2-angular-dropdowns'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component } from '@angular/core';
import { Query, DataManager, ODataV4Adaptor } from '@syncfusion/ej2-data'
```

```

@Component({
  imports: [
    FormsModule, ReactiveFormsModule, DropDownListModule, ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the DropDownList component with
  // change event
  template: `<ejs-dropdownlist id='ddlelement' #samples [dataSource]='data'
    [fields]='fields' [placeholder]='text' [query]='query' [sortOrder]='sorting'
    (actionComplete)="actionComplete($event)"></ejs-dropdownlist>`
})
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(6);
  //set the placeholder to DropDownList input
  public text: string = "Select a customer";
  //sort the result items
  public sorting: string = 'Ascending';
  public actionComplete(e: any): void {
    // initially result contains 6 items
    console.log("Before modified the result: " + e.result.length);
    // remove first 2 items from result.
    e.result.splice(0, 2);
    // now displays the result count is 4.
    console.log("After modified the result: " + e.result.length);
  }
}

```

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 bind in Angular Drop down list component

Before component rendering, you can get the total items count by using [actionComplete](#) event with its result arguments. After rendering this component, you can get the total items count by using [Link to the Video](#) method.

The following example demonstrate how to get the total items count.

APP.COMPONENT.TS


```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule } from '@angular/forms'
import { DropDownListModule } from '@syncfusion/ej2-angular-dropdowns'
import { Component, ViewChild } from '@angular/core';
import { DropDownListComponent } from '@syncfusion/ej2-angular-dropdowns';
import { Query, DataManager, ODataV4Adaptor } from '@syncfusion/ej2-data'
@Component({
  imports: [
    FormsModule, DropDownListModule
  ],
  standalone: true,
  selector: 'control-content',
  // specifies the template path for DropDownList component
  templateUrl: `template.html`
})
export class AppComponent {
  @ViewChild('sample')
  public dropDownListObject = DropDownListComponent;
  constructor() {
  }
  ngAfterViewInit(){
    let proxy=this;
    document.getElementById('btn')!.onclick = () => {
      // get items count using getItems method
      console.log("Total items count: " + (proxy.dropDownListObject as
any).getItems().length);
      let element: HTMLElement = document.createElement('p');
      element.innerText = "Total items count: " +
(proxy.dropDownListObject as any).getItems().length;
      document.getElementById('event')!.append(element);
    }
    //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(6);
    //set the placeholder to DropDownList input
    public text: string = "Select a customer";
    //sort the result items
    public sorting: string = 'Ascending';
    public actionComplete(e: any): void {
      // get total items count
      console.log("Total items count: " + e.result.length);
      let element: HTMLElement = document.createElement('p');
      element.innerText = "Total items count: " + e.result.length;
      document.getElementById('event')!.append(element);
    }
  }
}

```

MAIN.TS

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

Remove item in Angular Drop down list component

To get started quickly with removing items in angular DropDownList component, you can check the video below.

The following example demonstrate about how to remove an item from DropDownList.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule } from '@angular/forms'
import { DropDownListModule } from '@syncfusion/ej2-angular-dropdowns'
import { Component, ViewChild } from '@angular/core';
import { DropDownListComponent } from '@syncfusion/ej2-angular-dropdowns';
@Component({
  imports: [
    FormsModule, DropDownListModule
  ],
  standalone: true,
  selector: 'control-content',
  // specifies the template path for DropDownList component
  templateUrl: `remove.html`
})
export class AppComponent {
  constructor() {
  }
  ngAfterViewInit() {
    document.getElementById('btn')!.onclick = () => {
      // create DropDownList object
      let obj: any = document.getElementById('ddlelement');
      if (obj.ej2_instances[0].list) {
        // Remove the selected value if 0th index selected
        if (((this.dropDownListObject as any) as any).index === 0) {
          ((this.dropDownListObject as any) as any).value = null;
          ((this.dropDownListObject as any) as any).dataBind();
        }
        // remove first item in list
        (obj.ej2_instances[0].list.querySelectorAll('li')[0]).remove();
        if (!obj.ej2_instances[0].list.querySelectorAll('li')[0]) {
          (this.dropDownListObject as any).dataSource = [];
          // enable the nodata template when no data source is
          empty.
          obj.ej2_instances[0].list.classList.add('e-nodata');
        }
      } else {
        // remove first item in list
        (this.dropDownListObject as any).dataSource.splice(0, 1);
      }
    }
  }
}
```

```

    }
  }
}
// defined the array of data
public data: { [key: string]: Object }[] = [ { Id: 'game1', Game:
'Badminton' },
      { Id: 'game2', Game: 'Football' }, { Id: 'game3', Game:
'Tennis' }];
// maps the appropriate column to fields property
public fields: Object = { text: 'Game', value: 'Id' };
//set the placeholder to DropDownList input
public text: string = "Select a game";
@ViewChild('ddlelement')
public dropDownListObject = DropDownListComponent;
}

```

MAIN.TS

```

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

```

Search on filtering in Angular Drop down list component

The following example demonstrates about how to set limit the search result on filtering.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule, ReactiveFormsModule } from '@angular/forms'
import { DropDownListModule } from '@syncfusion/ej2-angular-dropdowns'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component } from '@angular/core';
import { Query, DataManager, ODataV4Adaptor } from '@syncfusion/ej2-data';
import { FilteringEventArgs } from '@syncfusion/ej2-dropdowns';
import { EmitType } from '@syncfusion/ej2-base';
@Component({
  imports: [
    FormsModule, ReactiveFormsModule, DropDownListModule, ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the DropDownList component with
  // change event
  template: `<ejs-dropdownlist id='ddlelement' #samples [dataSource]='data'
[query]='query' [fields]='fields' [placeholder]='text'
[allowFiltering]='true' [sortOrder]='sorting'
(filtering)='onFiltering($event)'></ejs-dropdownlist>`
})
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
    });
    public query: Query = new
Query().from('Customers').select(['ContactName', 'CustomerID']).take(6);
    // maps the appropriate column to fields property
    public fields: Object = { text: 'ContactName', value: 'CustomerID' };
    // set the placeholder to the DropDownList input
    public text: string = "Select a customer";
    //sort the result 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.data);
        } else {
            // set limit as 4 to search result
            let query: Query = new
Query().from('Customers').select(['ContactName', 'CustomerID']).take(4);
            query = (e.text !== '') ? query.where('ContactName', 'startswith',
e.text, true) : query;
            e.updateData(this.data, query);
        }
    };
}

```

MAIN.TS

```

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

```

Tooltip in Angular Drop down list component

You can achieve this behavior by using `ej2-tooltip` component. When the mouse hover on the DropDownList option that tooltip display some details related to hovered list item.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule, ReactiveFormsModule } from '@angular/forms'
import { DropDownListModule } from '@syncfusion/ej2-angular-dropdowns'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component } from '@angular/core';
import { Tooltip, TooltipEventArgs } from '@syncfusion/ej2-popups';
@Component({
    imports: [
        FormsModule, ReactiveFormsModule, DropDownListModule, ButtonModule
    ],
    standalone: true,
    selector: 'app-root',

```

```

    // specifies the template string for the DropDownList component with
    change event
    template: `<ejs-dropdownlist id='ddltooltip' #samples [dataSource]='data'
[fields]='fields' [placeholder]='text' (close)='onClose($event)'></ejs-
dropdownlist>`
})
export class AppComponent {
    constructor() {
    }
    public tooltip?: Tooltip | any;
    ngAfterViewInit() {
        //Initialize Tooltip component
        this.tooltip = new Tooltip({
            // default content of tooltip
            content: 'Loading...',
            // set target element to tooltip
            target: '.e-list-item',
            // set position of tooltip
            position: 'TopCenter',
            // bind beforeRender event
            beforeRender: this.onBeforeRender
        });
        this.tooltip.appendTo('body');
    }
    // defined the array of data
    public data: { [key: string]: Object }[] = [
        { id: '1', text: 'Australia', content: 'National sports is Cricket'
},
        { id: '2', text: 'Bhutan', content: 'National sports is Archery' },
        { id: '3', text: 'China', content: 'National sports is Table Tennis'
},
        { id: '4', text: 'Cuba', content: 'National sports is Baseball' },
        { id: '5', text: 'India', content: 'National sports is Hockey' },
        { id: '6', text: 'Spain', content: 'National sports is Football' },
        { id: '7', text: 'United States', content: 'National sports is
Baseball' }];
    // maps the appropriate column to fields property
    public fields: Object = { text: 'text', tooltip: 'Id' };
    //set the placeholder to DropDownList input
    public text: string = "Select a country";
    // close event
    onClose(e: any) {
        this.tooltip.close();
    }
    //beforeRender event of tooltip
    onBeforeRender(args: TooltipEventArgs): void {
        // get the target element
        let listElement = document.getElementById('ddltooltip');
        let result: Object[] = (listElement! as
any).ej2_instances[0].dataSource;
        let i: number;
        for (i = 0; i < result.length; i++) {
            if ((result[i] as any).text === args.target.textContent) {
                (this as any).content = (result[i] as any).content;
                (this as any).dataBind();
                break;
            }
        }
    }
}

```

```

    }
  }
}

```

MAIN.TS

```

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

```

Value change in Angular Drop down list component

You can check about whether value change happened by manual or programmatic by

using [change](#) event argument that argument name is `isInteracted`.

The following example demonstrate, how to check whether value change happened by manual or programmatic.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule } from '@angular/forms'
import { DropDownListModule } from '@syncfusion/ej2-angular-dropdowns'
import { Component, ViewChild } from '@angular/core';
import { DropDownListComponent } from '@syncfusion/ej2-angular-dropdowns';
@Component({
  imports: [
    FormsModule, DropDownListModule
  ],
  standalone: true,
  selector: 'app-root',
  // specifies the template path for DropDownList component
  templateUrl: `template.html`
})
export class AppComponent {
  constructor() {
  }
  ngAfterViewInit() {
    document.getElementById('btn')!.onclick = () => {
      this.dropDownListObject!.value = 'game3';
    }
  }
  // defined the array of data
  public data: { [key: string]: Object }[] = [ { Id: 'game1', Game: 'Badminton' },
    { Id: 'game2', Game: 'Football' }, { Id: 'game3', Game: 'Tennis' } ];
  // maps the appropriate column to fields property
  public fields: Object = { text: 'Game', value: 'Id' };
  //set the placeholder to DropDownList input
  public text: string = "Select a game";
  @ViewChild('ddelement')
  public dropDownListObject?: DropDownListComponent;
  onChange(args: any): void {

```

```

    let element: HTMLElement = document.createElement('p');
    if (args.isInteracted) {
        element.innerText = 'Changes happened by Interaction';
    } else {
        element.innerText = 'Changes happened by programmatic';
    }
    document.getElementById('event')!.append(element);
}
}

```

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="control-section">
  <div class="content-wrapper">
    <div id='icon'>
      <div class="content" style="margin: 50px auto 0; width:250px;">
        <ejs-dropdownlist id='ddlelement' #ddlelement
        [dataSource]='data' [fields]='fields' [placeholder]='text'
        (change)='onChange($event)'></ejs-dropdownlist>
        <div style='padding: 50px 0'>
          <button id='btn' ej-button #btn
          class="e-control e-btn"> Set Value Dynamically </button>
        </div>
        <p id='event'></p>
      </div>
    </div>
  </div>
</div>

```

Value support in Angular Drop down list component
yes, value for each list items should be unique.

Ej1 api migration in Angular Drop down list component

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

DataBinding

<!-- markdownlint-disable MD010 -->

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

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

| **Default** | **Property:** *dataSource*
<input type="text" id="dropdown1" ej-dropdownlist
[dataSource]="empList"> | **Property:** *dataSource*
<ejs-dropdownlist id="state"
[dataSource]="stateData"></ejs-dropdownlist>|

| **Fields for mapping** | **Property:** *fields* `
<input type="text" id="dropdown1" ej-dropdownlist [fields]="fieldsvalues">` | **Property:** *fields* `
<ejs-dropdownlist id="state" [fields]="stateFields"></ejs-dropdownlist>` |

| **Query** | **Property:** *query* `
<input type="text" id="dropdown1" ej-dropdownlist [query]="query">` | **Property:** *Query* `
<ejs-dropdownlist id="state" [query]='query'></ejs-dropdownlist>` |

| **Begin event** | **Event:** *actionBegin* `
<input type="text" id="dropdown1" ej-dropdownlist (actionBegin)="begin($event)">` | **Event:** *actionBegin* `
<ejs-dropdownlist id="state" (actionBegin)="begin($event)"></ejs-dropdownlist>` |

| **Complete event** | **Event:** *actionComplete* `
<input type="text" id="dropdown1" ej-dropdownlist (actionComplete)="empList($event)">` | **Event:** *actionComplete* `
<ejs-dropdownlist id="state" (actionComplete)="complete($event)"></ejs-dropdownlist>` |

| **Failure event** | **Event:** *actionFailure* `
<input type="text" id="dropdown1" ej-dropdownlist (actionFailure)="empList($event)">` | **Event:** *actionFailure* `
<ejs-dropdownlist id="state" (actionFailure)="failure($event)"></ejs-dropdownlist>` |

| **Success event** | **Event:** *actionSuccess* `
<input type="text" id="dropdown1" ej-dropdownlist (actionSuccess)="success($event)">` | **Not Applicable** |

| **Data binding event** | **Event:** *dataBound* `
<input type="text" id="dropdown1" ej-dropdownlist (dataBound)="databinding($event)">` | **Event:** *dataBind* `
<ejs-dropdownlist id="state" (dataBind)="dataBind($event)"></ejs-dropdownlist>` |

Filtering

<!-- markdownlint-disable MD010 -->

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

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

| **Default** | **Property:** *enableFilterSearch* `
<input type="text" id="dropdown1" ej-dropdownlist [enableFilterSearch]="enableFilterSearch">` | **Property:** *allowFiltering* `
<ejs-dropdownlist id="state" [allowFiltering]="allowFiltering"></ejs-dropdownlist>` |

| **Server filtering** | **Property:** *enableServerFiltering* `
<input type="text" id="dropdown1" ej-dropdownlist [enableServerFiltering]="enableServerFiltering">` | **Property:** *allowFiltering* `
<ejs-dropdownlist id="state" [allowFiltering]="allowFiltering"></ejs-dropdownlist>` |

| **Filter type** | **Property:** *filterType* `
<input type="text" id="dropdown1" ej-dropdownlist [filterType]="filtertype">` | <https://ej2.syncfusion.com/angular/demos/#/material/drop-down-list/filtering> |

| **No Records Template** | **Not Applicable** | **Property:** *noRecordsTemplate* `
<ejs-dropdownlist id="state" [noRecordsTemplate]="noRecordsTemplate"></ejs-dropdownlist>` |

| **Filter Bar watermark text** | **Not Applicable** | **Property:** *filterBarPlaceholder* `
<ejs-dropdownlist id="state" [filterBarPlaceholder]="filterBarPlace"></ejs-dropdownlist>` |

| **Ignore casing and diacritics** | **Not Applicable** | **Property:** *ignoreAccent*
 <ejs-dropdownlist id="state" [ignoreAccent]="ignoreAccent"></ejs-dropdownlist> |

| **Incremental search** | **Property:** *enableIncrementalSearch*
 <input type="text" id="dropdown1" ej-dropdownlist [enableIncrementalSearch]="enableIncrementalSearch"> | **By default it is true** |

| **Case sensitivity** | **Property:** *caseSensitiveSearch*
 <input type="text" id="dropdown1" ej-dropdownlist [caseSensitiveSearch]="caseSensitiveSearch"> |
<https://ej2.syncfusion.com/angular/demos/#/material/drop-down-list/filtering> |

| **Search event** | **Event:** *search*
 <input type="text" id="dropdown1" ej-dropdownlist (search)="search(\$event)"> | **Event:** *filtering*
 <ejs-dropdownlist id="state" (filtering)="filtering(\$event)"></ejs-dropdownlist> |

Template

<!-- markdownlint-disable MD010 -->

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

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

| **Default** | **Property:** *template*
 <input type="text" id="dropdown1" ej-dropdownlist [template]="template"> | **Property:** *itemTemplate*
 <ejs-dropdownlist id="state" [itemTemplate]="itemTemplate"></ejs-dropdownlist> |

| **Group Template** | **Not Applicable** | **Property:** *groupTemplate*
 <ejs-dropdownlist id="state" [groupTemplate]="groupTemplate"></ejs-dropdownlist> |

| **ValueTemplate** | **Not Applicable** | **Property:** *valueTemplate*
 <ejs-dropdownlist id="state" [valueTemplate]="valueTemplate"></ejs-dropdownlist> |

| **Header Template** | **Property:** *headerTemplate*
 <input type="text" id="dropdown1" ej-dropdownlist [headerTemplate]="headerTemplate"> | **Property:** *headerTemplate*
 <ejs-dropdownlist id="state" [headerTemplate]="headerTemplate"></ejs-dropdownlist> |

| **FooterTemplate** | **Not applicable** | **Property:** *footerTemplate*
 <ejs-dropdownlist id="state" [footerTemplate]="footerTemplate"></ejs-dropdownlist> |

| **No records Template** | **Not applicable** | **Property:** *noRecordsTemplate*
 <ejs-dropdownlist id="state" [noRecordsTemplate]="noRecordsTemplate"></ejs-dropdownlist> |

| **Action failure Template** | **Not applicable** | **Property:** *actionFailureTemplate*
 <ejs-dropdownlist id="state" [actionFailureTemplate]="actionFailureTemplate"></ejs-dropdownlist> |

Virtual Scrolling

<!-- markdownlint-disable MD010 -->

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

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

| **Default** | **Property:** *allowVirtualScrolling*
 <input type="text" id="dropdown1" ej-dropdownlist [allowVirtualScrolling]="caseSensitiveSearch"> | **Not applicable** |

Applying CSS

<!-- markdownlint-disable MD010 -->

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

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

| **Default** | **Property:** *cssClass*
<input type="text" id="dropdown1" ej-dropdownlist [cssClass]="customClass"> | **Property:** *cssClass*
<ejs-dropdownlist id="state" [cssClass]="cssClass"></ejs-dropdownlist>|

| **showRoundedCorner** | **Property:** *showRoundedCorner*
<input type="text" id="dropdown1" ej-dropdownlist [showRoundedCorner]="showRoundedCorner"> | **Property:** *cssClass*
<ejs-dropdownlist id="state" [cssClass]="cssClass"></ejs-dropdownlist>|

Sorting

<!-- markdownlint-disable MD010 -->

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

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

| **Default** | **Property:** *enableSorting*
<input type="text" id="dropdown1" ej-dropdownlist [enableSorting]="enableSorting"> | **Enabled only on using sortOrder Property** |

| **Order of sorting** | **Property:** *sortOrder*
<input type="text" id="dropdown1" ej-dropdownlist [sortOrder]="sortOrder"> | **Property:** *sortOrder*
<ejs-dropdownlist id="state" [sortOrder]="sortOrder"></ejs-dropdownlist>|

Popup

<!-- markdownlint-disable MD010 -->

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

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

| **Popup height** | **Property:** *popupHeight*
<input type="text" id="dropdown1" ej-dropdownlist [popupHeight]="popupHeight"> | **Property:** *popupHeight*
<ejs-dropdownlist id="state" [floatLabelType]="floatLabelType"></ejs-dropdownlist>|

| **Popup width** | **Property:** *popupWidth*
<input type="text" id="dropdown1" ej-dropdownlist [popupWidth]="popupWidth"> | **Property:** *popupWidth*
<ejs-dropdownlist id="state" [floatLabelType]="floatLabelType"></ejs-dropdownlist>|

| **Popup show on load** | **Property:** *showPopupOnLoad*
 <input type="text" id="dropdown1" ej-dropdownlist [showPopupOnLoad]="showPopupOnLoad">| **By default, the data load on demand.** |

| **enableAnimation** | **Property:** *enableAnimation*
<input type="text" id="dropdown1" ej-dropdownlist [enableAnimation]="enableAnimation">| **Not applicable** |

| **Popup resizing** | **Property:** *enablePopupResize*
<input type="text" id="dropdown1" ej-dropdownlist [enablePopupResize]="enablePopupResize">| **Not applicable** |

| **Maximum Popup height** | **Property:** *maxPopupHeight*
<input type="text" id="dropdown1" ej-dropdownlist [maxPopupHeight]="maxPopupHeight"> | **Not applicable** |

| **Minimum Popup height** | **Property:** *minPopupHeight*
<input type="text" id="dropdown1" ej-dropdownlist [minPopupHeight]="minPopupHeight">
}); | **Not applicable** |

| **Maximum Popup width** | **Property:** *maxPopupWidth*
<input type="text" id="dropdown1" ej-dropdownlist [maxPopupWidth]="maxPopupWidth"> | **Not applicable** |

| **Minimum Popup width** | **Property:** *minPopupWidth*
<input type="text" id="dropdown1" ej-dropdownlist [minPopupWidth]="minPopupWidth"> | **Not applicable** |

| **Loading data** | **Property:** *loadOnDemand*
<input type="text" id="dropdown1" ej-dropdownlist [loadOnDemand]="loadOnDemand"> | **By default, it is true** |

| **Popup showing manually** | **Method:** *showPopup*
<input type="text" id="dropdown1" ej-dropdownlist>

\$('#dropdown1').ejDropDownList('showPopup') | **Method:** *showPopup*
<ejs-dropdownlist id="state" #sample></ejs-dropdownlist>

@ViewChild('sample') public ddlObj: DropDownListComponent;

ddlObj.showPopup(); |

| **Popup hiding manually** | **Method:** *hidePopup*
<input type="text" id="dropdown1" ej-dropdownlist>

\$('#dropdown1').ejDropDownList('hidePopup') | **Method:** *hidePopup*
<ejs-dropdownlist id="state" #sample></ejs-dropdownlist>

@ViewChild('sample') public ddlObj: DropDownListComponent;

ddlObj.hidePopup(); |

| **Before Popup hide event** | **Event:** *beforePopupHide*
<input type="text" id="dropdown1" ej-dropdownlist (beforePopupHide)="beforePopupHide(\$event)"> | **Not applicable** |

| **Before Popup shown event** | **Event:** *beforePopupShown*
<input type="text" id="dropdown1" ej-dropdownlist (beforePopupShown)="beforePopupShown(\$event)"> | **Event:** *beforeOpen*
<input type="text" id="dropdown1" ej-dropdownlist (beforeOpen)="beforeOpen(\$event)"> |

| **Popup hide event** | **Event:** *popupHide*
<input type="text" id="dropdown1" ej-dropdownlist (popupHide)="popupHide(\$event)"> | **Event:** *close*
<input type="text" id="dropdown1" ej-dropdownlist (close)="close(\$event)"> |

| **Popup resize event** | **Event:** *popupResize*
<input type="text" id="dropdown1" ej-dropdownlist [popupResize]="popupResize(\$event)"> | **Not applicable** |

| **Popup resize start event** | **Event:** *popupResizeStart*
<input type="text" id="dropdown1" ej-dropdownlist (popupResizeStart)="popupResizeStart(\$event)"> | **Not applicable** |

| **Popup resize stop event** | **Event:** *popupResizeStop*
<input type="text" id="dropdown1" ej-dropdownlist (popupResizeStop)="popupResizeStop(\$event)"> | **Not applicable** |

| **Popup shown event** | **Event:** *popupShown*
<input type="text" id="dropdown1" ej-dropdownlist (popupShown)="popupShown(\$event)"> | **Event:** *open*
<input type="text" id="dropdown1" ej-dropdownlist (open)="open(\$event)"> |

Placeholder

<!-- markdownlint-disable MD010 -->

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

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

| **Watermark text** | **Property:** *watermarkText*
<input type="text" id="dropdown1" ej-dropdownlist [watermarkText]="watermarkText"> | **Property:** *placeholder*
<ejs-dropdownlist id="state" [placeholder]="placeholder"></ejs-dropdownlist> |

| **Floating of watermark text** | **Not applicable** | **Property:** *floatLabelType*
<ejs-dropdownlist id="state" [floatLabelType]="floatLabelType"></ejs-dropdownlist> |

Grouping

<!-- markdownlint-disable MD010 -->

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

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

| **Default** | **Property:** *fields.groupBy*
<input type="text" id="dropdown1" ej-dropdownlist [fields]="fields"> | **Property:** *fields.groupBy*
@Html.EJS().DropDownList("games").Fields(new DropDownListFieldSettings { GroupBy = "Game" }).Render() |

| **Group Template** | **Not applicable** | **Property:** *groupTemplate*
<ejs-dropdownlist id="state" [groupTemplate]="groupTemplate"></ejs-dropdownlist> |

Accessibility

<!-- markdownlint-disable MD010 -->

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

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

| **Globalization** | **Property:** *locale*
<input type="text" id="dropdown1" ej-dropdownlist [locale]="locale"> | **Property:** *locale*
<ejs-dropdownlist id="state" [locale]="locale"></ejs-dropdownlist> |

| **Rtl support** | **Property:** *enableRtl*
<input type="text" id="dropdown1" ej-dropdownlist [enableRtl]="enableRtl"> | **Property:** *enableRtl*
<ejs-dropdownlist id="state" [enableRtl]="enableRtl"></ejs-dropdownlist> |

Miscellaneous

<!-- markdownlint-disable MD010 -->

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

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

| **Enable/disable** | **Property:** *enabled*
<input type="text" id="dropdown1" ej-dropdownlist [enabled]="enabled"> | **Property:** *enabled*
<ejs-dropdownlist id="state" [enabled]="enabled"></ejs-dropdownlist> |

| Read only | **Property:** *readOnly*
<input type="text" id="dropdown1" ej-dropdownlist [readOnly]="readOnly"> |
Property: *readOnly*
<ejs-dropdownlist id="state" [readOnly]="readOnly"></ejs-dropdownlist> |

| Persistence of data | **Property:** *enablePersistence*
<input type="text" id="dropdown1" ej-dropdownlist [enablePersistence]="enablePersistence"> | **Property:** *enablePersistence*
<ejs-dropdownlist id="state" [enablePersistence]="enablePersistence"></ejs-dropdownlist> |

| **Disable** | **Method:** *disable*
<input type="text" id="dropdown1" ej-dropdownlist >

\$('#dropdown1').ejDropDownList('disable') | **Property:** *enabled*
<ejs-dropdownlist id="state" [enabled]="enabled"></ejs-dropdownlist> |

| **Enable** | **Method:** *enable*
<input type="text" id="dropdown1" ej-dropdownlist >

\$('#dropdown1').ejDropDownList('enable') | **Property:** *enabled*
<ejs-dropdownlist id="state" [enabled]="enabled"></ejs-dropdownlist> |

| **Height** | **Property:** *height*
<input type="text" id="dropdown1" ej-dropdownlist [height]="height"> | **Property:** *height*
<ejs-dropdownlist id="state" [height]="height"></ejs-dropdownlist> |

| **Width** | **Property:** *width*
<input type="text" id="dropdown1" ej-dropdownlist [width]="width"> | **Property:** *width*
<ejs-dropdownlist id="state" [width]="width"></ejs-dropdownlist> |

Selection

<!-- markdownlint-disable MD010 -->

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

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

| Selecting particular index | **Property:** *selectedIndex*
<input type="text" id="dropdown1" ej-dropdownlist [selectedIndex]="selectedIndex"> | **Property:** *index*
<ejs-dropdownlist id="state" [index]="index"></ejs-dropdownlist> |

| **Selecting particular value** | **Property:** *value*
<input type="text" id="dropdown1" ej-dropdownlist [value]="value"> | **Property:** *value*
<ejs-dropdownlist id="state" [value]="text"></ejs-dropdownlist> |

| **Selecting particular text** | **Property:** *text*
<input type="text" id="dropdown1" ej-dropdownlist [text]="text"> | **Property:** *text*
<ejs-dropdownlist id="state" [text]="text"></ejs-dropdownlist> |

| **Target id** | **Property:** *targetId*
<input type="text" id="dropdown1" ej-dropdownlist [targetId]="targetid"> | **Not applicable** |

| **Selecting item using text** | **Method:** *selectItemByText*
<input type="text" id="dropdown1" ej-dropdownlist >

\$('#dropdown1').ejDropDownList('selectItemByText', 'car') | **Not applicable** |

| **Unselect item using text** | **Method:** *unselectItemByText*
 ej-dropdownlist >

\$('#dropdown1').ejDropDownList('unselectItemByText','car') | **Not applicable** |

| **Selecting item using value** | **Method:** *selectItemByValue*
 ej-dropdownlist >

\$('#dropdown').ejDropDownList('selectItemByValue','car') | **Not applicable** |

| **Getting data by using value** | **Method:** *getItemDataByValue*
 id="dropdown1" ej-dropdownlist
 >

\$('#dropdown').ejDropDownList('unselectItemByValue','car') | **Method:**
getDataByValue
 <ejs-dropdownlist #sample id="state" ></ejs-
 dropdownlist>

@ViewChild('sample') public ddlObj:
 DropDownListComponent;

ddlObj.getItemDataByValue("data");|

| **Get selected value** | **Method:** *getSelectedItem*
 ej-dropdownlist >

\$('#dropdown').ejDropDownList('getSelectedItem') | **Not applicable** |

| **Get selected text** | **Method:** *getSelectedText*
 ej-dropdownlist >

\$('#dropdown').ejDropDownList('getSelectedText') | **Property:**
 text
<ejs-dropdownlist id="state" [text]="text"></ejs-dropdownlist> |

| **Select event** | **Event:** *select*
 ej-dropdownlist select="onSelect"> | **Event:** *select*
 ej-dropdownlist id="state" (select)="select(\$event)"></ejs-dropdownlist> |

| **Addition of Html attributes** | **Property:** *htmlAttributes*
 ej-dropdownlist [htmlAttribute]="attribute"> | **Property:** *htmlAttributes*
 ej-dropdownlist id="state" [htmlAttributes]="htmlAttributes"></ejs-dropdownlist> |

Common

<!-- markdownlint-disable MD010 -->

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

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

| **Adding new item** | **Method :** *addItem*
 ej-dropdownlist >

\$('#dropdown1').ejDropDownList("addItem", {text:"India"}); | **Method:**
addItem
 ej-dropdownlist #sample/>

@ViewChild('sample') public ddlObj:
 DropDownListComponent;

ddlObj.addItem({Id: 'game4', Game: 'Golf', 2}); |

| **Clearing the text** | **Method :** *clearText*
 ej-dropdownlist >

\$('#dropdown').ejDropDownList('clearText') | **Property:** *value*
 ej-dropdownlist id="state" [value]=""></ejs-dropdownlist> |

| **Destroy the component** | **Method :** *destroy*
 ej-dropdownlist >

\$('#dropdown1').ejDropDownList('destroy') | **Method:** *destroy*
 ej-dropdownlist #sample/>

@ViewChild('sample') public ddlObj:
 DropDownListComponent;

ddlObj.destroy; |

| **Getting the data** | **Method** : `getListData`
`<input type="text" id="dropdown1" ej-dropdownlist >`
`</>``{'#dropdown1').ejDropDownList('getListData')}` | **Method** : `getItems`
`<ejs-dropdownlist #sample/>`
`</>``@ViewChild('sample') public ddlObj: DropDownListComponent;`
`</>``ddlObj.getItems;` |

| **Create event** | **Event**: `create`
`<input type="text" id="dropdown1" ej-dropdownlist (create)="created($event)">` | **Event**: `created`
`<ejs-dropdownlist (created)="created($event)" />` |

| **Destroy event** | **Event**: `destroy`
`<input type="text" id="dropdown1" ej-dropdownlist (destroy)="destroy($event)">` | **Event**: `destroyed`
`<ejs-dropdownlist (destroyed)="destroy($event)" />` |

| **Cascade event** | **Event**: `cascade`
`<input type="text" id="dropdown1" ej-dropdownlist (cascade)="cascade($event)">` | <https://ej2.syncfusion.com/angular/demos/#/material/drop-down-list/cascading> |

| **Change event** | **Event**: `change`
`<input type="text" id="dropdown1" ej-dropdownlist (change)="change($event)">` | **Event**: `change`
`<ejs-dropdownlist (change)="change($event)" />` |

| **Focus out event** | **Event**: `focusOut`
`<input type="text" id="dropdown1" ej-dropdownlist (focusOut)="focusOut($event)">` | **Event**: `blur`
`<ejs-dropdownlist (blur)="blur($event)" />` |

| **Focus in event** | **Event**: `focusIn`
`<input type="text" id="dropdown1" ej-dropdownlist (focusIn)="focusIn($event)">` | **Event**: `focus`
`<ejs-dropdownlist (focus)="onfocus($event)" />` |

Dropdown Tree

Getting started with Angular Drop down tree component

This section explains you about how to create a simple **Dropdown Tree** component and configure its available functionalities in Angular.

Dependencies

The following list of dependencies are required to use the Dropdown Tree component in your application.

```
`javascript
|-- @syncfusion/ej2-angular-dropdowns
|-- @syncfusion/ej2-base
|-- @syncfusion/ej2-data
|-- @syncfusion/ej2-ng-base
|-- @syncfusion/ej2-dropdowns
|-- @syncfusion/ej2-lists
|-- @syncfusion/ej2-inputs
|-- @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 an Angular Application

Start a new Angular application using below Angular CLI command.

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

Installing Syncfusion DropDownTree 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:"20.2.38-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 Dropdown Tree Module

Import Dropdown Tree 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 DropDownTreeModule for the DropDownTree component
import { DropDownTreeModule } from '@syncfusion/ej2-angular-dropdowns';
import { AppComponent } from './app.component';
@NgModule({
  //declaration of ej2-angular-dropdowns module into NgModule
  imports: [ BrowserModule, DropDownTreeModule ],
  declarations: [ AppComponent ],
  bootstrap: [ AppComponent ]
})
export class AppModule { }
```

Adding CSS Reference

Add Dropdown Tree component's styles as given below in `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-navigations/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';
,
```

Note: If you want to refer the combined component styles, please make use of our [CRG](#) (Custom Resource Generator) in your application.

Add Dropdown Tree component

Modify the template in [src/app/app.component.ts] file to render the Dropdown Tree component. Add the Angular Dropdown Tree by using `ejs-dropdowntree` selector in `template` section of the `app.component.ts` file.

```
`typescript
import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
  template: <ejs-dropdowntree id='dropdowntree'></ejs-dropdowntree>
})
export class AppComponent {}
,
```

Binding data source

The Dropdown Tree component can load the data either from local data sources or remote data services. This can be done using the `dataSource` property that is a member of the `fields` property. The `dataSource` property supports array of JavaScript objects and `DataManager`. Here, an array of JSON values is passed to the Dropdown Tree component.

```
`typescript
import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
  // specifies the template string for the DropDownTree component
  template: <ejs-dropdowntree id='dropdowntree' [fields]='fields' placeholder='Select a Item'></ejs-dropdowntree>
})
export class AppComponent {
  constructor() {
  }
  // defined the array of data
  public data: { [key: string]: Object }[] = [
```

```
{
  nodeId: '01', nodeText: 'Music',
  nodeChild: [
    { nodeId: '01-01', nodeText: 'Gouttes.mp3' }
  ]
},
{
  nodeId: '02', nodeText: 'Videos', expanded: true,
  nodeChild: [
    { nodeId: '02-01', nodeText: 'Naturals.mp4' },
    { nodeId: '02-02', nodeText: 'Wild.mpeg' },
  ]
},
{
  nodeId: '03', nodeText: 'Documents',
  nodeChild: [
    { nodeId: '03-01', nodeText: 'Environment Pollution.docx' },
    { nodeId: '03-02', nodeText: 'Global Water, Sanitation, & Hygiene.docx' },
    { nodeId: '03-03', nodeText: 'Global Warming.ppt' },
    { nodeId: '03-04', nodeText: 'Social Network.pdf' },
    { nodeId: '03-05', nodeText: 'Youth Empowerment.pdf' },
  ]
},
];

//binding data source through fields property
public fields: Object = { dataSource: this.data, value: 'nodeId', text: 'nodeText', child: 'nodeChild' };
}
,
```

Run the application

After completing the configuration required to render a basic Dropdown Tree, run the following command to display the output in your default browser.

```
`javascript
ng serve --open
,
```

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 { DropDownTreeModule } from '@syncfusion/ej2-angular-dropdowns'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component } from '@angular/core';
@Component({
  imports: [
    FormsModule, ReactiveFormsModule, ButtonModule, DropDownTreeModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the DropDownTree component
  template: `<ejs-dropdowntree id='dropdowntree' [fields]='fields'
placeholder='Select a Item'></ejs-dropdowntree>`
})
export class AppComponent {
  constructor() {
  }
  // defined the array of data
  public data: { [key: string]: Object }[] = [
    {
      nodeId: '01', nodeText: 'Music',
      nodeChild: [
        { nodeId: '01-01', nodeText: 'Gouttes.mp3' }
      ]
    },
    {
      nodeId: '02', nodeText: 'Videos', expanded: true,
      nodeChild: [
        { nodeId: '02-01', nodeText: 'Naturals.mp4' },
        { nodeId: '02-02', nodeText: 'Wild.mpeg' },
      ]
    },
    {
      nodeId: '03', nodeText: 'Documents',
      nodeChild: [
        { nodeId: '03-01', nodeText: 'Environment Pollution.docx' },
        { nodeId: '03-02', nodeText: 'Global Water, Sanitation, &
Hygiene.docx' },
        { nodeId: '03-03', nodeText: 'Global Warming.ppt' },
        { nodeId: '03-04', nodeText: 'Social Network.pdf' },
        { nodeId: '03-05', nodeText: 'Youth Empowerment.pdf' },
      ]
    },
  ],
};
  //binding data source through fields property
  public fields: Object = { dataSource: this.data, value: 'nodeId', text:
'nodeText', child: 'nodeChild' };
}
```

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 Drop down tree component

The Dropdown Tree component provides an option to load the data either from local data sources or from remote data services. This can be done through `dataSource` property that is a member of the `fields` property. The `dataSource` property supports array of JavaScript objects and `DataManager`. It also supports different kinds of data services such as OData, OData V4, Web API, URL, and JSON with the help of `DataManager` adaptors.

Dropdown Tree has `load on demand` (Lazy load) option. It reduces the bandwidth size when consuming the huge data. By default, the `loadOnDemand` is set to false. By enabling this property, it loads first level items initially, and when parent item is expanded, loads the child items based on the `parentValue/child` member.

Local data

To bind local data to the Dropdown Tree, you can assign a JavaScript object array to the `dataSource` property.

The Dropdown Tree component requires three fields (Value, text, and parentValue) to render local data source. When mapper fields are not specified, it takes the default values as the mapping fields. Local data source can also be provided as an instance of the `DataManager`. It supports two kinds of local data binding methods.

- Hierarchical data
- Self-referential data

Hierarchical data

Dropdown Tree can be populated with the hierarchical data source that contains nested array of JSON objects. You can directly map the hierarchical data and the field members with corresponding key values from the hierarchical data to the `fields` property.

In the following example, **code**, **name**, and **countries** columns from the hierarchical data have been mapped to **value**, **text**, and **child** fields, respectively.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule, ReactiveFormsModule } from '@angular/forms'
import { DropDownTreeModule } from '@syncfusion/ej2-angular-dropdowns'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component } from '@angular/core';
@Component({
  imports: [
    FormsModule, ReactiveFormsModule, ButtonModule, DropDownTreeModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the DropDownTree component
```

```
template: `<ejs-dropdowntree id='dropdownTree' [fields]='fields'></ejs-  
dropdowntree>`  
})  
export class AppComponent {  
  constructor() {  
  }  
  public data?: { [key: string]: Object; }[] = [  
    {  
      code: 'AF', name: 'Africa', countries: [  
        { code: 'NGA', name: 'Nigeria' },  
        { code: 'EGY', name: 'Egypt' },  
        { code: 'ZAF', name: 'South Africa' }  
      ]  
    },  
    {  
      code: 'AS', name: 'Asia', expanded: true, countries: [  
        { code: 'CHN', name: 'China' },  
        { code: 'IND', name: 'India', selected: true },  
        { code: 'JPN', name: 'Japan' }  
      ]  
    },  
    {  
      code: 'EU', name: 'Europe', countries: [  
        { code: 'DNK', name: 'Denmark' },  
        { code: 'FIN', name: 'Finland' },  
        { code: 'AUT', name: 'Austria' }  
      ]  
    },  
    {  
      code: 'NA', name: 'North America', countries: [  
        { code: 'USA', name: 'United States of America' },  
        { code: 'CUB', name: 'Cuba' },  
        { code: 'MEX', name: 'Mexico' }  
      ]  
    },  
    {  
      code: 'SA', name: 'South America', countries: [  
        { code: 'BRA', name: 'Brazil' },  
        { code: 'COL', name: 'Colombia' },  
        { code: 'ARG', name: 'Argentina' }  
      ]  
    },  
    {  
      code: 'OC', name: 'Oceania', countries: [  
        { code: 'AUS', name: 'Australia' },  
        { code: 'NZL', name: 'New Zealand' },  
        { code: 'WSM', name: 'Samoa' }  
      ]  
    },  
    {  
      code: 'AN', name: 'Antarctica', countries: [  
        { code: 'BVT', name: 'Bouvet Island' },  
        { code: 'ATF', name: 'French Southern Lands' }  
      ]  
    }  
  ],  
};
```

```
public fields :Object = { dataSource: this.data, value: 'code', text:
'name', child: 'countries' };
}
```

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

Dropdown Tree can be populated from the self-referential data structure that contains array of JSON objects with `parentValue` mapping.

You can directly assign the self-referential data and map all the field members with corresponding key values from self-referential data to the `fields` property.

To render the root level items, specify the `parentValue` as null or no need to specify the `parentValue` in the `dataSource`.

In the following example, `id`, `pid`, `hasChild`, and `name` columns from self-referential data have been mapped to `value`, `parentValue`, `hasChildren`, and `text` fields, respectively.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule, ReactiveFormsModule } from '@angular/forms'
import { DropDownTreeModule } from '@syncfusion/ej2-angular-dropdowns'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component } from '@angular/core';
@Component({
  imports: [
    FormsModule, ReactiveFormsModule, ButtonModule, DropDownTreeModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the DropDownTree component
  template: `<ejs-dropdowntree id='dropdownTree' [fields]='fields'></ejs-
dropdowntree>`
})
export class AppComponent {
  constructor() {
  }
  // dataSource
  public data: { [key: string]: Object }[] = [
    { id: 1, name: 'Discover Music', hasChild: true, expanded: true },
    { id: 2, pid: 1, name: 'Hot Singles' },
    { id: 3, pid: 1, name: 'Rising Artists' },
    { id: 4, pid: 1, name: 'Live Music' },
    { id: 6, pid: 1, name: 'Best of 2017 So Far' },
    { id: 7, name: 'Sales and Events', hasChild: true },
    { id: 8, pid: 7, name: '100 Albums - $5 Each' },
    { id: 9, pid: 7, name: 'Hip-Hop and R&B Sale' },
    { id: 10, pid: 7, name: 'CD Deals' },
```

```
{ id: 11, name: 'Categories', hasChild: true },
{ id: 12, pid: 11, name: 'Songs' },
{ id: 13, pid: 11, name: 'Bestselling Albums' },
{ id: 14, pid: 11, name: 'New Releases' },
{ id: 15, pid: 11, name: 'Bestselling Songs' },
{ id: 16, name: 'MP3 Albums', hasChild: true },
{ id: 17, pid: 16, name: 'Rock' },
{ id: 18, pid: 16, name: 'Gospel' },
{ id: 19, pid: 16, name: 'Latin Music' },
{ id: 20, pid: 16, name: 'Jazz' },
{ id: 21, name: 'More in Music', hasChild: true },
{ id: 22, pid: 21, name: 'Music Trade-In' },
{ id: 23, pid: 21, name: 'Redeem a Gift Card' },
{ id: 24, pid: 21, name: 'Band T-Shirts' },
];
// defining fieldMapping
public fields :Object = { dataSource: this.data, value: 'id', text:
'name', parentValue:"pid", hasChildren: 'hasChild' };
}
```

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

Dropdown Tree can also be populated from a remote data service with the help of the **DataManager** control and **Query** property.

It supports different kinds of data services such as OData, OData V4, Web API, URL, and JSON with the help of **DataManager** adaptors.

You can assign service data as an instance of **DataManager** to the **dataSource**. To interact with remote data source, you must provide the endpoint **url**.

The **DataManager** that acts as an interface between the service endpoint and the Dropdown Tree requires the following information to interact with service endpoint properly.

- **DataManager->url**: Defines the service endpoint to fetch data.
- **DataManager->adaptor**: Defines the adaptor option. By default, **ODataAdaptor** is used for remote binding.

Adaptor is responsible for processing response and request from/to the service endpoint. The **@syncfusion/ej2-data** package provides some pre-defined adaptors designed to interact with service endpoints. They are,

- **UrlAdaptor**: Used to interact with remote services. This is the base adaptor for all remote based adaptors.
- **ODataAdaptor**: Used to interact with OData endpoints.

- **ODataV4Adaptor**: Used to interact with OData V4 endpoints.
- **WebApiAdaptor**: Used to interact with Web API created under OData standards.
- **WebMethodAdaptor**: Used to interact with web methods.

In the following example, **ODataV4Adaptor** is used to fetch data from the remote services. The **EmployeeID**, **FirstName**, and **EmployeeID** columns from the Employees table have been mapped to **value**, **text**, and **hasChildren** fields respectively for first level items.

The **OrderID**, **EmployeeID**, and **ShipName** columns from the orders table have been mapped to **value**, **parentValue**, and **text** fields respectively for second level items.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule, ReactiveFormsModule } from '@angular/forms'
import { DropDownTreeModule } from '@syncfusion/ej2-angular-dropdowns'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component } from '@angular/core';
import { Query, DataManager, ODataV4Adaptor } from '@syncfusion/ej2-data';
@Component({
  imports: [
    FormsModule, ReactiveFormsModule, ButtonModule, DropDownTreeModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the DropDownTree component
  template: `<ejs-dropdowntree id='dropdownTree' [fields]='fields'></ejs-dropdowntree>`
})
export class AppComponent {
  constructor() {
    // Use data manager to get dropdown tree data from remote source
    public data: Object = new DataManager({
      url: 'https://services.odata.org/V4/Northwind/Northwind.svc',
      adaptor: new ODataV4Adaptor,
      crossDomain: true,
    });
    // Set queries to filter and fetch remote data
    public query: Object = new
    Query().from('Employees').select('EmployeeID,FirstName,Title').take(5);
    public query1: Object = new
    Query().from('Orders').select('OrderID,EmployeeID,ShipName').take(5);
    public fields: Object = {
      dataSource: this.data, query: this.query, value: 'EmployeeID', text:
      'FirstName', hasChildren: 'EmployeeID',
      child: { dataSource: this.data, query: this.query1, value: 'OrderID',
      parentValue: 'EmployeeID', text: 'ShipName' }
    };
  }
}
```

MAIN.TS

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

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

Templates in Angular Drop down tree component

The Dropdown Tree provides support to customize each list item, header, and footer elements. It uses the Essential JS 2 Template engine to compile and render the elements properly.

Item template

The content of each list item within the Dropdown Tree can be customized with the help of [itemTemplate](#) property.

In the following sample, the Dropdown Tree list items are customized with employee information such as **name** and **job** using the **itemTemplate** property.

The template expression should be provided inside the `${...}` interpolation syntax and `{% raw %}{{...}}{% endraw %}` for ng-template .

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule, ReactiveFormsModule } from '@angular/forms'
import { DropDownTreeModule } from '@syncfusion/ej2-angular-dropdowns'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component, Inject, OnInit, ViewChild } from '@angular/core';
import { DropDownTreeComponent } from '@syncfusion/ej2-angular-dropdowns';
@Component({
  imports: [
    FormsModule, ReactiveFormsModule, ButtonModule, DropDownTreeModule
  ],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-dropdowntree id='dropdownTree' cssClass="custom"
[fields]='fields' [popupHeight]='height' [placeholder]='watermark'
[itemTemplate]='itemTemplate'></ejs-dropdowntree>`
})
export class AppComponent {
  public data: { [key: string]: Object }[] = [
    { id: 1, name: 'Steven Buchanan', eimg: '10', job: 'General Manager',
    hasChild: true, expanded: true, status:'busy' },
    { id: 2, pid: 1, name: 'Laura Callahan', eimg: '2', job: 'Product
Manager', hasChild: true, status:'online' },
    { id: 3, pid: 2, name: 'Andrew Fuller', eimg: '7', job: 'Team Lead',
    hasChild: true, status:'away' },
    { id: 4, pid: 3, name: 'Anne Dodsworth', eimg: '1', job: 'Developer',
    status:'busy' },
    { id: 10, pid: 3, name: 'Lilly', eimg: '5', job: 'Developer',
    status:'online' },
    { id: 5, pid: 1, name: 'Nancy Davolio', eimg: '4', job: 'Product
Manager', hasChild: true, status:'away' },
    { id: 6, pid: 5, name: 'Michael Suyama', eimg: '9', job: 'Team Lead',
    hasChild: true, status:'online' },
    { id: 7, pid: 6, name: 'Robert King', eimg: '8', job: 'Developer ',
    status:'online' },
```

```

    { id: 11, pid: 6, name: 'Mary', eimg: '6', job: 'Developer ',
      status:'away' },
    { id: 9, pid: 1, name: 'Janet Leverling', eimg: '3', job: 'HR',
      status:'online' }
  ];
  public fields: Object = { dataSource: this.data, text: 'name', value: 'id',
    parentValue: 'pid', hasChildren: 'hasChild' };
  public height: string = '300px';
  public watermark: string = 'Select an employee';
  public itemTemplate: string = '<div><span class="ename">${name} -
</span><span class="ejjob">${job}</span></div>';
}

```

MAIN.TS

```

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

```

Header template

The header element is shown statically at the top of the popup list items within the Dropdown Tree. A custom element can be placed as a header element using the [headerTemplate](#) property.

In the following sample, the header is customized with the custom element.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule, ReactiveFormsModule } from '@angular/forms'
import { DropDownTreeModule } from '@syncfusion/ej2-angular-dropdowns'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component, Inject, OnInit, ViewChild } from '@angular/core';
import { DropDownTreeComponent } from '@syncfusion/ej2-angular-dropdowns';
@Component({
  imports: [
    FormsModule, ReactiveFormsModule, ButtonModule, DropDownTreeModule
  ],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-dropdowntree id='dropdownTree' cssClass="custom"
[fields]='fields' [popupHeight]='height' [placeholder]='watermark'
[headerTemplate]='headerTemplate'></ejs-dropdowntree>`
})
export class AppComponent {
  public data: { [key: string]: Object }[] = [
    { id: 1, name: 'Steven Buchanan', eimg: '10', job: 'General Manager',
      hasChild: true, expanded: true, status:'busy' },
    { id: 2, pid: 1, name: 'Laura Callahan', eimg: '2', job: 'Product
Manager', hasChild: true, status:'online' },
    { id: 3, pid: 2, name: 'Andrew Fuller', eimg: '7', job: 'Team Lead',
      hasChild: true, status:'away' },
    { id: 4, pid: 3, name: 'Anne Dodsworth', eimg: '1', job: 'Developer',
      status:'busy' },
  ],
}

```

```

    { id: 10, pid: 3, name: 'Lilly', eimg: '5', job: 'Developer',
      status:'online' },
    { id: 5, pid: 1, name: 'Nancy Davolio', eimg: '4', job: 'Product
      Manager', hasChild: true, status:'away' },
    { id: 6, pid: 5, name: 'Michael Suyama', eimg: '9', job: 'Team Lead',
      hasChild: true, status:'online' },
    { id: 7, pid: 6, name: 'Robert King', eimg: '8', job: 'Developer ',
      status:'online' },
    { id: 11, pid: 6, name: 'Mary', eimg: '6', job: 'Developer ',
      status:'away' },
    { id: 9, pid: 1, name: 'Janet Leverling', eimg: '3', job: 'HR',
      status:'online' }
  ];
  public fields: Object = { dataSource: this.data, text: 'name', value: 'id',
    parentValue: 'pid', hasChildren: 'hasChild' };
  public height: string = '300px';
  public watermark: string = 'Select an employee';
  public headerTemplate: string = '<div class="head"> Employee List </div>';
}

```

MAIN.TS

```

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

```

Footer template

The Dropdown Tree has options to show a footer element at the bottom of the list items in the popup list. Here, you can place any custom element as a footer element using the [footerTemplate](#) property.

In the following sample, the footer element displays the total number of employees present in the Dropdown Tree.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule, ReactiveFormsModule } from '@angular/forms'
import { DropDownTreeModule } from '@syncfusion/ej2-angular-dropdowns'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component, Inject, OnInit, ViewChild } from '@angular/core';
import { DropDownTreeComponent } from '@syncfusion/ej2-angular-dropdowns';
@Component({
  imports: [
    FormsModule, ReactiveFormsModule, ButtonModule, DropDownTreeModule
  ],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-dropdowntree id='dropdownTree' cssClass="custom"
    [fields]='fields' [popupHeight]='height' [placeholder]='watermark'
    [footerTemplate]='footerTemplate'></ejs-dropdowntree>`
})
export class AppComponent {
  public data: { [key: string]: Object }[] = [

```

```

    { id: 1, name: 'Steven Buchanan', eimg: '10', job: 'General Manager',
      hasChild: true, expanded: true, status:'busy' },
    { id: 2, pid: 1, name: 'Laura Callahan', eimg: '2', job: 'Product
      Manager', hasChild: true, status:'online' },
    { id: 3, pid: 2, name: 'Andrew Fuller', eimg: '7', job: 'Team Lead',
      hasChild: true, status:'away' },
    { id: 4, pid: 3, name: 'Anne Dodsworth', eimg: '1', job: 'Developer',
      status:'busy' },
    { id: 10, pid: 3, name: 'Lilly', eimg: '5', job: 'Developer',
      status:'online' },
    { id: 5, pid: 1, name: 'Nancy Davolio', eimg: '4', job: 'Product
      Manager', hasChild: true, status:'away' },
    { id: 6, pid: 5, name: 'Michael Suyama', eimg: '9', job: 'Team Lead',
      hasChild: true, status:'online' },
    { id: 7, pid: 6, name: 'Robert King', eimg: '8', job: 'Developer ',
      status:'online' },
    { id: 11, pid: 6, name: 'Mary', eimg: '6', job: 'Developer ',
      status:'away' },
    { id: 9, pid: 1, name: 'Janet Leverling', eimg: '3', job: 'HR',
      status:'online' }
  ];
  public fields: Object = { dataSource: this.data, text: 'name', value: 'id',
    parentValue: 'pid', hasChildren: 'hasChild' };
  public height: string = '300px';
  public watermark: string = 'Select an employee';
  public footerTemplate: string = "<span class='foot'> Total number of
    employees: " + this.data.length + "</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 Dropdown Tree is supports to display custom design in the popup list content using the [noRecordsTemplate](#) property when no matches found on search.

In the following sample, popup list content displays the notification of no data available.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule, ReactiveFormsModule } from '@angular/forms'
import { DropDownTreeModule } from '@syncfusion/ej2-angular-dropdowns'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component, Inject, OnInit, ViewChild } from '@angular/core';
import { DropDownTreeComponent } from '@syncfusion/ej2-angular-dropdowns';
@Component({
  imports: [
    FormsModule, ReactiveFormsModule, ButtonModule, DropDownTreeModule
  ],
  standalone: true,

```

```

    selector: 'app-container',
    template: `<ejs-dropdowntree id='dropdownTree' [fields]='fields'
[popupHeight]='height' [placeholder]='watermark'
[noRecordsTemplate]='noRecordsTemplate'></ejs-dropdowntree>`
  })
  export class AppComponent {
    public data: { [key: string]: Object }[] = [ ];
    public fields: Object = { dataSource: this.data, text: 'name', value: 'id',
parentValue: 'pid', hasChildren: 'hasChild' };
    public height: string = '300px';
    public watermark: string = 'Select an employee';
    public noRecordsTemplate: Object = '<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));

```

Action failure template

The Dropdown Tree provides an option to custom design the popup list content using [actionFailureTemplate](#) property, when the data fetch request fails at the remote server.

In the following sample, when the data fetch request fails, the Dropdown Tree displays the notification.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule, ReactiveFormsModule } from '@angular/forms'
import { DropDownTreeModule } from '@syncfusion/ej2-angular-dropdowns'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component, Inject, OnInit, ViewChild } from '@angular/core';
import { DropDownTreeComponent } from '@syncfusion/ej2-angular-dropdowns';
import { Query, DataManager, ODataV4Adaptor } from '@syncfusion/ej2-data';
@Component({
  imports: [
    FormsModule, ReactiveFormsModule, ButtonModule, DropDownTreeModule
  ],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-dropdowntree id='dropdownTree' [placeholder]='placeholder'
[fields]='field' [actionFailureTemplate]='actionFailureTemplate'></ejs-
dropdowntree>`
})
export class AppComponent {
  public data: Object = new DataManager({
    url: 'https://services.odata.org/V4/Northwind/Northwind.svs',
    adaptor: new ODataV4Adaptor,
    crossDomain: true,
  });
  public query: Object = new
Query().from('Employees').select('EmployeeID,FirstName,Title').take(5);

```

```

    public query1: Object = new
Query().from('Orders').select('OrderID,EmployeeID,ShipName').take(5);
    public field: Object = {
        dataSource: this.data, query: this.query, value: 'EmployeeID', text:
'FirstName', hasChildren: 'EmployeeID',
        child: { dataSource: this.data, query: this.query1, value: 'OrderID',
parentValue: 'EmployeeID', text: 'ShipName' }
    };
    public actionFailureTemplate: Object = '<span class="action-failure">
Data fetch request fails<span>';
    public placeholder: string = 'Select an employee';
}

```

MAIN.TS

```

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

```

Custom template to show selected items in input

In Dropdown Tree, while selecting more than one items via checkbox or multi selection support, all the selected items will be displayed in the input. Instead of displaying all the selected item text, the custom template can be displayed by setting the the [mode](#) property as **Custom** and [customTemplate](#) property.

When the **mode** property is set as **Custom**, the Dropdown Tree displays the default template value (**\${value.length} item(s) selected**) like **1 item(s) selected** or **2 item(s) selected**. The default template can be customized by setting **customTemplate** property.

In the following sample, the Dropdown Tree is rendered with default value of the **customTemplate** property like “**1 item(s) selected** or **2 item(s) selected**”.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule, ReactiveFormsModule } from '@angular/forms'
import { DropDownTreeModule } from '@syncfusion/ej2-angular-dropdowns'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component, Inject, OnInit, ViewChild } from '@angular/core';
import { DropDownTreeComponent } from '@syncfusion/ej2-angular-dropdowns';
@Component({
  imports: [
    FormsModule, ReactiveFormsModule, ButtonModule, DropDownTreeModule
  ],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-dropdowntree id='dropdownTree' [popupHeight]='height'
[placeholder]='watermark' [showCheckBox]='showCheckBox' [mode]='mode'
[customTemplate]='customTemplate' [treeSettings]='treeSettings'
[fields]='fields'></ejs-dropdowntree>`
})
export class AppComponent {
  public data: { [key: string]: Object }[] = [

```

```

    { id: 1, name: 'Steven Buchanan', eimg: '10', job: 'General Manager',
      hasChild: true, expanded: true, status: 'busy' },
    { id: 2, pid: 1, name: 'Laura Callahan', eimg: '2', job: 'Product
      Manager', hasChild: true, status: 'online' },
    { id: 3, pid: 2, name: 'Andrew Fuller', eimg: '7', job: 'Team Lead',
      hasChild: true, status: 'away' },
    { id: 4, pid: 3, name: 'Anne Dodsworth', eimg: '1', job: 'Developer',
      status: 'busy' },
    { id: 10, pid: 3, name: 'Lilly', eimg: '5', job: 'Developer',
      status: 'online' },
    { id: 5, pid: 1, name: 'Nancy Davolio', eimg: '4', job: 'Product
      Manager', hasChild: true, status: 'away' },
    { id: 6, pid: 5, name: 'Michael Suyama', eimg: '9', job: 'Team Lead',
      hasChild: true, status: 'online' },
    { id: 7, pid: 6, name: 'Robert King', eimg: '8', job: 'Developer ',
      status: 'online' },
    { id: 11, pid: 6, name: 'Mary', eimg: '6', job: 'Developer ',
      status: 'away' },
    { id: 9, pid: 1, name: 'Janet Leverling', eimg: '3', job: 'HR',
      status: 'online' }
  ];
  public fields: Object = { dataSource: this.data, text: 'name', value: 'id',
    parentValue: 'pid', hasChildren: 'hasChild' };
  public height: string = '300px';
  public watermark: string = 'Select an employee';
  public showCheckBox = true;
  public mode = "Custom";
  public customTemplate = "${value.length} item(s) selected";
  public treeSettings: Object = { autoCheck: true };
}

```

MAIN.TS

```

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

```

In the following sample, the Dropdown Tree is rendered with custom value of the **customTemplate** property like **Selected items count: 2**.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule, ReactiveFormsModule } from '@angular/forms'
import { DropDownTreeModule } from '@syncfusion/ej2-angular-dropdowns'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component, Inject, OnInit, ViewChild } from '@angular/core';
import { DropDownTreeComponent } from '@syncfusion/ej2-angular-dropdowns';
@Component({
  imports: [
    FormsModule, ReactiveFormsModule, ButtonModule, DropDownTreeModule
  ],
  standalone: true,
  selector: 'app-container',

```



```

    template: `<ejs-dropdowntree id='dropdownTree' [popupHeight]='height'
[placeholder]='watermark' [showCheckBox]='showCheckBox' [mode]='mode'
[customTemplate]='customTemplate' [treeSettings]='treeSettings'
[fields]='fields'></ejs-dropdowntree>`
  })
  export class AppComponent {
    public data: { [key: string]: Object }[] = [
      { id: 1, name: 'Steven Buchanan', eimg: '10', job: 'General Manager',
        hasChild: true, expanded: true, status: 'busy' },
      { id: 2, pid: 1, name: 'Laura Callahan', eimg: '2', job: 'Product
        Manager', hasChild: true, status: 'online' },
      { id: 3, pid: 2, name: 'Andrew Fuller', eimg: '7', job: 'Team Lead',
        hasChild: true, status: 'away' },
      { id: 4, pid: 3, name: 'Anne Dodsworth', eimg: '1', job: 'Developer',
        status: 'busy' },
      { id: 10, pid: 3, name: 'Lilly', eimg: '5', job: 'Developer',
        status: 'online' },
      { id: 5, pid: 1, name: 'Nancy Davolio', eimg: '4', job: 'Product
        Manager', hasChild: true, status: 'away' },
      { id: 6, pid: 5, name: 'Michael Suyama', eimg: '9', job: 'Team Lead',
        hasChild: true, status: 'online' },
      { id: 7, pid: 6, name: 'Robert King', eimg: '8', job: 'Developer ',
        status: 'online' },
      { id: 11, pid: 6, name: 'Mary', eimg: '6', job: 'Developer ',
        status: 'away' },
      { id: 9, pid: 1, name: 'Janet Leverling', eimg: '3', job: 'HR',
        status: 'online' }
    ];
    public fields: Object = { dataSource: this.data, text: 'name', value: 'id',
      parentValue: 'pid', hasChildren: 'hasChild' };
    public height: string = '300px';
    public watermark: string = 'Select an employee';
    public showCheckBox = true;
    public mode = "Custom";
    public customTemplate = "Selected item(s) count: ${value.length} ";
    public treeSettings: Object = { autoCheck: true };
  }

```

MAIN.TS

```

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

```

Checkbox in Angular Drop down tree component

The Dropdown Tree component allows you to check more than one item from the tree without affecting the UI's appearance by enabling the `showCheckBox` property. When this property is enabled, checkbox appears before each item text in the popup.

In the following example, the `showCheckBox` property is enabled.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'

```

```

import { BrowserModule } from '@angular/platform-browser'
import { FormsModule, ReactiveFormsModule } from '@angular/forms'
import { DropDownTreeModule } from '@syncfusion/ej2-angular-dropdowns'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component } from '@angular/core';
@Component({
  imports: [
    FormsModule, ReactiveFormsModule, ButtonModule, DropDownTreeModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the DropDownTree component
  template: `<ejs-dropdowntree id='dropdownTree' [fields]='fields'
[showCheckBox]='true'></ejs-dropdowntree>`
})
export class AppComponent {
  constructor() {
  }
  // dataSource
  public data: { [key: string]: Object }[] = [
    { id: 1, name: 'Discover Music', hasChild: true, expanded: true },
    { id: 2, pid: 1, name: 'Hot Singles' },
    { id: 3, pid: 1, name: 'Rising Artists' },
    { id: 4, pid: 1, name: 'Live Music' },
    { id: 6, pid: 1, name: 'Best of 2017 So Far' },
    { id: 7, name: 'Sales and Events', hasChild: true },
    { id: 8, pid: 7, name: '100 Albums - $5 Each' },
    { id: 9, pid: 7, name: 'Hip-Hop and R&B Sale' },
    { id: 10, pid: 7, name: 'CD Deals' },
    { id: 11, name: 'Categories', hasChild: true },
    { id: 12, pid: 11, name: 'Songs' },
    { id: 13, pid: 11, name: 'Bestselling Albums' },
    { id: 14, pid: 11, name: 'New Releases' },
    { id: 15, pid: 11, name: 'Bestselling Songs' },
    { id: 16, name: 'MP3 Albums', hasChild: true },
    { id: 17, pid: 16, name: 'Rock' },
    { id: 18, pid: 16, name: 'Gospel' },
    { id: 19, pid: 16, name: 'Latin Music' },
    { id: 20, pid: 16, name: 'Jazz' },
    { id: 21, name: 'More in Music', hasChild: true },
    { id: 22, pid: 21, name: 'Music Trade-In' },
    { id: 23, pid: 21, name: 'Redeem a Gift Card' },
    { id: 24, pid: 21, name: 'Band T-Shirts' },
  ];
  // defining fieldMapping
  public fields :Object = { dataSource: this.data, value: 'id', text:
'name', parentValue:"pid", hasChildren: 'hasChild' };
}

```

MAIN.TS

```

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

```

Auto Check

By default, the checkbox state of the parent and child items in the Dropdown Tree will not be dependent over each other. If you need dependent checked state, then enable the `autoCheck` property which is a member of `treeSettings` property.

- If one or more child items are not in the checked state, then the parent item will be in the intermediate state.
- If all the child items are checked, then the parent item will also be in the checked state.
- If a parent item is checked, then all the child items will also be changed to the checked state.

In the following example, the `autoCheck` property is enabled.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule, ReactiveFormsModule } from '@angular/forms'
import { DropDownTreeModule } from '@syncfusion/ej2-angular-dropdowns'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component } from '@angular/core';
@Component({
  imports: [
    FormsModule, ReactiveFormsModule, ButtonModule, DropDownTreeModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the DropDownTree component
  template: `<ejs-dropdowntree id='dropdownTree' [fields]='fields'
[showCheckBox]='true' [treeSettings]='treeSettings'></ejs-dropdowntree>`
})
export class AppComponent {
  constructor() {
  }
  // dataSource
  public data: { [key: string]: Object }[] = [
    { id: 1, name: 'Discover Music', hasChild: true, expanded: true },
    { id: 2, pid: 1, name: 'Hot Singles' },
    { id: 3, pid: 1, name: 'Rising Artists' },
    { id: 4, pid: 1, name: 'Live Music' },
    { id: 6, pid: 1, name: 'Best of 2017 So Far' },
    { id: 7, name: 'Sales and Events', hasChild: true },
    { id: 8, pid: 7, name: '100 Albums - $5 Each' },
    { id: 9, pid: 7, name: 'Hip-Hop and R&B Sale' },
    { id: 10, pid: 7, name: 'CD Deals' },
    { id: 11, name: 'Categories', hasChild: true },
    { id: 12, pid: 11, name: 'Songs' },
    { id: 13, pid: 11, name: 'Bestselling Albums' },
    { id: 14, pid: 11, name: 'New Releases' },
    { id: 15, pid: 11, name: 'Bestselling Songs' },
    { id: 16, name: 'MP3 Albums', hasChild: true },
    { id: 17, pid: 16, name: 'Rock' },
    { id: 18, pid: 16, name: 'Gospel' },
    { id: 19, pid: 16, name: 'Latin Music' },
    { id: 20, pid: 16, name: 'Jazz' },
    { id: 21, name: 'More in Music', hasChild: true },
```

```

        { id: 22, pid: 21, name: 'Music Trade-In' },
        { id: 23, pid: 21, name: 'Redeem a Gift Card' },
        { id: 24, pid: 21, name: 'Band T-Shirts' },
    ];
    // defining fieldMapping
    public fields :Object = { dataSource: this.data, value: 'id', text:
'name', parentValue:"pid", hasChildren: 'hasChild' };
    // treeSettings
    public treeSettings: Object = { autoCheck: true }
}

```

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 Dropdown Tree component has in-built support to select all the tree items using Select All options in the header.

When the `showSelectAll` property is set to true, a checkbox will be displayed in the popup header that allows you to select or deselect all the tree items in the popup.

By default, `Select All` and `unSelect All` text values will be showcased along with the checkbox in the popup header to indicate the action to be performed on checking or unchecking the checkbox. You can customize these name attributes by using `selectAllText` and `unSelectAllText` properties respectively.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule, ReactiveFormsModule } from '@angular/forms'
import { DropDownTreeModule } from '@syncfusion/ej2-angular-dropdowns'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component } from '@angular/core';
@Component({
  imports: [
    FormsModule, ReactiveFormsModule, ButtonModule, DropDownTreeModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the DropDownTree component
  template: `<ejs-dropdowntree id='dropdownTree' [fields]='fields'
[showCheckBox]='true' [showSelectAll]='true' selectAllText='Check All'
unSelectAllText='UnCheck All'></ejs-dropdowntree>`
})
export class AppComponent {
  constructor() {
  }
  // dataSource
  public data: { [key: string]: Object }[] = [
    { id: 1, name: 'Discover Music', hasChild: true, expanded: true },
    { id: 2, pid: 1, name: 'Hot Singles' },

```

```

    { id: 3, pid: 1, name: 'Rising Artists' },
    { id: 4, pid: 1, name: 'Live Music' },
    { id: 6, pid: 1, name: 'Best of 2017 So Far' },
    { id: 7, name: 'Sales and Events', hasChild: true },
    { id: 8, pid: 7, name: '100 Albums - $5 Each' },
    { id: 9, pid: 7, name: 'Hip-Hop and R&B Sale' },
    { id: 10, pid: 7, name: 'CD Deals' },
    { id: 11, name: 'Categories', hasChild: true },
    { id: 12, pid: 11, name: 'Songs' },
    { id: 13, pid: 11, name: 'Bestselling Albums' },
    { id: 14, pid: 11, name: 'New Releases' },
    { id: 15, pid: 11, name: 'Bestselling Songs' },
    { id: 16, name: 'MP3 Albums', hasChild: true },
    { id: 17, pid: 16, name: 'Rock' },
    { id: 18, pid: 16, name: 'Gospel' },
    { id: 19, pid: 16, name: 'Latin Music' },
    { id: 20, pid: 16, name: 'Jazz' },
    { id: 21, name: 'More in Music', hasChild: true },
    { id: 22, pid: 21, name: 'Music Trade-In' },
    { id: 23, pid: 21, name: 'Redeem a Gift Card' },
    { id: 24, pid: 21, name: 'Band T-Shirts' },
  ];
  // defining fieldMapping
  public fields :Object = { dataSource: this.data, value: 'id', text:
'name', parentValue:"pid", hasChildren: 'hasChild' };
}

```

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 Drop down tree component

The Dropdown Tree component can be localized to any culture by defining the texts and messages of the Dropdown Tree in the corresponding culture. The default locale of the Dropdown Tree is **en** (English). The following table represents the default texts and messages of the Dropdown Tree in **en** culture.

|KEY|Text/Message|

|----|----|

|noRecordsTemplate|No records found|

|actionFailureTemplate|Request failed|

|overflowCountTemplate|+\${count} more..|

|totalCountTemplate|\${count} selected|

Accessibility in Angular Dropdown Tree component

The Dropdown Tree component followed the accessibility guidelines and standards, including [ADA](#), [Section 508](#), [WCAG 2.2](#) standards, and [WCAG roles](#) that are commonly used to evaluate accessibility.

The accessibility compliance for the Dropdown Tree component is outlined below.

| Accessibility Criteria | Compatibility |

| -- | -- |

| [WCAG 2.2](#) Support | |

| [Section 508](#) Support | |

| Screen Reader Support | |

| Right-To-Left Support | |

| Color Contrast | |

| Mobile Device Support | |

| Keyboard Navigation Support | |

| [Accessibility Checker](#) Validation | |

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

<style>

.post .post-content img {

display: inline-block;

margin: 0.5em 0;

}

</style>

<div> - All features of the component meet the requirement.</div>

<div> - Some features of the component do not meet the requirement.</div>

<div> - The component does not meet the requirement.</div>

[WAI-ARIA attributes](#)

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

| Attributes | Purpose |

--- ---
role=checkbox All list items are contained within the element.
aria-disabled Indicates element is perceivable but disabled.
aria-owns This attribute contains the ID of the popup list to indicate popup as a child element.
aria-haspopup Indicates whether the Dropdown Tree input element has a popup list or not.
aria-expanded Indicates the state of the popup list for Dropdown Tree and the parent node's expansion status for TreeView.
aria-activedescendent This attribute holds the ID of the active list item to focus its descendant child element.
aria-labelledby This attribute points to the element(s) labeling the element it's applied to.
aria-describedby This attribute points to the element(s) describing the one it's set on.
role=tree All tree nodes are contained within the element.
role=treeitem Specifies the role of each tree node in a selectable TreeView and its containment within the tree.
role=group Specifies the role of each parent node container in the TreeView.
role=checkbox Indicates checkbox control along with treeitem element.
aria-multiselectable Indicates whether the TreeView enables multiple selection or not.
aria-selected Indicates the selected node.
aria-level Indicates the level of node in TreeView.
aria-checked Indicates the current checked state of TreeView checkbox.
aria-label Indicates the contextual message for the TreeView checkbox and Dropdown Tree.
aria-activedescendant Identifies the currently active element when focusing on the TreeView.

Keyboard interaction

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

Interaction Keys	Description
----- -----	
Alt + Down	Opens the popup.
Alt + Up	Closes the popup.
Esc(Escape)	Closes the popup when it is in an open state.
Arrow Up	Goes to the previous item in the popup.
Arrow Down	Goes to the next item in the popup.

| Arrow Right | Expands the current item in the popup. |

| Arrow Left | Collapses the current item in the popup. |

| Home | Goes to the first item in the popup. |

| End | Goes to the last item in the popup. |

| Enter | Selects the focused item in the popup. |

| Space | Checks the current item in the popup. |

Ensuring accessibility

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

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

See also

- [Accessibility in Syncfusion Angular components](#)

FileManager

Getting started with Angular File manager component

This section explains how to create a simple **File Manager** component and its basic usage.

Prerequisites

To get started with **File Manager** component, ensure the compatible versions of Angular and Typescript.

- Angular : 4+
- Typescript : 2.6+

Setting up angular project

Angular provides the easiest way to set angular CLI projects using Angular CLI tool.

Install the CLI application globally to your machine by using following command.

```
`sh
```

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

```
`
```

To Create a new application, refer the below command

```
`sh
```

```
ng new syncfusion-angular-app
```

```
`
```

Navigate to the created project folder by using following command.

```
`sh
```



```
cd syncfusion-angular-app
```

```
,
```

Refer [Syncfusion Angular Getting Started](#) section to know more about setting up angular-cli project.

Adding Dependencies

The following list of dependencies are required to use the file manager component in your application.

```
`javascript
```

```
|-- @syncfusion/ej2-angular-filemanager
```

```
|-- @syncfusion/ej2-base
```

```
|-- @syncfusion/ej2-layouts
```

```
|-- @syncfusion/ej2-popups
```

```
|-- @syncfusion/ej2-data
```

```
|-- @syncfusion/ej2-inputs
```

```
|-- @syncfusion/ej2-lists
```

```
|-- @syncfusion/ej2-buttons
```

```
|-- @syncfusion/ej2-splitbuttons
```

```
|-- @syncfusion/ej2-navigations
```

```
|-- @syncfusion/ej2-grids
```

```
|-- @syncfusion/ej2-filemanager
```

```
,
```

Installing Syncfusion FileManager 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-filemanager](#) package to the application.

```
`bash
```

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

```
`bash
npm install @syncfusion/ej2-angular-filemanager@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-filemanager:"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 style sheet to the application

To render the file manager component, import the file manager and its dependent component's styles as given below in `[src/styles.css]`.

```
`css
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-navigations/styles/material.css';
@import '../node_modules/@syncfusion/ej2-layouts/styles/material.css';
@import '../node_modules/@syncfusion/ej2-grids/styles/material.css';
@import '../node_modules/@syncfusion/ej2-angular-filemanager/styles/material.css';
`
```

Note: To refer the combined component styles, use Syncfusion [CRG](#) (Custom Resource Generator) in your application.

Adding File Manager module

After installing the package, the file manager component module is available to configure into your application from installed syncfusion package.

Refer to the following snippet to import the file manager module in `app.module.ts` from the `@syncfusion/ej2-angular-filemanager`.

```
`typescript
```

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { FormsModule } from '@angular/forms';
import { HttpClientModule } from '@angular/http';
import { AppComponent } from './app.component';
// Imported syncfusion filemanager module from filemanager package
import { FileManagerModule } from '@syncfusion/ej2-angular-filemanager';
@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,
    FormsModule,
    HttpClientModule,
    // Registering EJ2 filemanager Module
    FileManagerModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
`
```

Adding Syncfusion component

Add the FileManager component by using `<ejs-filemanager>` selector in `template` section of the `app.component.ts` file.

Refer the FileManager component snippet in `app.component.ts` as follows.

`typescript

```
import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
  styleUrls: ['app/app.component.css'],
  template: `<ejs-filemanager id='default-filemanager' [ajaxSettings]='ajaxSettings'>
</ejs-filemanager>`
})
```

```
}}  
export class AppComponent {  
  public hostUrl: string = 'https://ej2-aspcore-service.azurewebsites.net/';  
  public ajaxSettings: object = {  
    url: this.hostUrl + 'api/FileManager/FileOperations'  
  };  
}  
`
```

Note: The [ajaxSettings](#) must be defined while initializing the file manager. File manager utilizes the URL's mentioned in ajaxSettings to send [file operation](#) request to the server.

The File Manager service link is given in `hostUrl`.

Run the application

Use the npm run start command to run the application in the browser.

```
`sh  
npm start  
`
```

The following samples shows the basic file manager component in browser.

APP.COMPONENT.TS

```
import { BrowserModule } from '@angular/platform-browser'  
import { NgModule } from '@angular/core'  
import { FileManagerModule } from '@syncfusion/ej2-angular-filemanager'  
import { Component } from '@angular/core';  
@Component({  
  imports: [FileManagerModule, ],  
  standalone: true,  
  selector: 'app-root',  
  styleUrls: ['./app.component.css'],  
  template: `<ejs-filemanager id='default-filemanager' #filemanagerObj  
[ajaxSettings]='ajaxSettings' [view]='view'>  
  </ejs-filemanager>`  
})  
export class AppComponent {  
  public view?: any;  
  public hostUrl: string = 'https://ej2-aspcore-service.azurewebsites.net/';  
  public ajaxSettings: object = {  
    url: this.hostUrl + 'api/FileManager/FileOperations'  
  };  
}
```

MAIN.TS

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

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

File Download support

To perform the download operation, initialize the `downloadUrl` property in a [ajaxSettings](#) of File Manager component.

```
`typescript
import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
  styleUrls: ['app/app.component.css'],
  template: `<ejs-filemanager id='default-filemanager' [ajaxSettings]='ajaxSettings'>
</ejs-filemanager>`
})
export class AppComponent {
  public hostUrl: string = 'https://ej2-aspcore-service.azurewebsites.net/';
  public ajaxSettings: object = {
    url: this.hostUrl + 'api/FileManager/FileOperations',
    downloadUrl: this.hostUrl + 'api/FileManager/Download'
  };
}
```

File Upload support

To perform the upload operation, initialize the `uploadUrl` property in a [ajaxSettings](#) of File Manager Component.

```
`typescript
import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
  styleUrls: ['app/app.component.css'],
  template: `<ejs-filemanager id='default-filemanager' [ajaxSettings]='ajaxSettings'>
</ejs-filemanager>`
})
export class AppComponent {
  public hostUrl: string = 'https://ej2-aspcore-service.azurewebsites.net/';
  public ajaxSettings: object = {
```

```
url: this.hostUrl + 'api/FileManager/FileOperations',
uploadUrl: this.hostUrl + 'api/FileManager/Upload'
};
}
`
```

Image Preview support

To perform the image preview support in the File Manager component, need to initialize the `getImageUrl` property in a `ajaxSettings` of File Manager component.

APP.COMPONENT.TS

```
import { BrowserModule } from '@angular/platform-browser'
import { NgModule } from '@angular/core'
import { FileManagerModule } from '@syncfusion/ej2-angular-filemanager'
import { Component } from '@angular/core';
@Component({
  imports: [FileManagerModule, ],
  standalone: true,
  selector: 'app-root',
  styleUrls: ['./app.component.css'],
  template: `<ejs-filemanager id='default-filemanager' #filemanagerObj
[ajaxSettings]='ajaxSettings' [view]='view'>
  </ejs-filemanager>`
})
export class AppComponent {
  public view?: any;
  public hostUrl: string = 'https://ej2-aspcore-service.azurewebsites.net/';
  public ajaxSettings: object = {
    url: this.hostUrl + 'api/FileManager/FileOperations',
    getImageUrl: this.hostUrl + 'api/FileManager/GetImage'
  };
}
```

MAIN.TS

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

Injecting feature modules

Basically, the file manager component contains a context menu for performing file operations, large-icons view for displaying the files and folders, and a breadcrumb for navigation. However, these basic functionalities can be extended by using the additional feature modules like toolbar, navigation pane, and details view to simplify the navigation and file operations within the file system. The above modules can be injected into file manager by importing them as providers in `app.module.ts`.

Refer the code snippet in for `app.module.ts`.

`typescript

```

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

import { FileManagerModule, NavigationPaneService, ToolbarService, DetailsViewService } from
 '@syncfusion/ej2-angular-filemanager';

import { AppComponent } from './app.component';

@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule, FileManagerModule
  ],
  providers: [NavigationPaneService, ToolbarService, DetailsViewService],
  bootstrap: [AppComponent]
})
export class AppModule { }

```

APP.COMPONENT.TS

```

import { BrowserModule } from '@angular/platform-browser'
import { NgModule } from '@angular/core'
import { FileManagerModule, NavigationPaneService, ToolbarService,
DetailsViewService } from '@syncfusion/ej2-angular-filemanager'
import { Component } from '@angular/core';
@Component({
  imports: [FileManagerModule, ],
  providers: [ NavigationPaneService, ToolbarService, DetailsViewService],
  standalone: true,
  selector: 'app-root',
  styleUrls: ['./app.component.css'],
  template: `<ejs-filemanager id='overview' [ajaxSettings]='ajaxSettings'
[toolbarSettings]='toolbarSettings'
[navigationPaneSettings]='navigationPaneSettings'></ejs-filemanager>`
})
export class AppComponent {
  public toolbarSettings?: any;
  public navigationPaneSettings?: any;
  public hostUrl: string = 'https://ej2-aspcore-service.azurewebsites.net/';
  public ajaxSettings: object = {
    url: this.hostUrl + 'api/FileManager/FileOperations',
    getImageUrl: this.hostUrl + 'api/FileManager/GetImage',
    uploadUrl: this.hostUrl + 'api/FileManager/Upload',
    downloadUrl: this.hostUrl + 'api/FileManager/Download'
  };
}

```

MAIN.TS

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

Note: The appearance of the File Manager can be customized by using [cssClass](#) property. This adds a css class to the root of the File Manager which can be used to add new styles or override existing styles to the File Manager.

Switching initial view of the File Manager

The initial view of the File Manager can be changed to details or largeicons view with the help of [view](#) property. By default, the File Manager will be rendered in large icons view.

When the File Manager is initially rendered, [created](#) will be triggered. This event can be utilized for performing operations once the File Manager has been successfully created.

APP.COMPONENT.TS

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { FileManagerModule, NavigationPaneService, ToolbarService,
DetailsViewService } from '@syncfusion/ej2-angular-filemanager';
import { Component } from '@angular/core';
@Component({
  imports: [FileManagerModule, ],
  providers: [ NavigationPaneService, ToolbarService, DetailsViewService],
  standalone: true,
  selector: 'app-root',
  styleUrls: ['./app.component.css'],
  template: `<ejs-filemanager id='file-manager'
[ajaxSettings]='ajaxSettings' [view]='view' (created)='onCreate($event)'>
  </ejs-filemanager>`
})
export class AppComponent {
  public ajaxSettings?: object;
  public view?: string;
  public hostUrl: string = 'https://ej2-aspcore-
service.azurewebsites.net/';
  public ngOnInit(): void {
    this.ajaxSettings = {
      url: this.hostUrl + 'api/FileManager/FileOperations',
      getImageUrl: this.hostUrl + 'api/FileManager/GetImage',
      uploadUrl: this.hostUrl + 'api/FileManager/Upload',
      downloadUrl: this.hostUrl + 'api/FileManager/Download'
    };
    // Initial view of File Manager is set to details view
    this.view = "Details";
  };
  // File Manager's created event function
  onCreate(args: any) {
    console.log("File Manager has been created successfully");
  }
}
```



```
}

```

MAIN.TS

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

Maintaining component state on page reload

The File Manager supports maintaining the component state on page reload. This can be achieved by enabling [enablePersistence](#) property which maintains the following,

- Previous view of the File Manager - [View](#)
- Previous path of the File Manager - [Path](#)
- Previous selected items of the File Manager - [SelectedItems](#)

For every operation in File Manager, ajax request will be sent to the server which then processes the request and sends back the response. When the ajax request is success, [success](#) event will be triggered and [failure](#) event will be triggered if the request gets failed.

APP.COMPONENT.TS

```
import { BrowserModule } from '@angular/platform-browser'
import { NgModule } from '@angular/core'
import { FileManagerModule, NavigationPaneService, ToolbarService,
DetailsViewService } from '@syncfusion/ej2-angular-filemanager'
import { Component } from '@angular/core';
@Component({
  imports: [FileManagerModule, ],
  providers:[ NavigationPaneService, ToolbarService, DetailsViewService],
  standalone: true,
  selector: 'app-root',
  styleUrls: ['./app.component.css'],
  template: `<ejs-filemanager id='file-manager'
[ajaxSettings]='ajaxSettings' [enablePersistence]='enablePersistence'
(success)='onAjaxSuccess($event)' (failure)='onAjaxFailure($event)'>
</ejs-filemanager>`
})
export class AppComponent {
  public ajaxSettings?: object;
  public enablePersistence?: boolean;
  public baseUrl: string = 'https://ej2-aspcore-
service.azurewebsites.net/';
  public ngOnInit(): void {
    this.ajaxSettings = {
      url: this.baseUrl + 'api/FileManager/FileOperations',
      getImageUrl: this.baseUrl + 'api/FileManager/GetImage',
      uploadUrl: this.baseUrl + 'api/FileManager/Upload',
      downloadUrl: this.baseUrl + 'api/FileManager/Download'
    };
    this.enablePersistence = true;
  };
  // File Manager's file onSuccess function
```

```

onAjaxSuccess(args: any) {
    console.log("Ajax request successful");
}
// File Manager's file onError function
onAjaxFailure(args: any) {
    console.log("Ajax request has failed");
}
}

```

MAIN.TS

```

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

```

Note: The files of the current folder opened in the File manager can be refreshed programatically by calling [refreshFiles](#) method.

Rendering component in right-to-left direction

It is possible to render the File Manager in right-to-left direction by setting the [enableRtl](#) API to true.

APP.COMPONENT.TS

```

import { BrowserModule } from '@angular/platform-browser'
import { NgModule } from '@angular/core'
import { FileManagerModule, NavigationPaneService, ToolbarService,
DetailsViewService } from '@syncfusion/ej2-angular-filemanager'
import { Component } from '@angular/core';
@Component({
  imports: [FileManagerModule, ],
  providers: [ NavigationPaneService, ToolbarService, DetailsViewService],
  standalone: true,
  selector: 'app-root',
  styleUrls: ['./app.component.css'],
  template: `<ejs-filemanager id='file-manager'
[ajaxSettings]='ajaxSettings' [enableRtl]='enableRtl'>
</ejs-filemanager>`
})
export class AppComponent {
  public ajaxSettings?: object;
  public enableRtl?: boolean;
  public hostUrl: string = 'https://ej2-aspcore-
service.azurewebsites.net/';
  public ngOnInit(): void {
    this.ajaxSettings = {
      url: this.hostUrl + 'api/FileManager/FileOperations',
      getImageUrl: this.hostUrl + 'api/FileManager/GetImage',
      uploadUrl: this.hostUrl + 'api/FileManager/Upload',
      downloadUrl: this.hostUrl + 'api/FileManager/Download'
    };
    this.enableRtl = true;
  };
}

```

MAIN.TS

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

Specifying the current path of the File Manager

The current path of the File Manager can be specified initially or dynamically using the [path](#) property.

The following code snippet demonstrates specifying the current path in File Manager on rendering.

`typescript

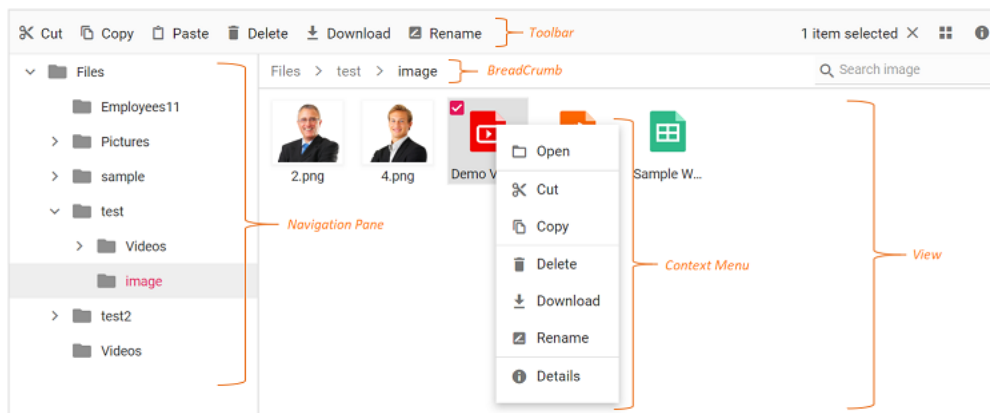
```
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  styleUrls: ['app/app.component.css'],
  template: `<ejs-filemanager id='default-filemanager' [ajaxSettings]='ajaxSettings'>
</ejs-filemanager>`
})
export class AppComponent {
  public ajaxSettings: object;
  public path: string;
  public hostUrl: string = 'https://ej2-aspcore-service.azurewebsites.net/';
  public ngOnInit(): void {
    this.ajaxSettings = {
      url: this.hostUrl + 'api/FileManager/FileOperations',
      getImageUrl: this.hostUrl + 'api/FileManager/GetImage',
      uploadUrl: this.hostUrl + 'api/FileManager/Upload',
      downloadUrl: this.hostUrl + 'api/FileManager/Download'
    };
    // Specify the required current path
    this.path = '/Food';
  }
}
```

User interface in Angular File manager component

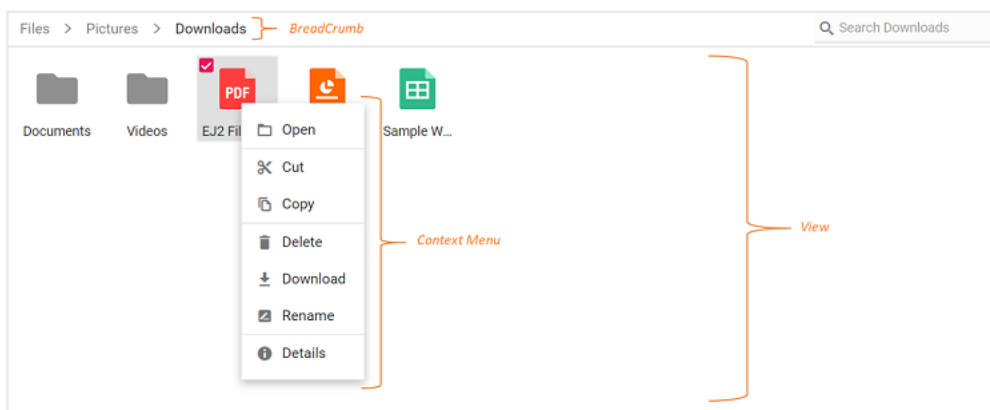
The file manager UI is comprised of several sections like view, toolbar, breadcrumb, context menu, and so on. The UI of the file manager is enhanced with injectable modules like **Details View** for browsing files and folders in a grid, **Navigation Pane** for folder navigation, and **Toolbar** for file operations. The file manager with all feature modules have the following sections in its UI.

- [View](#) (Large Icons view for browsing files and folders),
- [Toolbar](#) (For direct access to file operations),
- [Navigation Pane](#) (For easy navigation between folders),
- [Breadcrumb](#) (For parent folder navigations),
- [Context Menu](#) (For accessing file operations).



The basic file manager is a light weight component with all the basic functions. The basic file manager have the following sections in its UI to browse files and folders and manage them with file operations.

- [View](#) (Large Icons view for browsing files and folders),
- [Breadcrumb](#) (For parent folder navigations),
- [Context Menu](#) (For accessing file operations).

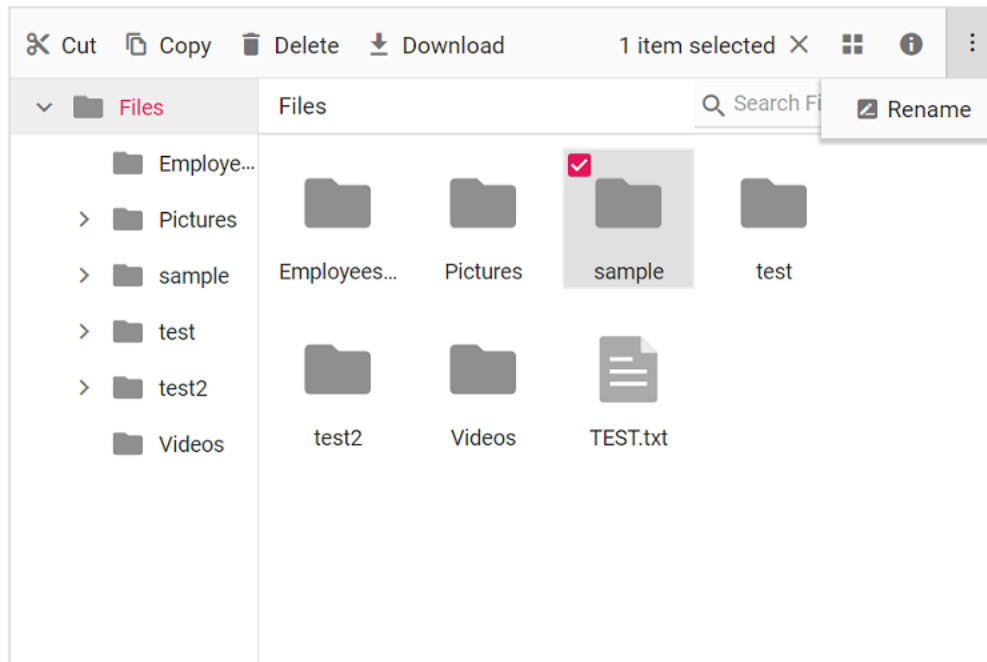


Toolbar

The toolbar is an injectable module in file manager. It should be injected before rendering the file manager to avail its functionality. It is present at the top of the file manager. Toolbar provides easy access to the file operations using different buttons.

If the toolbar items exceed the size of the toolbar, then the exceeding toolbar size will be moved to toolbar popup with a dropdown button at the end of toolbar.

**Refer [Toolbar](#) section in file operations to know more about the buttons present in toolbar*.*



Files and folders navigation

The file manager provides navigation between files and folders using the following two options.

- [Navigation Pane](#)
- [Breadcrumb](#)

Navigation pane

The navigation pane is an injectable module so, it should be injected before rendering the file manager to use its functionality.

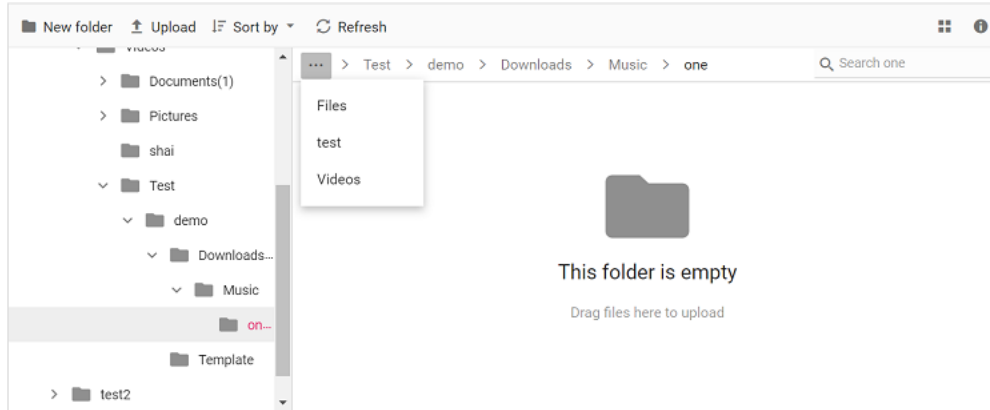
It displays the folder hierarchy of the file system and provides easy navigation to the desired folder. Using [navigationPaneSettings](#) minimum and maximum width of the navigation pane can be changed.

The navigation pane can be shown or hidden using the `visible` option in the [navigationPaneSettings](#).

BreadCrumb

The file manager provides breadcrumb for navigating to the parent folders. The breadcrumb in the file manager is responsible for resizing.

Whenever the path length exceeds the breadcrumb length, a dropdown button will be added at the starting of the breadcrumb to hold the parent folders adjacent to root.



View

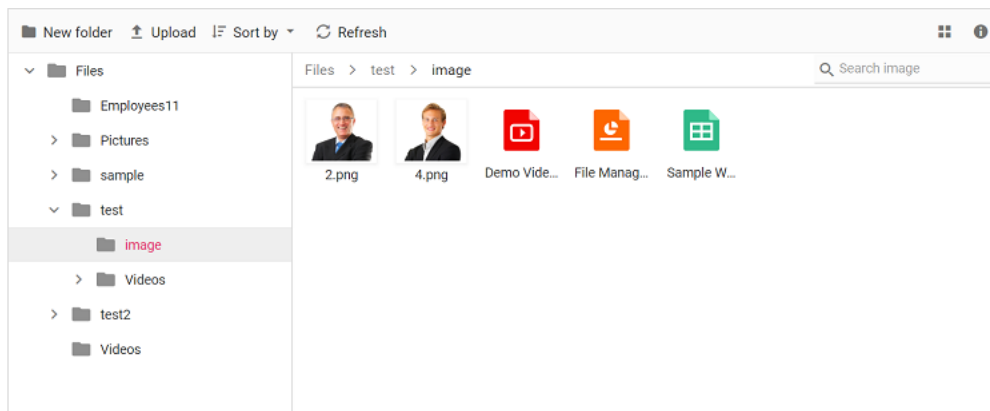
View is the section where the files and folders are displayed for the user to browse. The file manager has two types of views to display the files and folders.

- [Large Icons View](#)
- [Details View](#)

The **large icons view** is the default starting view in the file manager. The view can be changed by using the [toolbar](#) view button or by using the view menu in [context menu](#). The [view](#) API can also be used to change the initial view of the file manager.

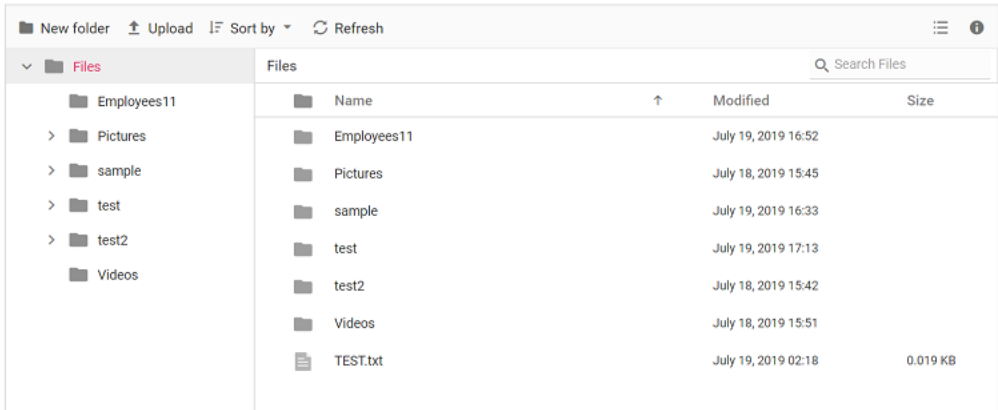
Large icons view

In the large icons view, the thumbnail icons will be shown in a larger size, which displays the data in a form that best suits their content. For image and video type files, a **preview** will be displayed. Extension thumbnails will be displayed for other type files.



Details view

Details view is an injectable module in the file manager so, it should be injected before rendering the file manager to avail its functionality. In the details view, the files are displayed in a sorted list order. This file list comprises of several columns of information about the files such as **Name**, **Date Modified**, **Type**, and **Size**. Each file has its own small icon representing the file type. Additional columns can be added using [detailsViewSettings](#) API. The details view allows you to perform sorting using column header.

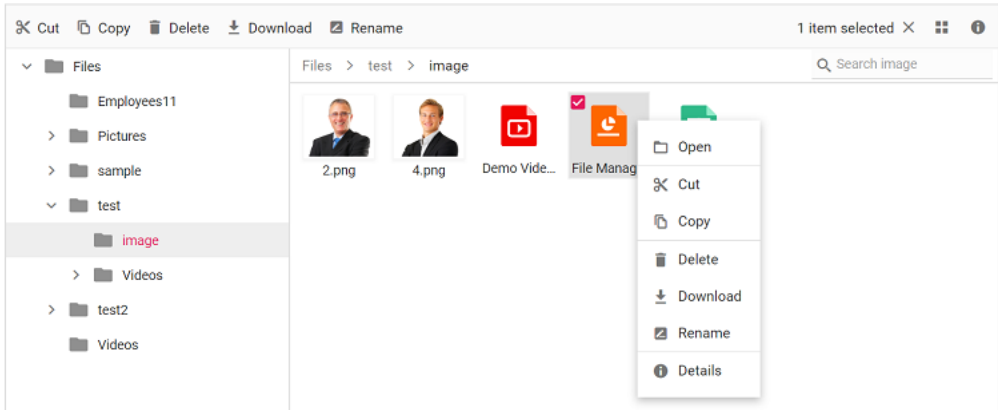


Context menu

The context menu appears on user interaction such as right-click. The file manager is provided with context menu support to perform list of file operations with the files and folders. Context menu appears with varying menu items based on the targets such as file, folder (including navigation pane folders), and layout (empty area in view).

Context menu can be customized using the [contextMenuSettings](#), [menuOpen](#), and [menuClick](#) events.

**Refer [Context Menu](#) section in file operations to know more about the menu items present in context menu*.*



File operations in Angular File manager component

The file manager component is used to browse, manage, and organize the files and folders in a file system through a web application. All basic file operations like creating a new folder, uploading and downloading of files in the file system, and deleting and renaming of existing files and folders are available in the file manager component. Additionally, previewing of image files is also provided in the file manager component.

The following table represents the basic operations available in the file manager and their corresponding functions.

|Operation Name|Function|

|----|----|

|read|Read the details of files or folders available in the given path from the file system, to display the files for the user to browse the content.|

|create| Creates a new folder in the current path of the file system. |

|delete| Removes the file or folder from the file server. |

|rename| Rename the selected file or folder in the file system. |

|search| Searches for items matching the search string in the current and child directories. |

|details| Gets the detail of the selected item(s) from the file server. |

|copy| Copy the selected file or folder in the file system. |

|move| Cut the selected file or folder in the file server. |

|upload| Upload files to the current path or directory in the file system. |

|download| Downloads the file from the server and the multiple files can be downloaded as ZIP files. |

The *CreateFolder*, *Remove*, and *Rename* actions will be reflected in the file manager only after the successful response from the server.

Folder Upload support

To perform the directory(folder) upload in File Manager, set [directoryUpload](#) as true within the `uploadSettings` property. The directory upload feature is supported for the following file service providers:

- Physical file service provider.
- Azure file service provider.
- NodeJS file service provider.
- Amazon file service provider.

In the following example, directory upload is enabled/disabled on DropDownButton selection.

APP.COMPONENT.TS

```
import { BrowserModule } from '@angular/platform-browser'
import { NgModule } from '@angular/core'
import { FileManagerModule, NavigationPaneService, ToolbarService,
DetailsViewService } from '@syncfusion/ej2-angular-filemanager'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import { FileManagerComponent, NavigationPaneService, ToolbarService,
DetailsViewService } from '@syncfusion/ej2-angular-filemanager';
import { DropDownButton, ItemModel } from '@syncfusion/ej2-splitbuttons';
/**
 * File Manager directory upload feature sample
 */
@Component({
  imports: [FileManagerModule, ],
  providers:[NavigationPaneService, ToolbarService, DetailsViewService],
  standalone: true,
  selector: 'app-root',
  styleUrls: ['./app.component.css'],
  template: `<ejs-filemanager id='file' #fileObj
[ajaxSettings]='ajaxSettings' (created)="onCreated($event)">
</ejs-filemanager>`
})
export class AppComponent{
  @ViewChild('fileObj')
```



```

public fileObj?: FileManagerComponent;
public ajaxSettings?: object;
public hostUrl: string = 'https://ej2-aspcore-
service.azurewebsites.net/';
public ngOnInit(): void {
    this.ajaxSettings = {
        url: this.hostUrl + 'api/FileManager/FileOperations',
        getImageUrl: this.hostUrl + 'api/FileManager/GetImage',
        uploadUrl: this.hostUrl + 'api/FileManager/Upload',
        downloadUrl: this.hostUrl + 'api/FileManager/Download'
    };
}
//DropDownButton items definition
public items: ItemModel[] = [{ text: 'Folder' }, { text: 'Files' }];
onCreated(args: any) {
    let customBtn: HTMLElement = document.getElementById('file_tb_upload') as
HTMLElement;
    customBtn.onclick = (e) => {
        e.stopPropagation();
    };
    let drpDownBtn: DropDownButton = new DropDownButton(
        {
            items: this.items,
            select: (args) => {
                if (args.item.text === 'Folder') {
                    (this.fileObj as
FileManagerComponent).uploadSettings.directoryUpload = true;
                } else {
                    (this.fileObj as
FileManagerComponent).uploadSettings.directoryUpload = false;
                }
                setTimeout(function () {
                    let uploadBtn: HTMLElement = document.querySelector(
                        '.e-file-select-wrap button'
                    ) as HTMLElement;
                    uploadBtn.click();
                }, 100);
            },
        },
        '#file_tb_upload'
    );
}
}

```

MAIN.TS

```

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

```

Physical file service provider

To achieve the directory upload in the physical file service provider, use the below code snippet in `IActionResult Upload` method in the `Controllers/FileManagerController.cs` file.

```
`typescript
[Route("Upload")]
public IActionResult Upload(string path, IList<IFormFile> uploadFiles, string action)
{
    FileManagerResponse uploadResponse;
    foreach (var file in uploadFiles)
    {
        var folders = (file.FileName).Split('/');
        // checking the folder upload
        if (folders.Length > 1)
        {
            for (var i = 0; i < folders.Length - 1; i++)
            {
                string newDirectoryPath = Path.Combine(this.basePath + path, folders[i]);
                if (!Directory.Exists(newDirectoryPath))
                {
                    this.operation.ToCamelCase(this.operation.Create(path, folders[i]));
                }
                path += folders[i] + "/";
            }
        }
        uploadResponse = operation.Upload(path, uploadFiles, action, null);
        if (uploadResponse.Error != null)
        {
            Response.Clear();
            Response.ContentType = "application/json; charset=utf-8";
            Response.StatusCode = Convert.ToInt32(uploadResponse.Error.Code);
            Response.HttpContext.Features.Get<IHttpResponseFeature>().ReasonPhrase =
                uploadResponse.Error.Message;
        }
        return Content("");
    }
}
```

`

Refer to the [GitHub](#) for more details

And also add the below code snippet in `FileManagerResponse Upload` method in `Models/PhysicalFileProvider.cs` file.

```
`typescript
string[] folders = name.Split('/');
string fileName = folders[folders.Length - 1];
var fullName = Path.Combine((this.contentRootPath + path), fileName);
```

`

Refer to the [GitHub](#) for more details.

Azure file service provider

For Azure file service provider, no customizations are needed for directory upload with server side and this will work with the below default upload method code.

Refer to the [GitHub](#) for more details.

NodeJS file service provider

To perform the directory upload in the NodeJS file service provider, use the below code snippet in `app.post` method in the `filesystem-server.js` file.

```
`typescript
var folders = (req.body.filename).split('/');
var filepath = req.body.path;
var uploadedFileName = folders[folders.length - 1];
// checking the folder upload
if (folders.length > 1)
{
  for (var i = 0; i < folders.length - 1; i++)
  {
    var newDirectoryPath = path.join(contentRootPath + filepath, folders[i]);
    if (!fs.existsSync(newDirectoryPath)) {
      fs.mkdirSync(newDirectoryPath);
      (async () => {
        await FileManagerDirectoryContent(req, res, newDirectoryPath).then(data => {
          response = { files: data };
          response = JSON.stringify(response);
        });
      })();
    }
  }
}
```

```
}
filepath += folders[i] + "/";
}
fs.rename('./' + uploadedFileName, path.join(contentRootPath, filepath + uploadedFileName), function
(err) {
if (err) {
if (err.code !== 'EBUSY') {
errorValue.message = err.message;
errorValue.code = err.code;
}
}
});
}
`
```

Refer to the [GitHub](#) for more details.

Amazon file service provider

To perform the directory upload in the Amazon file service provider, use the below code snippet in `IActionResult AmazonS3Upload` method in the `Controllers/AmazonS3ProviderController.cs` file.

```
`typescript
foreach (var file in uploadFiles)
{
var folders = (file.FileName).Split('/');
// checking the folder upload
if (folders.Length > 1)
{
for (var i = 0; i < folders.Length - 1; i++)
{
if (!this.operation.checkFileExist(path, folders[i]))
{
this.operation.ToCamelCase(this.operation.Create(path, folders[i], dataObject));
}
path += folders[i] + "/";
}
}
}
```

```
}  
`
```

Refer to the [GitHub](#) for more details.

And also add the below code snippet in `AsyncUpload` method in `Models/AmazonS3FileProvider.cs` file.

```
`typescript  
string[] folders = file.FileName.Split('/');  
string name = folders[folders.Length - 1];  
`
```

Refer to the [GitHub](#) for more details.

File operation request and response Parameters

The default parameters available in file operation request from the file manager and the corresponding response parameters required by the file manager are listed as follows.

Read

The following table represents the request parameters of *read* operations.

Parameter	Type	Default	Explanation
---- ---- ---- ----			
action	String	read	Name of the file operation.
path	String	-	Relative path from which the data has to be read.
showHiddenItems	Boolean	-	Defines show or hide the hidden items.
data	FileManagerDirectoryContent	-	Details about the current path (directory).

**Refer [File request and response contents](#) for the contents of data*.*

Example:

```
`typescript  
{  
  action: "read",  
  path: "/",  
  showHiddenItems: false,  
  data: []  
}  
`
```

The following table represents the response parameters of *read* operations.

Parameter	Type	Default	Explanation
---- ---- ---- ----			

|cwd|FileManagerDirectoryContent|-|Path (Current Working Directory) details.|

|files|FileManagerDirectoryContent[]|-|Details of files and folders present in given path or directory.|

|error|ErrorDetails|-|Error Details|

**Refer [File request and response contents](#) for the contents of cwd, files, and error*.*

Example:

```
`typescript
{
  cwd:
  {
    name:"Download",
    size:0,
    dateModified:"2019-02-28T03:48:19.8319708+00:00",
    dateCreated:"2019-02-27T17:36:15.812193+00:00",
    hasChild:false,
    isFile:false,
    type:"",
    filterPath:"\\Download\\"
  },
  files:[
  {
    name:"Sample Work Sheet.xlsx",
    size:6172,
    dateModified:"2019-02-27T17:23:50.9651206+00:00",
    dateCreated:"2019-02-27T17:36:15.8151955+00:00",
    hasChild:false,
    isFile:true,
    type:".xlsx",
    filterPath:"\\Download\\"
  }
  ],
  error:null,
  details:null
}
```

`

Create

The following table represents the request parameters of *create* operations.

Parameter	Type	Default	Explanation
action	String	create	Name of the file operation.
path	String	-	Relative path in which the folder has to be created.
name	String	-	Name of the folder to be created.
data	FileManagerDirectoryContent	-	Details about the current path (directory).

Refer [File request and response contents](#) for the contents of data

Example:

```
`typescript
{
  action: "create",
  data: [
    {
      dateCreated: "2019-02-27T17:36:15.6571949+00:00",
      dateModified: "2019-03-12T10:17:31.8505975+00:00",
      filterPath: "\",
      hasChild: true,
      isFile: false,
      name: files,
      nodeId: "fe_tree",
      size: 0,
      type: ""
    }
  ],
  name: "Hello",
  path: "/"
}
```

The following table represents the response parameters of *create* operations.

Parameter	Type	Default	Explanation
-----------	------	---------	-------------

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

|files|FileManagerDirectoryContent[]|-|Details of the created folder|

|error|ErrorDetails|-|Error Details|

**Refer [File request and response contents](#) for the contents of files and error*.*

Example:

```
`typescript
{
  cwd: null,
  files: [
    {
      dateCreated: "2019-03-15T10:25:05.3596171+00:00",
      dateModified: "2019-03-15T10:25:05.3596171+00:00",
      filterPath: null,
      hasChild: false,
      isFile: false,
      name: "New",
      size: 0,
      type: ""
    }
  ],
  details: null,
  error: null
}
```

Rename

The following table represents the request parameters of *rename* operations.

Parameter	Type	Default	Explanation
-----------	------	---------	-------------

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

action	String	rename	Name of the file operation.
--------	--------	--------	-----------------------------

path	String	-	Relative path in which the item is located.
------	--------	---	---------------------------------------------

name	String	-	Current name of the item to be renamed.
------	--------	---	-----------------------------------------

newname	String	-	New name for the item.
---------	--------	---	------------------------

data	FileManagerDirectoryContent	-	Details of the item to be renamed.
------	-----------------------------	---	------------------------------------

**Refer [File request and response contents](#) for the contents of data*.*

Example:

```
`typescript
{
  action: "rename",
  data: [
    {
      dateCreated: "2019-03-20T05:22:34.621Z",
      dateModified: "2019-03-20T08:45:56.000Z",
      filterPath: "\\Pictures\\Nature\\",
      hasChild: false,
      iconClass: "e-fe-image",
      isFile: true,
      name: "seaviews.jpg",
      size: 95866,
      type: ".jpg"
    }
  ],
  newname: "seaview.jpg",
  name: "seaviews.jpg",
  path: "/Pictures/Nature/"
}
```

The following table represents the response parameters of *rename* operations.

Parameter	Type	Default	Explanation
files	FileManagerDirectoryContent[]	-	Details of the renamed item.
error	ErrorDetails	-	Error Details

**Refer [File request and response contents](#) for the contents of files and error*.*

Example:

```
`typescript
{
  cwd:null,
```

```
files:[
{
name:"seaview.jpg",
size:95866,
dateModified:"2019-03-20T08:45:56+00:00",
dateCreated:"2019-03-20T05:22:34.6214847+00:00",
hasChild:false,
isFile:true,
type:".jpg",
filterPath:"\\Pictures\\Nature\\seaview.jpg"
}
],
error:null,
details:null
},
`
```

Delete

The following table represents the request parameters of *delete* operations.

Parameter	Type	Default	Explanation
----	----	----	----
action	String	delete	Name of the file operation.
path	String	-	Relative path where the items to be deleted are located.
names	String[]	-	List of the items to be deleted.
data	FileManagerDirectoryContent	-	Details of the item to be deleted.

**Refer [File request and response contents](#) for the contents of data*.*

Example:

```
`typescript
{
action: "delete",
path: "/Hello/",
names: ["New"],
data: []
}
```

,

The following table represents the response parameters of *delete* operations.

Parameter	Type	Default	Explanation
files	FileManagerDirectoryContent[]	-	Details about the deleted item(s).
error	ErrorDetails	-	Error Details

**Refer [File request and response contents](#) for the contents of files and error*.*

Example:

```
`typescript
{
  cwd: null,
  details: null,
  error: null,
  files: [
    {
      dateCreated: "2019-03-15T10:13:30.346309+00:00",
      dateModified: "2019-03-15T10:13:30.346309+00:00",
      filterPath: "\\Hello\\folder",
      hasChild: true,
      isFile: false,
      name: "folder",
      size: 0,
      type: ""
    }
  ]
}
```

Details

The following table represents the request parameters of *details* operations.

Parameter	Type	Default	Explanation
action	String	details	Name of the file operation.
path	String	-	Relative path where the items are located.

|names|String[]|-|List of the items to get details.|

|data|FileManagerDirectoryContent|-|Details of the selected item.|

**Refer [File request and response contents](#) for the contents of data*.*

Example:

```
`typescript
{
  action: "details",
  path: "/FileContents/",
  names: ["All Files"],
  data: []
}
```

The following table represents the response parameters of *details* operations.

Parameter	Type	Default	Explanation
-----------	------	---------	-------------

----	----	----	----
------	------	------	------

details	FileManagerDirectoryContent -	Details of the requested item(s).
---------	-------------------------------	-----------------------------------

error	ErrorDetails -	Error Details
-------	----------------	---------------

**Refer [File request and response contents](#) for the contents of details and error*.*

Example:

```
`typescript
{
  cwd:null,
  files:null,
  error:null,
  details:
  {
    name:"All Files",
    location:"\\Files\\FileContents\\All Files",
    isFile:false,
    size:"679.8 KB",
    created:"3/8/2019 10:18:37 AM",
    modified:"3/8/2019 10:18:39 AM",
    multipleFiles:false
  }
}
```

```
}  
}  
`
```

Search

The following table represents the request parameters of *search* operations.

Parameter	Type	Default	Explanation
----	----	----	----
action	String	search	Name of the file operation.
path	String	-	Relative path to the directory where the files should be searched.
showHiddenItems	Boolean	-	Defines show or hide the hidden items.
caseSensitive	Boolean	-	Defines search is case sensitive or not.
searchString	String	-	String to be searched in the directory.
data	FileManagerDirectoryContent	-	Details of the searched item.

Example:

```
`typescript  
{  
  action: "search",  
  path: "/",  
  searchString: "nature",  
  showHiddenItems: false,  
  caseSensitive: false,  
  data: []  
}  
`
```

The following table represents the response parameters of *search* operations.

Parameter	Type	Default	Explanation
----	----	----	----
cwd	FileManagerDirectoryContent	-	Path (Current Working Directory) details.
files	FileManagerDirectoryContent[]	-	Files and folders in the searched directory that matches the search input.
error	ErrorDetails	-	Error Details

Refer [File request and response contents](#) for the contents of cwd, files and error.

Example:

```
`typescript
```

```
{
  cwd:
  {
    name:files,
    size:0,
    dateModified:"2019-03-15T10:07:00.8658158+00:00",
    dateCreated:"2019-02-27T17:36:15.6571949+00:00",
    hasChild:true,
    isFile:false,
    type:"",
    filterPath:"\\"
  },
  files:[
    {
      name:"Nature",
      size:0,
      dateModified:"2019-03-08T10:18:42.9937708+00:00",
      dateCreated:"2019-03-08T10:18:42.5907729+00:00",
      hasChild:true,
      isFile:false,
      type:"",
      filterPath:"\\FileContents\\Nature"
    }
  ],
  error:null,
  details:null
}
```

Copy

The following table represents the request parameters of *copy* operations.

Parameter	Type	Default	Explanation
-----------	------	---------	-------------

----	----	----	----
------	------	------	------

action	String	copy	Name of the file operation.
--------	--------	------	-----------------------------

path	String	-	Relative path to the directory where the files should be copied.
names	String[]	-	List of files to be copied.
targetPath	String	-	Relative path where the items to be pasted are located.
data	FileManagerDirectoryContent	-	Details of the copied item.
renameFiles	String[]	-	Details of the renamed item.

Example:

```
`typescript
{
  action: "copy",
  path: "/",
  names: ["6.png"],
  renameFiles: ["6.png"],
  targetPath: "/Videos/"
}
```

The following table represents the response parameters of *copy* operations.

Parameter	Type	Default	Explanation
----	----	----	----
cwd	FileManagerDirectoryContent	-	Path (Current Working Directory) details.
files	FileManagerDirectoryContent[]	-	Details of copied files or folders
error	ErrorDetails	-	Error Details

Refer [File request and response contents](#) for the contents of cwd, files and error.

Example:

```
`typescript
{
  cwd:null,
  files:[
    {
      path:null,
      action:null,
      newName:null,
      names:null,
      name:"justin.mp4",
```

```
size:0,
previousName:"album.mp4",
dateModified:"2019-06-21T06:58:32+00:00",
dateCreated:"2019-06-24T04:22:14.6245618+00:00",
hasChild:false,
isFile:true,
type:".mp4",
id:null,
filterPath:"\\"
}
],
error:null,
details:null
}
`
```

Move

The following table represents the request parameters of *move* operations.

Parameter	Type	Default	Explanation
----	----	----	----
action	String	move	Name of the file operation.
path	String	-	Relative path to the directory where the files should be copied.
names	String[]	-	List of files to be moved.
targetPath	String	-	Relative path where the items to be pasted are located.
data	FileManagerDirectoryContent	-	Details of the moved item.
renameFiles	String[]	-	Details of the renamed item.

Example:

```
`typescript
{
  action: "move",
  path: "/",
  names: ["6.png"],
  renameFiles: ["6.png"],
  targetPath: "/Videos/"
```



```
}  
`
```

The following table represents the response parameters of *copy* operations.

Parameter	Type	Default	Explanation
----	----	----	----
cwd	FileManagerDirectoryContent	-	Path (Current Working Directory) details.
files	FileManagerDirectoryContent[]	-	Details of cut files or folders
error	ErrorDetails	-	Error Details

Refer [File request and response contents](#) for the contents of cwd, files and error.

Example:

```
`typescript  
{  
  cwd:null,  
  files:[  
    {  
      path:null,  
      action:null,  
      newName:null,  
      names:null,  
      name:"justin biber.mp4",  
      size:0,  
      previousName:"justin biber.mp4",  
      dateModified:"2019-06-21T06:58:32+00:00",  
      dateCreated:"2019-06-24T04:26:49.2690476+00:00",  
      hasChild:false,  
      isFile:true,  
      type:".mp4",  
      id:null,  
      filterPath:"\\Videos\\"  
    }  
  ],  
  error:null,  
  details:null
```

```
}  
`
```

Upload

The following table represents the request parameters of *Upload* operations.

Parameter	Type	Default	Explanation
action	String	Save	Name of the file operation.
path	String	-	Relative path to the location where the file has to be uploaded.
uploadFiles	IList<IFormFile>	-	File that are uploaded.

Example:

```
`typescript  
uploadFiles: (binary),  
path: /,  
action: Save,  
data: {  
  path:null,  
  action:null,  
  newName:null,  
  names:null,  
  name:"Downloads",  
  size:0,  
  previousName:null,  
  dateModified:"2019-07-22T11:23:46.7153977 00:00",  
  dateCreated:"2019-07-22T11:26:13.9047229 00:00",  
  hasChild:false,  
  isFile:false,  
  type:"",  
  id:null,  
  filterPath:"\\",  
  targetPath:null,  
  renameFiles:null,  
  uploadFiles:null,  
  caseSensitive:false,
```

```
searchString:null,  
showHiddenItems:false,  
fmiconClass:null,  
fmid:"fetree1",  
fmpld:null,  
fmselected:false,  
fmicon:null,  
data:null,  
targetData:null,  
permission:null  
}  
`
```

The upload response is an empty string.

Download

The following table represents the request parameters of *download* operations.

Parameter	Type	Default	Explanation
action	String	download	Name of the file operation
path	String	-	Relative path to location where the files to download are present.
names	String[]	-	Name list of the items to be downloaded.
data	FileManagerDirectoryContent	-	Details of the download item.

Example:

```
`typescript  
{  
  action:"download",  
  path:"/",  
  names:["1.png"],  
  data:[  
    {  
      path:null,  
      action:null,  
      newName:null,  
      names:null,  

```

```
name:"1.png",
size:49792,
previousName:null,
dateModified:"2019-07-22T12:15:45.0972405+00:00",
dateCreated:"2019-07-22T12:15:45.0816042+00:00",
hasChild:false,
isFile:true,
type:".png",
id:null,
filterPath:"\\",
targetPath:null,
renameFiles:null,
uploadFiles:null,
caseSensitive:false,
searchString:null,
showHiddenItems:false,
fmiconClass:"e-fe-image",
fmid:null,
fmpld:null,
fmselected:false,
fmicon:null,
data:null,
targetData:null,
permission:null,
fmcreated:"2019-07-22T12:15:45.081Z",
fmmodified:"2019-07-22T12:15:45.097Z",
fmimageUrl:"https://ej2-aspcore-service.azurewebsites.net/api/FileManager/GetImage?path=/1.png",
fmimageAttr:
{
  alt:"1.png"
},
fmhtmlAttr:
{
```

```

class:"e-large-icon",
title:"1.png"
}
}
]
}
,

```

Downloads the requested items from the file server in response.

[GetImage](#)

The following table represents the request parameters of *GetImage* operations.

Parameter	Type	Default	Explanation
path	String	-	Relative path to the image file

Return the image as a file stream in response.

The request from the file manager can be customized using the [beforeSend](#) event. Additional information can be passed to the file manager in file operation response and can be used in customization.

[File request and response contents](#)

The following table represents the contents of *data*, *cwd*, and *files* in the file manager request and response.

Parameter	Type	Default	Explanation
name	String	-	File name
dateCreated	String	-	Date in which file was created (UTC Date string).
dateModified	String	-	Date in which file was last modified (UTC Date string).
filterPath	String	-	Relative path to the file or folder.
hasChild	Boolean	-	Defines this folder has any child folder or not.
isFile	Boolean	-	Say whether the item is file or folder.
size	Number	-	File size
type	String	-	File extension

The following table represents the contents of *error* in the file manager request and response.

Parameter	Type	Default	Explanation
code	String	-	Error code
message	String	-	Error message

|fileExists|String[]|-|List of duplicate file names|

The following table represents the contents of *details* in the file manager request and response.

|Parameter|Type|Default|Explanation|

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

|name|String|-|File name|

|dateCreated|String|-|Date in which file was created (UTC Date string).|

|dateModified|String|-|Date in which file was last modified (UTC Date string).|

|filterPath|String|-|Relative path to the file or folder.|

|hasChild|Boolean|-|Defines this folder has any child folder or not.|

|isFile|Boolean|-|Say whether the item is file or folder.|

|size|Number|-|File size|

|type|String|-|File extension|

|multipleFiles|Boolean|-|Say whether the details are about single file or multiple files.|

Action Buttons

The file manager has several menu buttons to access the file operations. The list of menu buttons available in the file manager is given in the following table.

|Menu Button|Behaviour|

|----|----|

|SortBy| Opens the sub menu to choose the sorting order and sorting parameter. |

|View| Opens the sub menu to choose the View. |

|Open| Navigates to the selected folder. Opens the preview for image files. |

|Refresh| Initiates the read operation for the current directory and displays the updated directory content. |

|NewFolder| Opens the new folder dialog box to receive the name for the new folder. |

|Rename| Opens the rename dialog box to receive the new name for the selected item. |

|Delete| Opens the delete dialog box to confirm the removal of the selected items from the file system. |

|Upload| Opens the upload box to select the items to upload to the file system. |

|Download| Downloads the selected item(s). |

|Details| Get details about the selected items and display them in details dialog box. |

|SelectAll| Selects all the files and folders displayed in the view section. |

The action menu buttons are present in the toolbar and context menu. The toolbar contains the buttons based on the selected items count, while the context menu will appear with a list based on the target.

Toolbar

The toolbar can be divided into two sections as right and left. Whenever the toolbar buttons exceed the size, the buttons present in the left section of the toolbar will be moved to the toolbar popup.

The following table provides the toolbar buttons that appear based on the selection.

<!-- markdownlint-disable MD033 -->

Selected Items Count	Left section	Right section
0 (none of the item)	<i>SortBy Refresh NewFolder Upload</i>	<i>View Details</i>
1 (single item selected)	<i>Delete Download* Rename</i>	<i>Selected items count View* Details</i>
>1 (multiple selection)	<i>Delete Download</i>	<i>Selected items count View* Details</i>

Context menu

The following table provides the default context menu item and the corresponding target areas.

<!-- markdownlint-disable MD033 -->

Menu Name	Menu Items	Target
Layout	<i>SortBy View Refresh NewFolder Upload Details* Select all</i>	<i>Empty space in the view section (details view and large icon view area). Empty folder content.</i>
Folders	<i>Open Delete Rename Downloads* Details</i>	<i>* Folders in treeview, details view, and large icon view.</i>
Files	<i>Open Delete Rename Downloads* Details</i>	<i>* Files in details view and large icon view.</i>

Views in Angular File manager component

View is the section where the files and folders are displayed for the user to browse. The [view](#) API can also be used to change the initial view of the file manager.

The file manager has two types of [views](#) to display the files and folders.

- [Largelcons View](#)
- [Details View](#)

Largelcons View

By Default, File Manager is rendered with largeicons view. The following example demonstrate this.

APP.COMPONENT.TS

```
import { BrowserModule } from '@angular/platform-browser'
import { NgModule } from '@angular/core'
import { FileManagerModule, NavigationPaneService, ToolbarService,
DetailsViewService } from '@syncfusion/ej2-angular-filemanager'
import { Component } from '@angular/core';
@Component({
  imports: [FileManagerModule, ],
```

```

providers:[ NavigationPaneService, ToolbarService, DetailsViewService],
standalone: true,
  selector: 'app-root',
  styleUrls: ['./app.component.css'],
  template: `<ejs-filemanager id='file-manager'
[ajaxSettings]='ajaxSettings'>
  </ejs-filemanager>`
}))
export class AppComponent {
  public ajaxSettings?: object;
  public hostUrl: string = 'https://ej2-aspcore-
service.azurewebsites.net/';
  public ngOnInit(): void {
    this.ajaxSettings = {
      url: this.hostUrl + 'api/FileManager/FileOperations',
      getImageUrl: this.hostUrl + 'api/FileManager/GetImage',
      uploadUrl: this.hostUrl + 'api/FileManager/Upload',
      downloadUrl: this.hostUrl + 'api/FileManager/Download'
    };
  };
};
}

```

MAIN.TS

```

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

```

Details View

Details view is an injectable module in the file manager so, it should be injected before rendering the file manager to avail its functionality. The default appearance of the file manager can be changed from largeicons to details view by using the [view](#) property. The following example demonstrate the file manager with details view.

APP.COMPONENT.TS

```

import { BrowserModule } from '@angular/platform-browser'
import { NgModule } from '@angular/core'
import { FileManagerModule, NavigationPaneService, ToolbarService,
DetailsViewService } from '@syncfusion/ej2-angular-filemanager'
import { Component } from '@angular/core';
@Component({
  imports: [FileManagerModule, ],
  providers:[ NavigationPaneService, ToolbarService, DetailsViewService],
  standalone: true,
  selector: 'app-root',
  styleUrls: ['./app.component.css'],
  template: `<ejs-filemanager id='file-manager'
[ajaxSettings]='ajaxSettings' [view]='view'>
  </ejs-filemanager>`
})
export class AppComponent {
  public ajaxSettings?: object;
  public view?: string;
}

```



```

    public baseUrl: string = 'https://ej2-aspcore-
service.azurewebsites.net/';
    public ngOnInit(): void {
        this.ajaxSettings = {
            url: this.baseUrl + 'api/FileManager/FileOperations',
            getImageUrl: this.baseUrl + 'api/FileManager/GetImage',
            uploadUrl: this.baseUrl + 'api/FileManager/Upload',
            downloadUrl: this.baseUrl + 'api/FileManager/Download'
        };
        // Initial view of File Manager is set to details view
        this.view = "Details";
    }
}

```

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 File manager component

The file manager component allows customizing its functionalities like, context menu, searching, uploading, toolbar using APIs. Given below are some of the functionalities that can be customized in the File Manager,

- [Context menu customization](#)
- [Details view customization](#)
- [Navigation pane customization](#)
- [Show/Hide file extension](#)
- [Show/Hide hidden items](#)
- [Show/Hide thumbnail images in large icons view](#)
- [Toolbar customization](#)
- [Upload customization](#)
- [Tooltip customization](#)

Context menu customization

The context menu settings like, items to be displayed on files, folders and layout click and visibility can be customized using [contextMenuSettings](#) property.

APP.COMPONENT.TS

```

import { BrowserModule } from '@angular/platform-browser'
import { NgModule } from '@angular/core'
import { FileManagerModule, NavigationPaneService, ToolbarService,
DetailsViewService } from '@syncfusion/ej2-angular-filemanager'
import { Component } from '@angular/core';
@Component({
  imports: [FileManagerModule, ],
  providers: [NavigationPaneService, ToolbarService, DetailsViewService],
  standalone: true,
  selector: 'app-root',

```

```

        styleUrls: ['./app.component.css'],
        template: `<ejs-filemanager id='file-manager'
[ajaxSettings]='ajaxSettings' [contextMenuSettings]='contextMenuSettings'>
        </ejs-filemanager>`
    })
    export class AppComponent {
        public ajaxSettings?: object;
        public contextMenuSettings?: object;
        public hostUrl: string = 'https://ej2-aspcore-
service.azurewebsites.net/';
        public ngOnInit(): void {
            this.ajaxSettings = {
                url: this.hostUrl + 'api/FileManager/FileOperations',
                getImageUrl: this.hostUrl + 'api/FileManager/GetImage',
                uploadUrl: this.hostUrl + 'api/FileManager/Upload',
                downloadUrl: this.hostUrl + 'api/FileManager/Download'
            };
            // Context Menu settings customization
            this.contextMenuSettings = { file: ['Open', '|', 'Details'], folder:
['Open', '|', 'Details'], layout: ['SortBy', 'View', 'Refresh', '|',
'Details', '|'], 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));

```

Details view customization

The details view settings like, column width, header text, template for each field can be customized using [detailsViewSettings](#) property.

APP.COMPONENT.TS

```

import { BrowserModule } from '@angular/platform-browser'
import { NgModule } from '@angular/core'
import { FileManagerModule, NavigationPaneService, ToolbarService,
DetailsViewService } from '@syncfusion/ej2-angular-filemanager'
import { Component } from '@angular/core';
@Component({
    imports: [FileManagerModule, ],
    providers:[NavigationPaneService, ToolbarService, DetailsViewService],
    standalone: true,
    selector: 'app-root',
    styleUrls: ['./app.component.css'],
    template: `<ejs-filemanager id='file-manager'
[ajaxSettings]='ajaxSettings' [detailsViewSettings]='detailsViewSettings'
[view]='view'>
        </ejs-filemanager>`
})
export class AppComponent {
    public ajaxSettings?: object;

```

```

public view?: string;
public detailsViewSettings?: object;
public hostUrl: string = 'https://ej2-aspcore-
service.azurewebsites.net/';
public ngOnInit(): void {
    this.ajaxSettings = {
        url: this.hostUrl + 'api/FileManager/FileOperations',
        getImageUrl: this.hostUrl + 'api/FileManager/GetImage',
        uploadUrl: this.hostUrl + 'api/FileManager/Upload',
        downloadUrl: this.hostUrl + 'api/FileManager/Download'
    };
    // Initial view of File Manager is set to details view
    this.view = "Details";
    // Details View settings customization
    this.detailsViewSettings = {
        columns: [
            {field: 'name', headerText: 'File Name', minWidth: 120,
width: 'auto', customAttributes: { class: 'e-fe-grid-name' },template:
'${name}'},
            {field: 'size', headerText: 'File Size',minWidth: 50, width:
'110', template: '${size}'},
            { field: '_fm_modified', headerText: 'Date
Modified',minWidth: 50, width: '190'}
        ]
    };
};
}

```

MAIN.TS

```

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

```

Navigation pane customization

The navigation pane settings like, minimum and maximum width and visibility can be customized using [navigationPaneSettings](#) property.

APP.COMPONENT.TS

```

import { BrowserModule } from '@angular/platform-browser'
import { NgModule } from '@angular/core'
import { FileManagerModule, NavigationPaneService, ToolbarService,
DetailsViewService } from '@syncfusion/ej2-angular-filemanager'
import { Component } from '@angular/core';
@Component({
  imports: [FileManagerModule, ],
  providers:[NavigationPaneService, ToolbarService, DetailsViewService],
  standalone: true,
  selector: 'app-root',
  styleUrls: ['./app.component.css'],
  template: `<ejs-filemanager id='file-manager'
[ajaxSettings]='ajaxSettings'
[navigationPaneSettings]='navigationPaneSettings'>

```

```

    </ejs-filemanager>`
  })
  export class AppComponent {
    public ajaxSettings?: object;
    public navigationPaneSettings?: object;
    public hostUrl: string = 'https://ej2-aspcore-
service.azurewebsites.net/';
    public ngOnInit(): void {
      this.ajaxSettings = {
        url: this.hostUrl + 'api/FileManager/FileOperations',
        getImageUrl: this.hostUrl + 'api/FileManager/GetImage',
        uploadUrl: this.hostUrl + 'api/FileManager/Upload',
        downloadUrl: this.hostUrl + 'api/FileManager/Download'
      };
      // Navigation Pane settings customization
      this.navigationPaneSettings = { maxWidth: '850px', minWidth: '140px',
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));

```

Show/Hide file extension

The file extensions are displayed in the File Manager by default. This can be hidden by disabling the [showFileExtension](#) property.

In File Manager [fileLoad](#) and [fileOpen](#) events are triggered before the file/folder is rendered and before the file/folder is opened respectively. These events can be utilized to perform operations before a file/folder is rendered or opened.

APP.COMPONENT.TS

```

import { BrowserModule } from '@angular/platform-browser'
import { NgModule } from '@angular/core'
import { FileManagerModule ,NavigationPaneService, ToolbarService,
DetailsViewService } from '@syncfusion/ej2-angular-filemanager'
import { Component } from '@angular/core';
@Component({
  imports: [FileManagerModule, ],
  providers:[NavigationPaneService, ToolbarService, DetailsViewService],
  standalone: true,
  selector: 'app-root',
  styleUrls: ['./app.component.css'],
  template: `<ejs-filemanager id='file-manager'
[ajaxSettings]='ajaxSettings' [showFileExtension]='showFileExtension'
(fileLoad)='onBeforeFileLoad($event)' (fileOpen)='onBeforeFileOpen($event)'>
</ejs-filemanager>`
})
export class AppComponent {
  public ajaxSettings?: object;

```

```

    public showFileExtension?: boolean;
    public baseUrl: string = 'https://ej2-aspcore-
service.azurewebsites.net/';
    public ngOnInit(): void {
        this.ajaxSettings = {
            url: this.baseUrl + 'api/FileManager/FileOperations',
            getImageUrl: this.baseUrl + 'api/FileManager/GetImage',
            uploadUrl: this.baseUrl + 'api/FileManager/Upload',
            downloadUrl: this.baseUrl + 'api/FileManager/Download'
        };
        // Hides the file extension in File Manager
        this.showFileExtension = false;
    };
    // File Manager's file beforeFileLoad function
    onBeforeFileLoad(args: any) {
        console.log(args.fileDetails.name + " is loading");
    }
    // File Manager's file beforeFileOpen function
    onBeforeFileOpen(args: any) {
        console.log(args.fileDetails.name + " is opened");
    }
}

```

MAIN.TS

```

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

```

Show/Hide hidden items

The File Manager provides support to show/hide the hidden items by enabling/disabling the [showHiddenItems](#) property.

APP.COMPONENT.TS

```

import { BrowserModule } from '@angular/platform-browser'
import { NgModule } from '@angular/core'
import { FileManagerAllModule, NavigationPaneService, ToolbarService,
DetailsViewService } from '@syncfusion/ej2-angular-filemanager'
import { Component } from '@angular/core';
@Component({
  imports: [FileManagerAllModule, ],
  providers:[NavigationPaneService, ToolbarService, DetailsViewService],
  standalone: true,
  selector: 'app-root',
  styleUrls: ['./app.component.css'],
  template: `<ejs-filemanager id='file-manager'
[ajaxSettings]='ajaxSettings' [showHiddenItems]='showHiddenItems'>
</ejs-filemanager>`
})
export class AppComponent {
  public ajaxSettings?: object;
  public showHiddenItems?: boolean;
}

```

```

    public baseUrl: string = 'https://ej2-aspcore-
service.azurewebsites.net/';
    public ngOnInit(): void {
        this.ajaxSettings = {
            url: this.baseUrl + 'api/FileManager/FileOperations',
            getImageUrl: this.baseUrl + 'api/FileManager/GetImage',
            uploadUrl: this.baseUrl + 'api/FileManager/Upload',
            downloadUrl: this.baseUrl + 'api/FileManager/Download'
        };
        // The default value set for showHiddenItems is false
        this.showHiddenItems = true;
    }
}

```

MAIN.TS

```

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

```

Show/Hide thumbnail images in large icons view

The thumbnail images are displayed in the File Manager's large icons view by default. This can be hidden by disabling the [showThumbnail](#) property.

APP.COMPONENT.TS

```

import { BrowserModule } from '@angular/platform-browser'
import { NgModule } from '@angular/core'
import { FileManagerModule, NavigationPaneService, ToolbarService,
DetailsViewService } from '@syncfusion/ej2-angular-filemanager'
import { Component } from '@angular/core';
@Component({
  imports: [FileManagerModule, ],
  providers:[NavigationPaneService, ToolbarService, DetailsViewService],
  standalone: true,
  selector: 'app-root',
  styleUrls: ['./app.component.css'],
  template: `<ejs-filemanager id='file-manager'
[ajaxSettings]='ajaxSettings' [showThumbnail]='showThumbnail'>
</ejs-filemanager>`
})
export class AppComponent {
  public ajaxSettings?: object;
  public showThumbnail?: boolean;
  public baseUrl: string = 'https://ej2-aspcore-
service.azurewebsites.net/';
  public ngOnInit(): void {
    this.ajaxSettings = {
      url: this.baseUrl + 'api/FileManager/FileOperations',
      getImageUrl: this.baseUrl + 'api/FileManager/GetImage',
      uploadUrl: this.baseUrl + 'api/FileManager/Upload',
      downloadUrl: this.baseUrl + 'api/FileManager/Download'
    };
    // Hides the thumbnail images in File Manager's large icons view
  }
}

```

```

        this.showThumbnail = false;
    };
}

```

MAIN.TS

```

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

```

Toolbar customization

The toolbar settings like, items to be displayed in toolbar and visibility can be customized using [toolbarSettings](#) property.

APP.COMPONENT.TS

```

import { BrowserModule } from '@angular/platform-browser'
import { NgModule } from '@angular/core'
import { FileManagerModule, NavigationPaneService, ToolbarService,
DetailsViewService } from '@syncfusion/ej2-angular-filemanager'
import { Component } from '@angular/core';
@Component({
  imports: [FileManagerModule, ],
  providers:[NavigationPaneService, ToolbarService, DetailsViewService],
  standalone: true,
  selector: 'app-root',
  styleUrls: ['./app.component.css'],
  template: `<ejs-filemanager id='file-manager'
[ajaxSettings]='ajaxSettings' [toolbarSettings]='toolbarSettings'>
</ejs-filemanager>`
})
export class AppComponent {
  public ajaxSettings?: object;
  public toolbarSettings?: object;
  public hostUrl: string = 'https://ej2-aspcore-
service.azurewebsites.net/';
  public ngOnInit(): void {
    this.ajaxSettings = {
      url: this.hostUrl + 'api/FileManager/FileOperations',
      getImageUrl: this.hostUrl + 'api/FileManager/GetImage',
      uploadUrl: this.hostUrl + 'api/FileManager/Upload',
      downloadUrl: this.hostUrl + 'api/FileManager/Download'
    };
    // Toolbar settings customization
    this.toolbarSettings = { items: ['NewFolder', 'Refresh', 'View',
'Details'], visible: true};
  };
}

```

MAIN.TS

```

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

```

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

See Also

- [How to add new items or customize default items](#)

Upload customization

The upload settings like, minimum and maximum file size and enabling auto upload can be customized using [uploadSettings](#) property.

APP.COMPONENT.TS

```
import { BrowserModule } from '@angular/platform-browser'
import { NgModule } from '@angular/core'
import { FileManagerModule, NavigationPaneService, ToolbarService,
DetailsViewService } from '@syncfusion/ej2-angular-filemanager'
import { Component } from '@angular/core';
@Component({
  imports: [FileManagerModule, ],
  providers: [NavigationPaneService, ToolbarService, DetailsViewService],
  standalone: true,
  selector: 'app-root',
  styleUrls: ['./app.component.css'],
  template: `<ejs-filemanager id='file-manager'
[ajaxSettings]='ajaxSettings' [uploadSettings]='uploadSettings'>
</ejs-filemanager>`
})
export class AppComponent {
  public ajaxSettings?: object;
  public uploadSettings?: object;
  public hostUrl: string = 'https://ej2-aspcore-
service.azurewebsites.net/';
  public ngOnInit(): void {
    this.ajaxSettings = {
      url: this.hostUrl + 'api/FileManager/FileOperations',
      getImageUrl: this.hostUrl + 'api/FileManager/GetImage',
      uploadUrl: this.hostUrl + 'api/FileManager/Upload',
      downloadUrl: this.hostUrl + 'api/FileManager/Download'
    };
    // Upload settings customization
    this.uploadSettings = { maxFileSize: 233332, minFileSize: 120,
autoUpload: true};
  };
}
```

MAIN.TS

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


Tooltip customization

The tooltip value can be customized by adding extra content to the title of the toolbar, navigation pane, details view and large icons of the file manager element.

APP.COMPONENT.TS

```
import { BrowserModule } from '@angular/platform-browser'
import { NgModule } from '@angular/core'
import { TooltipAllModule } from '@syncfusion/ej2-angular-popups'
import { FileManagerModule, NavigationPaneService, ToolbarService,
DetailsViewService } from '@syncfusion/ej2-angular-filemanager'
import { Component, ViewChild } from '@angular/core';
import { FileManagerComponent, FileLoadEventArgs } from '@syncfusion/ej2-
angular-filemanager';
import { TooltipComponent, TooltipEventArgs } from '@syncfusion/ej2-angular-
popups';
import { getValue, select } from '@syncfusion/ej2-base';
@Component({
  imports: [FileManagerModule, TooltipAllModule],
  providers:[NavigationPaneService, ToolbarService, DetailsViewService],
  standalone: true,
  selector: 'app-root',
  styleUrls: ['./app.component.css'],
  template: ` <div class="filemanagerContainer">
    <ejs-filemanager id='file-manager' #fileObj
[ajaxSettings]='ajaxSettings' (fileLoad)="fileLoad($event)">
    </ejs-filemanager>
  </div>`
})
export class AppComponent {
  @ViewChild('fileObj',{ static: true })
  public fileObj?: FileManagerComponent;
  public ajaxSettings?: object;
  public hostUrl: string = 'https://ej2-aspcore-
service.azurewebsites.net/';
  public ngOnInit(): void {
    this.ajaxSettings = {
      url: this.hostUrl + 'api/FileManager/FileOperations',
      getImageUrl: this.hostUrl + 'api/FileManager/GetImage',
      uploadUrl: this.hostUrl + 'api/FileManager/Upload',
      downloadUrl: this.hostUrl + 'api/FileManager/Download'
    };
  };
  fileLoad (args: FileLoadEventArgs) {
    //Native tooltip customization to display additional information in
new line
    let target: Element = args.element as Element;
    if (args.module==='DetailsView') {
      let element: Element = select('[title]', args.element);
      let title: string = getValue('name', args.fileDetails) +
        '\n' + getValue('dateModified', args.fileDetails);
      element.setAttribute('title', title);
    } else if (args.module==='LargeIconsView') {
      let title: string = getValue('name', args.fileDetails) +
        '\n' + getValue('dateModified', args.fileDetails);
      target.setAttribute('title', title);
    }
  }
}
```

```
}
}
```

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 in Angular File manager component

The file manager allows you to select multiple files by enabling the [allowMultiSelection](#) property (enabled by default). The multiple selection can be done by pressing the **Ctrl** key or **Shift** key and selecting the files. The check box can also be used to do multiple selection. **Ctrl + A** can be used to select all files in the current directory. The [fileSelect](#) event will be triggered when the items of file manager control is selected or unselected.

APP.COMPONENT.TS

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { FileManagerModule, NavigationPaneService, ToolbarService,
DetailsViewService } from '@syncfusion/ej2-angular-filemanager';
import { Component } from '@angular/core';
import { FileSelectEventArgs } from '@syncfusion/ej2-filemanager';
@Component({
  imports: [FileManagerModule, ],
  providers: [ NavigationPaneService, ToolbarService, DetailsViewService],
  standalone: true,
  selector: 'app-root',
  styleUrls: ['./app.component.css'],
  template: `<ejs-filemanager id='multi' [ajaxSettings]='ajaxSettings'
[allowMultiSelection]='allowMultiSelection'
(fileSelect)='onFileSelect($event)'>
    </ejs-filemanager>`
})
export class AppComponent {
  public ajaxSettings?: object;
  public allowMultiSelection?: boolean;
  public hostUrl: string = 'https://ej2-aspcore-
service.azurewebsites.net/';
  public ngOnInit(): void {
    this.ajaxSettings = {
      url: this.hostUrl + 'api/FileManager/FileOperations',
      getImageUrl: this.hostUrl + 'api/FileManager/GetImage',
      uploadUrl: this.hostUrl + 'api/FileManager/Upload',
      downloadUrl: this.hostUrl + 'api/FileManager/Download'
    };
    this.allowMultiSelection = true;
  }
  // File Manager's file select event function
  onFileSelect(args: FileSelectEventArgs | any) {
    console.log(args.fileDetails.name + " has been " + args.action +
"ed");
  }
}
```

```
}

```

MAIN.TS

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

Note: The File Manager has support to select files and folders initially or dynamically by specifying their names in [selectedItems](#) property.

Drag and drop in Angular File manager component

The file manager allows files or folders to be moved from one folder to another by using the [allowDragAndDrop](#) property. It also supports uploading a file by dragging it from Windows Explorer to FileManager control. You can enable or disable this support by using the [allowDragAndDrop](#) property of file manager.

The event triggered in drag and drop support are

- [fileDragStart](#) - Triggers when the file/folder dragging is started.
- [fileDragging](#) - Triggers while dragging the file/folder.
- [fileDragStop](#) - Triggers when the file/folder is about to be dropped at the target.
- [fileDropped](#) - Triggers when the file/folder is dropped.

APP.COMPONENT.TS

```
import { BrowserModule } from '@angular/platform-browser'
import { NgModule } from '@angular/core'
import { FileManagerModule, NavigationPaneService, ToolbarService,
DetailsViewService } from '@syncfusion/ej2-angular-filemanager'
import { Component } from '@angular/core';
@Component({
  imports: [FileManagerModule, ],
  providers:[NavigationPaneService, ToolbarService, DetailsViewService],
  standalone: true,
  selector: 'app-root',
  styleUrls: ['./app.component.css'],
  template: `<ejs-filemanager id='file-manager'
[ajaxSettings]='ajaxSettings' [allowDragAndDrop]='allowDragAndDrop'
(fileDragStart)='onFileDragStart($event)'
(fileDragStop)='onFileDragStop($event)'
(fileDragging)='onFileDragging($event)' (fileDropped)='onFileDropped($event)'
>
    </ejs-filemanager>`
})
export class AppComponent {
  public ajaxSettings?: object;
  public allowDragAndDrop?: boolean;
  public hostUrl: string = 'https://ej2-aspcore-
service.azurewebsites.net/';
  public ngOnInit(): void {
    this.ajaxSettings = {
      url: this.hostUrl + 'api/FileManager/FileOperations',
```

```
        getImageUrl: this.hostUrl + 'api/FileManager/GetImage',
        uploadUrl: this.hostUrl + 'api/FileManager/Upload',
        downloadUrl: this.hostUrl + 'api/FileManager/Download'
    };
    this.allowDragAndDrop = true;
};
// File Manager's file drag start event function
onFileDragStart(args: any) {
    console.log("File drag start");
}
// File Manager's file drag stop event function
onFileDragStop(args: any) {
    console.log("File drag stop");
}
// File Manager's file dragging event function
onFileDragging(args: any) {
    console.log("File dragging");
}
// File Manager's file dropped event function
onFileDropped(args: any) {
    console.log("File dropped");
}
}
```

MAIN.TS

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

File system provider in Angular File manager component

The file system provider allows the File Manager component to manage the files and folders in a physical or cloud-based file system. It provides the methods for performing various file actions like creating a new folder, copying and moving of files or folders, deleting, uploading, and downloading the files or folders in the file system.

The following file providers are added in Syncfusion EJ2 File Manager component.

- [ASP.NET Core file system provider](#)
- [ASP.NET MVC 5 file system provider](#)
- [ASP.NET Core Azure cloud file system Provider](#)
- [ASP.NET MVC 5 Azure cloud file system Provider](#)
- [ASP.NET Core Amazon S3 cloud file provider](#)
- [ASP.NET MVC Amazon S3 cloud file provider](#)
- [File Transfer Protocol file system provider](#)
- [SQL database file system provider](#)
- [NodeJS file system provider](#)
- [Google Drive file system provider](#)
- [Firebase Realtime Database file system provider](#)

ASP.NET Core file system provider

The ASP.NET Core file system provider allows the users to access and manage the physical file system. To get started, clone the [ej2-aspcore-file-provider](https://github.com/SyncfusionExamples/ej2-aspcore-file-provider) using the following command.

```
`typescript
git clone https://github.com/SyncfusionExamples/ej2-aspcore-file-provider ej2-aspcore-file-provider
cd ej2-aspcore-file-provider
`
```

After cloning, just open the project in Visual Studio and restore the NuGet packages. Now, set the root directory of the physical file system in the FileManager controller.

After setting the root directory of the file system, just build and run the project. Now, the project will be hosted in `http://localhost:{port}` and just mapping the `ajaxSettings` property of the FileManager component to the appropriate controller methods allows to manage the files in the physical file system.

```
`typescript
import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
  styleUrls: ['app/app.component.css'],
  templateUrl: 'app/app.component.html'
})
export class AppComponent {
  public ajaxSettings: object;
  public hostUrl: string = 'http://localhost:{port}/';
  public ngOnInit(): void {
    // Initializing File Manager with ASP.NET Core service.
    this.ajaxSettings = {
      // Replace the hosted port number in the place of "{port}"
      url: this.hostUrl + "api/FileManager/FileOperations",
      downloadUrl: this.hostUrl + "api/FileManager/Download",
      uploadUrl: this.hostUrl + "api/FileManager/Upload",
      getImageUrl: this.hostUrl + "api/FileManager/GetImage"
    };
  }
}
```

Note: To learn more about the file actions that can be performed with ASP.NET Core file system provider, refer to this [link](#)

ASP.NET MVC 5 file system provider

The ASP.NET MVC5 file system provider allows the users to access and manage the physical file system. To get started, clone the [ej2-aspmvc-file-provider](#) using the following command.

```
`typescript
git clone https://github.com/SyncfusionExamples/ej2-aspmvc-file-provider ej2-aspmvc-file-provider
cd ej2-aspmvc-file-provider
`
```

After cloning, just open the project in Visual Studio and restore the NuGet packages. Now, set the root directory of the physical file system in the FileManager controller using the Root Folder method.

After setting the root directory of the file system, just build and run the project. Now, the project will be hosted in `http://localhost:{port}` and just mapping the ajaxSettings property of the FileManager component to the appropriate controller methods allows to manage the files in the physical file system.

```
`typescript
import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
  styleUrls: ['app/app.component.css'],
  templateUrl: 'app/app.component.html'
})
export class AppComponent {
  public ajaxSettings: object;
  public hostUrl: string = 'http://localhost:{port}/';
  public ngOnInit(): void {
    // Initializing File Manager with ASP.NET MVC service.
    this.ajaxSettings = {
      // Replace the hosted port number in the place of "{port}"
      url = this.hostUrl + "FileManager/FileOperations",
      downloadUrl = this.hostUrl + "FileManager/Download",
      uploadUrl = this.hostUrl + "FileManager/Upload",
      getImageUrl = this.hostUrl + "FileManager/GetImage"
    };
  }
}
```

Note: To learn more about the file actions that can be performed with ASP.NET MVC 5 file system provider, refer to this [link](#)

ASP.NET Core Azure cloud file system Provider

In ASP.NET Core, Azure file system provider allows the users to access and manage the blobs in the Azure blob storage. To get started, clone the [azure-aspcore-file-provider](#) using the following command.

```
`typescript
git clone https://github.com/SyncfusionExamples/azure-aspcore-file-provider azure-aspcore-file-provider
```

After cloning, just open the project in Visual Studio and restore the NuGet packages. Now, register the Azure storage by passing details like name, password, and blob name to the Register Azure method in the FileManager controller.

```
`typescript
void RegisterAzure(string accountName, string accountKey, string blobName)
```

Then, set the blob container and the root blob directory by passing the corresponding URLs as parameters in the setBlobContainer method as follows.

```
`typescript
void setBlobContainer(string blobPath, string filePath)
```

Note: Also, assign the same *blobPath* URL and *filePath* URL in [AzureFileOperations](#) and [AzureUpload](#) methods in the FileManager controller to determine the original path of the Azure blob.

After setting the blob container references, just build and run the project. Now, the project will be hosted in `http://localhost:{port}` and just mapping the ajaxSettings property of the FileManager component to the appropriate controller methods allows to manage the Azure blob storage.

```
`typescript
import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
  styleUrls: ['app/app.component.css'],
  templateUrl: 'app/app.component.html'
})
export class AppComponent {
  public ajaxSettings: object;
  public hostUrl: string = 'http://localhost:{port}/';
```

```
public ngOnInit(): void {  
  // File Manager sample with azure service.  
  this.ajaxSettings = {  
    // Replace the hosted port number in the place of "{port}"  
    url = this.hostUrl + "api/AzureProvider/AzureFileOperations",  
    downloadUrl = this.hostUrl + "api/AzureProvider/AzureDownload",  
    uploadUrl = this.hostUrl + "api/AzureProvider/AzureUpload",  
    getImageUrl = this.hostUrl + "api/AzureProvider/AzureGetImage"  
  };  
}  
}
```

NuGet: Additionally, we have created a [NuGet](#) package of **ASP.NET Core Azure file system provider**.

Please, use the following command to install the NuGet package in an application.

```
`typescript  
dotnet add package Syncfusion.EJ2.FileManager.AzureFileProvider.AspNet.Core  
`
```

Note: To learn more about the file actions that can be performed with ASP.NET Core Azure Cloud File System Provider, refer to this [link](#)

ASP.NET MVC 5 Azure cloud file system Provider

In ASP.NET MVC, Azure file system provider allows the users to access and manage the blobs in the Azure blob storage. To get started, clone the [ej2-azure-aspmvc-file-provider](#) using the following command.

```
`typescript  
git clone https://github.com/SyncfusionExamples/ej2-azure-aspmvc-file-provider ej2-azure-aspmvc-file-provider  
`
```

After cloning, just open the project in Visual Studio and restore the NuGet packages. Now, register the Azure storage by passing details like name, password, and blob name to the Register Azure method in the FileManager controller.

```
`typescript  
void RegisterAzure(string accountName, string accountKey, string blobName)  
`
```

Then, set the blob container and the root blob directory by passing the corresponding URLs as parameters in the setBlobContainer method as follows.


```
`typescript
void setBlobContainer(string blobPath, string filePath)
`
```

Note: Also, assign the same *blobPath* URL and *filePath* URL in [AzureFileOperations](#) and [AzureUpload](#) methods in the FileManager controller to determine the original path of the Azure blob.

After setting the blob container references, just build and run the project. Now, the project will be hosted in `http://localhost:{port}` and just mapping the `ajaxSettings` property of the FileManager component to the appropriate controller methods allows to manage the Azure blob storage.

```
`typescript
import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
  styleUrls: ['app/app.component.css'],
  templateUrl: 'app/app.component.html'
})
export class AppComponent {
  public ajaxSettings: object;
  public hostUrl: string = 'http://localhost:{port}/';
  public ngOnInit(): void {
    // File Manager sample with azure service.
    this.ajaxSettings = {
      // Replace the hosted port number in the place of "{port}"
      url = this.hostUrl + "AzureProvider/AzureFileOperations",
      downloadUrl = this.hostUrl + "AzureProvider/AzureDownload",
      uploadUrl = this.hostUrl + "AzureProvider/AzureUpload",
      getImageUrl = this.hostUrl + "AzureProvider/AzureGetImage"
    };
  }
}
```

Note: To learn more about the file actions that can be performed with ASP.NET MVC 5 Azure Cloud File System Provider, refer to this [link](#)

ASP.NET Core Amazon S3 cloud file provider

In ASP.NET Core, Amazon **S3** (*Simple Storage Service*) cloud file provider allows the users to access and manage a server hosted file system as collection of objects stored in the Amazon S3 Bucket. To get started, clone the [amazon-s3-aspcore-file-provider](https://github.com/SyncfusionExamples/amazon-s3-aspcore-file-provider) using the following command

```
`typescript
git clone https://github.com/SyncfusionExamples/amazon-s3-aspcore-file-provider.git amazon-s3-
aspcore-file-provider.git
`
```

Note: To learn more about creating and configuring an Amazon S3 bucket, refer to this [link](#).

After cloning, open the project in Visual Studio and restore the NuGet packages. Now, register Amazon S3 client account details like *awsAccessKeyId*, *awsSecretKeyId* and *awsRegion* details in **RegisterAmazonS3** method in the FileManager controller to perform the file operations.

```
`typescript
void RegisterAmazonS3(string bucketName, string awsAccessKeyId, string awsSecretAccessKey, string
bucketRegion)
`
```

After registering the Amazon client account details, just build and run the project. Now, the project will be hosted in `http://localhost:{port}` and just mapping the **ajaxSettings** property of the FileManager component to the appropriate controller methods allows to manage the Amazon **S3** (*Simple Storage Service*) bucket's objects storage.

```
`typescript
import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
  styleUrls: ['app/app.component.css'],
  templateUrl: 'app/app.component.html'
})
export class AppComponent {
  public ajaxSettings: object;
  public hostUrl: string = 'http://localhost:{port}/';
  public ngOnInit(): void {
    // File Manager sample with amazon service.
    this.ajaxSettings = {
      // Replace the hosted port number in the place of "{port}"
      url = this.hostUrl + "api/AmazonS3Provider/AmazonS3FileOperations",
      downloadUrl = this.hostUrl + "api/AmazonS3Provider/AmazonS3Download",
    }
  }
}
```

```

uploadUrl = this.hostUrl + "api/AmazonS3Provider/AmazonS3Upload",
getImageUrl = this.hostUrl + "api/AmazonS3Provider/AmazonS3GetImage"
};
}
}
`

```

Note: To learn more about the file actions that can be performed with Amazon S3 Cloud File provider, refer to this [link](#)

ASP.NET MVC Amazon S3 cloud file provider

In ASP.NET MVC, Amazon **S3** (*Simple Storage Service*) cloud file provider allows the users to access and manage a server hosted files as collection of objects stored in the Amazon S3 Bucket. To get started, clone the [ej2-amazon-s3-aspmvc-file-provider](#) using the following command

```

`typescript

git clone https://github.com/SyncfusionExamples/ej2-amazon-s3-aspmvc-file-provider.git ej2-amazon-
s3-aspmvc-file-provider.git
`

```

Note: To learn more about creating and configuring an Amazon S3 bucket, refer to this [link](#).

After cloning, open the project in Visual Studio and restore the NuGet packages. Now, register Amazon S3 client account details like *awsAccessKeyId*, *awsSecretKeyId* and *awsRegion* details in **RegisterAmazonS3** method in the FileManager controller to perform the file operations.

```

`typescript

void RegisterAmazonS3(string bucketName, string awsAccessKeyId, string awsSecretAccessKey, string
bucketRegion)
`

```

After registering the Amazon client account details, just build and run the project. Now, the project will be hosted in `http://localhost:{port}` and just mapping the **ajaxSettings** property of the FileManager component to the appropriate controller methods allows to manage the Amazon **S3** (*Simple Storage Service*) bucket's objects storage.

```

`typescript

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

@Component({
  selector: 'app-root',
  styleUrls: ['app/app.component.css'],
  templateUrl: 'app/app.component.html'
})

export class AppComponent {

```

```
public ajaxSettings: object;
public hostUrl: string = 'http://localhost:{port}/';
public ngOnInit(): void {
// File Manager sample with amazon service.
this.ajaxSettings = {
// Replace the hosted port number in the place of "{port}"
url: hostUrl + "FileManager/FileOperations",
downloadUrl: hostUrl + "FileManager/Download",
uploadUrl: hostUrl + "FileManager/Upload",
getImageUrl: hostUrl + "FileManager/GetImage"
};
}
}
```

Note: To learn more about the file actions that can be performed with ASP.NET MVC Amazon S3 Cloud File Provider, refer to this [link](#)

File Transfer Protocol file system provider

In ASP.NET Core, File Transfer Protocol file system provider allows the users to access to the hosted file system as collection of objects stored in the file storage using File Transfer Protocol. To get started, clone the [ftp-aspcore-file-provider](#) using the following command

```
`typescript
git clone https://github.com/SyncfusionExamples/ftp-aspcore-file-provider.git ftp-aspcore-file-
provider.git
`
```

After cloning, open the project in Visual Studio and restore the NuGet packages. Now, register File Transfer Protocol details like *hostName*, *userName* and *password* in **SetFTPConnection** method in the FileManager controller to perform the file operations.

```
`typescript
void SetFTPConnection(string hostName, string userName, string password)
`
```

After registering the File Transfer Protocol details, just build and run the project. Now, the project will be hosted in `http://localhost:{port}` and just mapping the **ajaxSettings** property of the FileManager component to the appropriate controller methods allows you to manage the FTP's objects storage.

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

```
selector: 'app-root',
styleUrls: ['app/app.component.css'],
templateUrl: 'app/app.component.html'
})
export class AppComponent {
public ajaxSettings: object;
public hostUrl: string = 'http://localhost:{port}/';
public ngOnInit(): void {
// File Manager sample with file transfer protocol service.
this.ajaxSettings = {
// Replace the hosted port number in the place of "{port}"
url = this.hostUrl + "api/FTPProvider/FTPFileOperations",
downloadUrl = this.hostUrl + "api/FTPProvider/FTPDownload",
uploadUrl = this.hostUrl + "api/FTPProvider/FTPUpload",
getImageUrl = this.hostUrl + "api/FTPProvider/FTPGetImage"
};
}
}
```

Note: To learn more about the file actions that can be performed with File Transfer Protocol file system provider, refer to this [link](#)

SQL database file system provider

In ASP.NET Core, SQL database file system provider allows the users to manage the file system being maintained in a SQL database table. Unlike the other file system providers, the SQL database file system provider works on ID basis. Here, each file and folder have a unique ID based on which all the file operations will be performed. To get started, clone the [sql-server-database-aspcore-file-provider](#) using the following command.

```
`typescript
<add name="FileExplorerConnection" connectionString="Data
Source=(LocalDB)\v11.0;AttachDbFilename=|DataDirectory|\FileManager.mdf;Integrated
Security=True;Trusted_Connection=true" />
```

After cloning, just open the project in Visual Studio and restore the NuGet packages. To establish the SQL server connection with the database file (for eg: FileManager.mdf), specify the connection string in the web config file as follows.

```
`typescript
```

```
<add name="FileExplorerConnection" connectionString="Data
Source=(LocalDB)\v11.0;AttachDbFilename=|DataDirectory|\FileManager.mdf;Integrated
Security=True;Trusted_Connection=true" />
```

Then, make an entry for the connection string in `appsettings.json` file as follows.

```
`typescript
"ConnectionStrings": {
  "FileManagerConnection": "Data
Source=(LocalDB)\\MSSQLLocalDB;AttachDbFilename=|DataDirectory|\\App_Data\\FileManager.mdf;In
tegrated Security=True;Connect Timeout=30"
}
```

Now, to configure the database connection, set the connection name, table name and root folder ID value by passing these values to the `SetSqlConnection` method.

```
`typescript
void SetSqlConnection(string name, string tableName, string tableID)
```

Refer to this [FileManager.mdf](#), to learn about the pre-defined file system SQL database for the EJ2 File Manager.

After configuring the connection, just build and run the project. Now, the project will be hosted in `http://localhost:{port}` and just mapping the `ajaxSettings` property of the `FileManager` component to the appropriate controller methods allows to manage the files in the SQL database table.

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

@Component({
  selector: 'app-root',
  styleUrls: ['app/app.component.css'],
  templateUrl: 'app/app.component.html'
})

export class AppComponent {
  public ajaxSettings: object;
  public hostUrl: string = 'http://localhost:{port}/';
  public ngOnInit(): void {
    // Initializing the File Manager with SQL database service.
    this.ajaxSettings = {
```

```
// Replace the hosted port number in the place of "{port}"
url = this.hostUrl + "api/SQLProvider/SQLFileOperations",
downloadUrl = this.hostUrl + "api/SQLProvider/SQLDownload",
uploadUrl = this.hostUrl + "api/SQLProvider/SQLUpload",
getImageUrl = this.hostUrl + "api/SQLProvider/SQLGetImage"
};
}
}
```

Note: To learn more about the file actions that can be performed with SQL database file system provider, refer to this [link](#)

NodeJS file system provider

The NodeJS file system provider allows the users to manage the files and folders in a physical file system. It provides methods for performing all basic file operations like creating a folder, copy, move, delete, and download files and folders in the file system. We can use of the NodeJS file system provider either by installing the [ej2-filemanager-node-filesystem](#) package or by cloning the [file system provider](#) from the GitHub.

Using ej2-filemanager-node-filesystem package

- Install the ej2-filemanager-node-filesystem package by running the below command.

```
`typescript
npm install @syncfusion/ej2-filemanager-node-filesystem
`
```

- After installing the package, navigate to the ej2-filemanager-node-filesystem package folder within the node-modules.
- Run the command **npm install** command.

Cloning the ej2-filemanager-node-filesystem from GitHub

- Clone the ej2-filemanager-node-filesystem using the following command.

```
`typescript
git clone https://github.com/SyncfusionExamples/ej2-filemanager-node-filesystem.git node-filesystem-provider
`
```

- After cloning, open the root folder and run the command **npm install** command.

After installing the packages, set the root folder directory of the physical file system in the package JSON under scripts sections as follows.

```
`typescript
"start": "node filesystem-server.js -d D:/Projects"
`
```

Note: By default, the root directory will be configured to set `C:/Users` as the root directory.

To set the port in which the project to be hosted and the root directory of the file system. Run the following command.

```
`typescript
set PORT=3000 && node filesystem-server.js -d D:/Projects
`
```

Note: By default, the service will run `8090` port.

Now, just mapping the **ajaxSettings** property of the FileManager component to the appropriate file operation methods in the filesystem-server.js file will allow to manage the physical file system with NodeJS file system provider.

```
`typescript
import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
  styleUrls: ['app/app.component.css'],
  templateUrl: 'app/app.component.html'
})
export class AppComponent {
  public ajaxSettings: object;
  public hostUrl: string = 'http://localhost:8090/';
  public ngOnInit(): void {
    // Initializing the File Manager with NodeJS service.
    this.ajaxSettings = {
      // Replace the hosted port number in the place of "{port}"
      url: this.hostUrl,
      downloadUrl: this.hostUrl+ "Download",
      uploadUrl: this.hostUrl+ "Upload",
      getImageUrl: this.hostUrl+ "GetImage"
    };
  }
}
```



```
}  
}  
,
```

Note: To learn more about the file actions that can be performed with NodeJS file system provider, refer to this [link](#)

Google Drive file system provider

In ASP.NET Core, Google Drive file system provider allows the users to manage the files and folders in a Google Drive account. The Google Drive file system provider works on ID basis where each file and folder have a unique ID. To get started, clone the [google-drive-aspcore-file-provider](#) using the following command.

```
`typescript  
git clone https://github.com/SyncfusionExamples/google-drive-aspcore-file-provider google-drive-  
aspcore-file-provider  
cd google-drive-aspcore-file-provider  
,
```

Google Drive file system provider use the [Google Drive APIs](#) to read the file in the file system and uses the [OAuth 2.0](#) protocol for authentication and authorization. To authenticate from the client end, obtain the OAuth 2.0 client credentials from the [Google API Console](#). To learn more about generating the client credentials from the from Google API Console, refer to this [link](#).

After generating the client secret data, copy the JSON data to the following specified JSON files in the cloned location.

- EJ2GoogleDriveFileProvider > credentials > client_secret.json
- GoogleOAuth2.0Base > credentials > client_secret.json

After updating the credentials, just build and run the project. Now, the project will be hosted in [http://localhost:{port}](#), and it will ask to log on to the Gmail account created the client secret credentials. Then, provide permission to access the Google Drive files by clicking the allow access button in the page. Now, just mapping the ajaxSettings property of the FileManager component to the appropriate controller methods will allows to manage the files from the Google Drive.

```
`typescript  
import { Component } from '@angular/core';  
@Component({  
  selector: 'app-root',  
  styleUrls: ['app/app.component.css'],  
  templateUrl: 'app/app.component.html'  
})  
export class AppComponent {  
  public ajaxSettings: object;
```

```
public hostUrl: string = 'http://localhost:{port}/';  
public ngOnInit(): void {  
  // Initializing the File Manager with Google Drive service.  
  this.ajaxSettings = {  
    // Replace the hosted port number in the place of "{port}"  
    url = this.hostUrl + "api/GoogleDriveProvider/GoogleDriveFileOperations",  
    downloadUrl = this.hostUrl + "api/GoogleDriveProvider/GoogleDriveDownload",  
    uploadUrl = this.hostUrl + "api/GoogleDriveProvider/GoogleDriveUpload",  
    getImageUrl = this.hostUrl + "api/GoogleDriveProvider/GoogleDriveGetImage"  
  };  
}  
}
```

> **Note:** To learn more about the file actions that can be performed with Google Drive file system provider, refer to this [link](#)

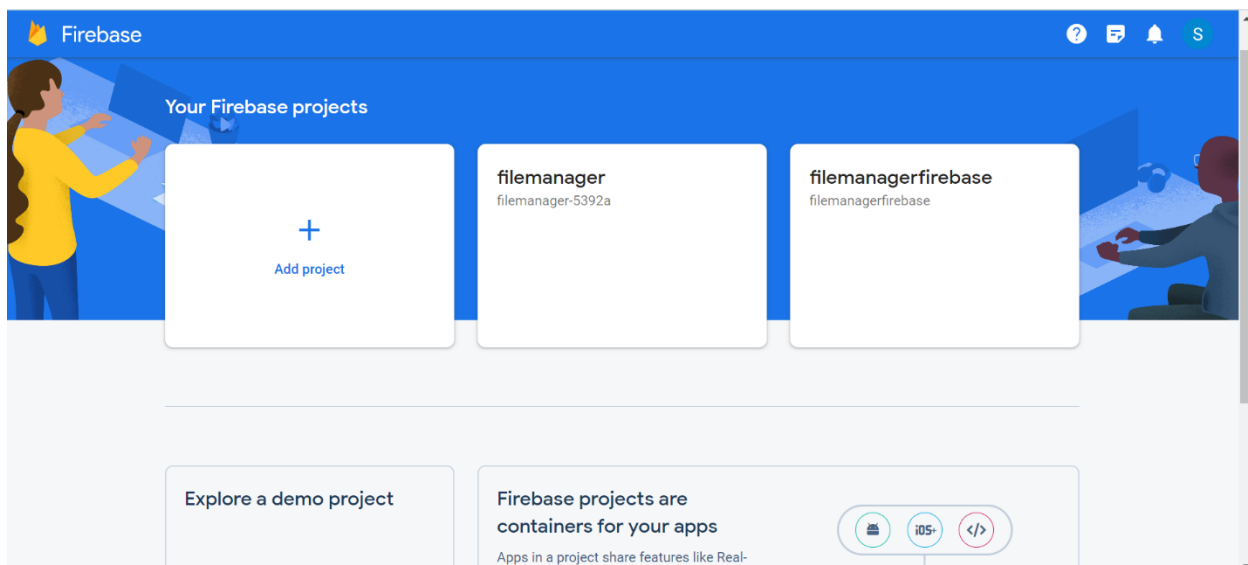
Firebase Realtime Database file system provider

The [Firebase Realtime Database](#) file system provider in **ASP.NET Core** provides the efficient way to store the File Manager file system in a cloud database as JSON representation.

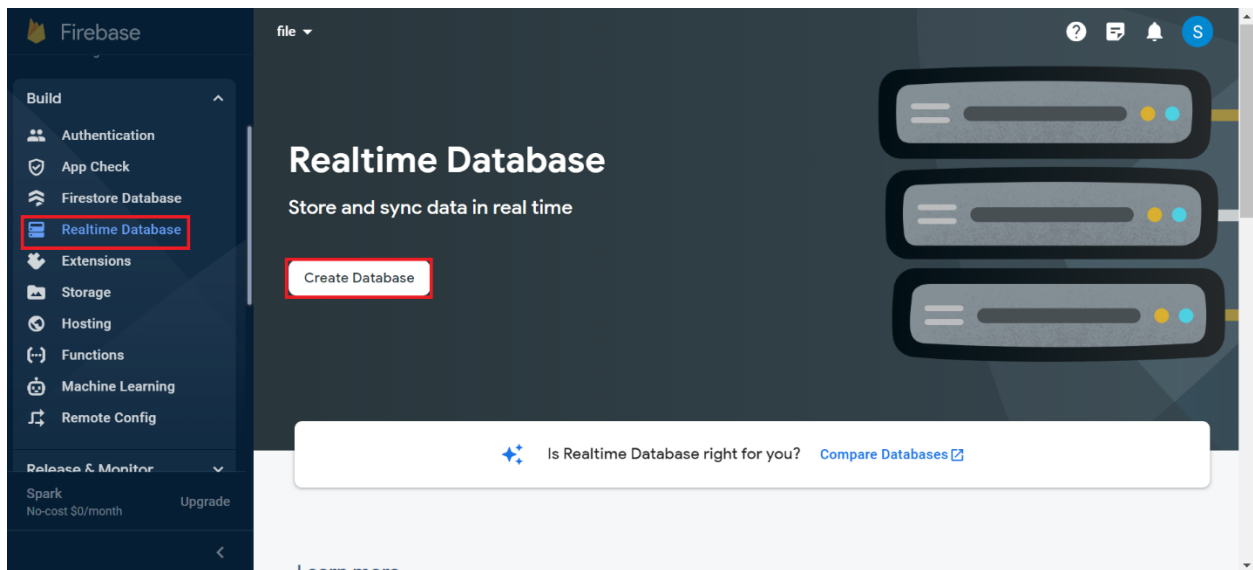
Generate Secret access key from service account

Follow the given steps to generate the secret access key:

- To access the Firebase console, please click on this [link](#). Once you have accessed the console, you can create a new project by filling in the necessary fields and clicking on the relevant buttons.



- Within the Firebase console, navigate to the **Build** tab. Under this tab, select the option for **Realtime Database**. From there, you can create a new database by clicking on the **Create Database** button.



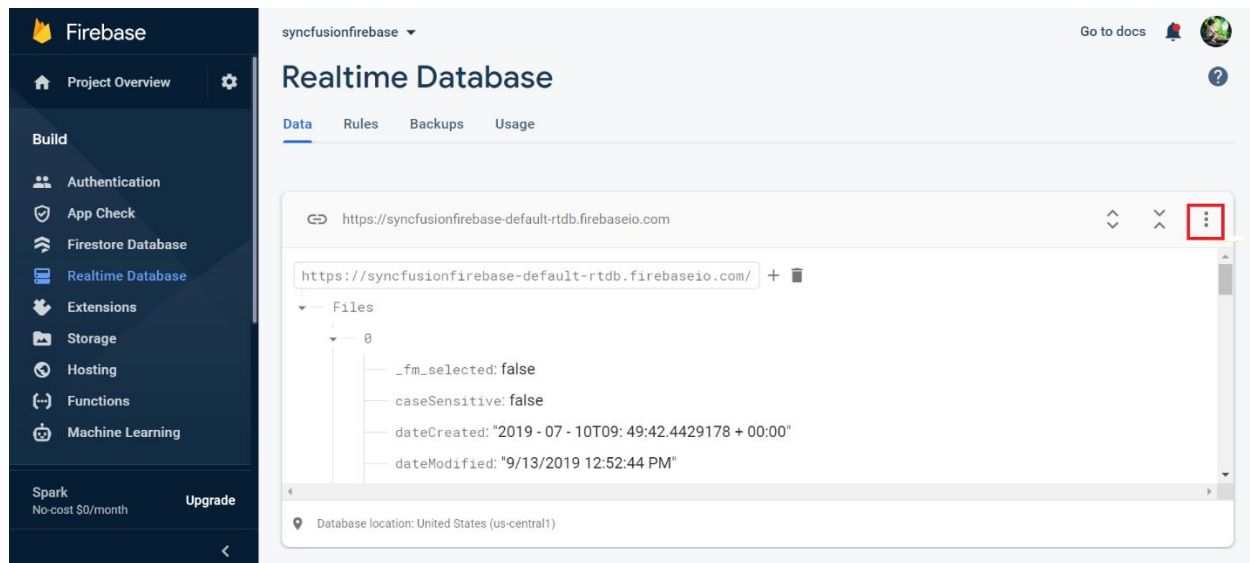
- To get started, create a root node and add any desired children to it. Please refer to the following code snippet for guidance on the structure of the JSON:

```
`typescript
{
  "Files" : [ {
    "caseSensitive" : false,
    "dateCreated" : "8/22/2019 5:17:55 PM",
    "dateModified" : "8/22/2019 5:17:55 PM",
    "filterId" : "0/",
    "filterPath" : "/",
    "hasChild" : false,
    "id" : "5",
    "isFile" : false,
    "isRoot" : true,
    "name" : "Music",
    "parentId" : "0",
    "selected" : false,
    "showHiddenItems" : false,
    "size" : 0,
```

```
"type" : "folder"
},
{
  "caseSensitive" : false,
  "dateCreated" : "8/22/2019 5:18:03 PM",
  "dateModified" : "8/22/2019 5:18:03 PM",
  "filterId" : "0/",
  "filterPath" : "/",
  "hasChild" : false,
  "id" : "6",
  "isFile" : false,
  "isRoot" : true,
  "name" : "videos",
  "parentId" : "0",
  "selected" : false,
  "showHiddenItems" : false,
  "size" : 0,
  "type" : ""
}]
}
```

Here, the `Files` denotes the `rootNode` and the subsequent object refers to the children of the root node. `rootNode` will be taken as the root folder of the file system loaded which will be loaded in File Manager component.

- To import a JSON file into the Firebase Realtime Database, navigate to the **Data** tab and click on the action icon shown in the accompanying image. From there, select the **Import JSON** option and upload the JSON file that was created using the code provided above.

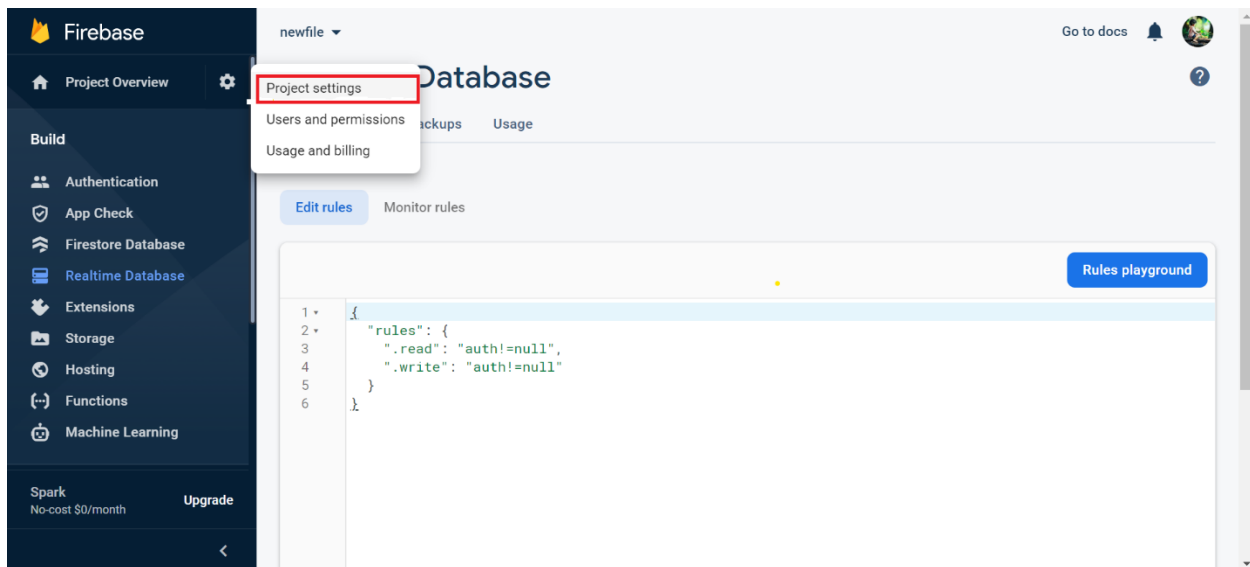


- To interact with the Firebase Realtime Database through your application, it is necessary to grant read and write permissions by defining appropriate rules in the Firebase project's **Rules tab**, as shown in the following code snippet. Once you have specified the rules, you can publish them by clicking the **Publish** button to enable the necessary authentication.

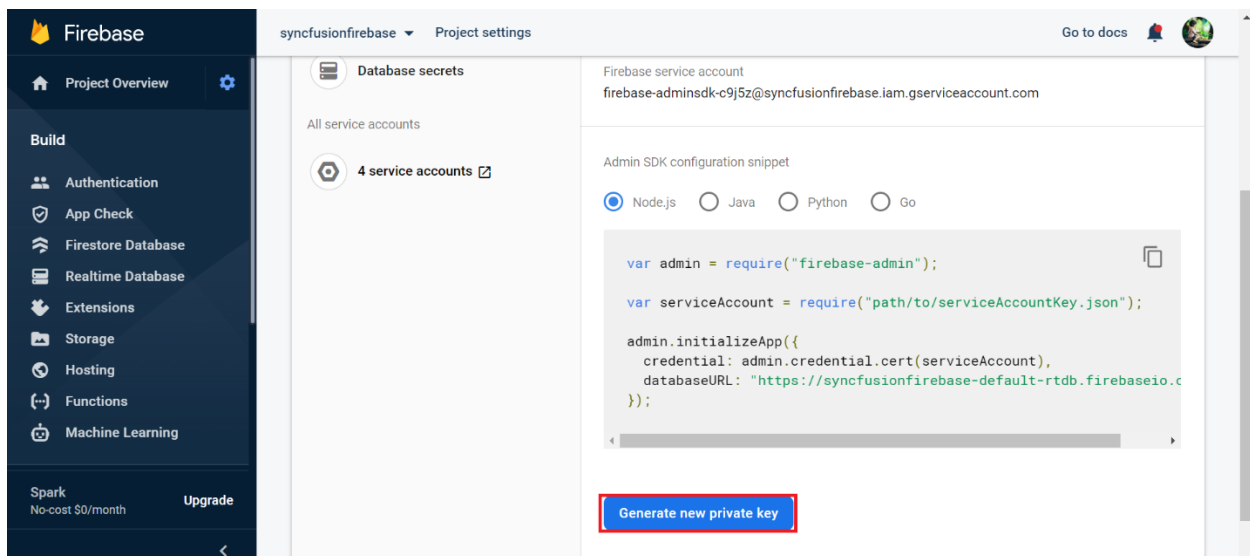
```
`typescript
{
/ Visit https://firebase.google.com/docs/database/security to learn more about security rules. /
"rules": {
".read": "auth!=null",
".write": "auth!=null"
}
}
`
```

Note: By default, rules of a Firebase project will be **false**. To read and write the data, configure the **Rules** as given in the following code snippet in the *Rules* tab in the Firebase Realtime Database project.

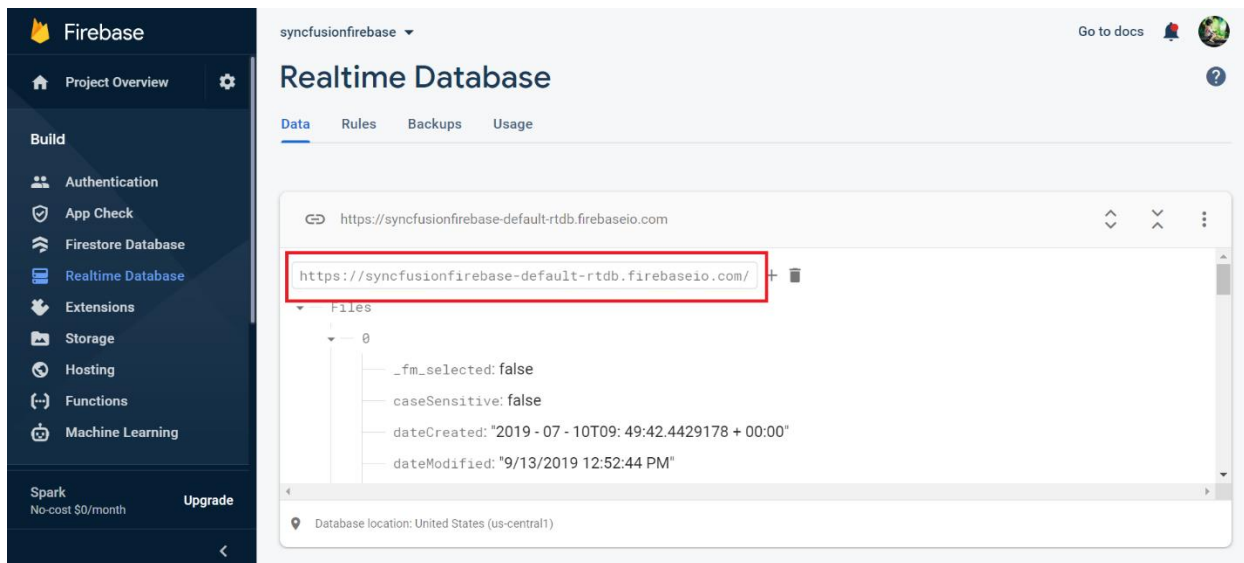
- Navigate to the project settings as instructed and then click on the **Service Account** tab.



- To obtain the access key JSON file, simply click on the **Generate new private key** button and then confirm by clicking the **Generate key** button in the pop-up window that appears.



- Next, you will need to clone the [firebase-realtime-database-apscore-file-provider](#) repository. Once cloned, simply open the project in Visual Studio and restore the NuGet package.
- Once you have generated the secret key, you will need to replace the JSON in the `access_key.json` file in the Firebase Realtime Database provider project with the newly generated key. This will enable authentication and allow you to perform read and write operations.
- In the **Data** tab, locate the project API URL and then paste it into the below mentioned section.



Register the Firebase Realtime Database by assigning *Firebase Realtime Database REST API link*, *rootNode*, and *serviceAccountKeyPath* parameters in the `RegisterFirebaseRealtimeDB` method of class `FirebaseRealtimeDBFileProvider` in the controller part of the ASP.NET Core application.

```
`typescript
```

```
this.operation.RegisterFirebaseRealtimeDB(string apiUrl, string rootNode, string serviceAccountKeyPath)
```

```
,
```

Example:

```
`typescript
```

```
this.operation.RegisterFirebaseRealtimeDB("{copy your API URL here}", "Files",  
hostingEnvironment.ContentRootPath + "\\FirebaseRealtimeDBHelper\\access_key.json");
```

```
,
```

In the above code,

- `{copy your API URL here}` denotes Firebase Realtime Database REST API link.
- `Files` denotes newly created root node in Firebase Realtime Database.
- `hostingEnvironment.ContentRootPath + "\\FirebaseRealtimeDBHelper\\access_key.json` denotes service account key path which has authentication key for the Firebase Realtime Database data.

After configuring the Firebase Realtime Database service link, build and run the project. Now, the project will be hosted in `http://localhost:{port}` and just mapping the **ajaxSettings** property of the File Manager component to the appropriate controller methods allows to manage the files in the Firebase Realtime Database.

```
`typescript
```

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

```
@Component({
```

```

selector: 'app-root',
styleUrls: ['app/app.component.css'],
templateUrl: 'app/app.component.html'
})
export class AppComponent {
  public ajaxSettings: object;
  public baseUrl: string = 'http://localhost:{port}/';
  public ngOnInit(): void {
    // Initializing File Manager with Firebase Realtime Database service.
    this.ajaxSettings = {
      // Replace the hosted port number in the place of "{port}"
      url: this.baseUrl + "api/FirebaseProvider/FirebaseRealtimeFileOperations",
      downloadUrl: this.baseUrl + "api/FirebaseProvider/FirebaseRealtimeDownload",
      uploadUrl: this.baseUrl + "api/FirebaseProvider/FirebaseRealtimeUpload",
      getImageUrl: this.baseUrl + "api/FirebaseProvider/FirebaseRealtimeGetImage"
    };
  }
}

```

> **Note:** To learn more about the file actions that can be performed with Firebase Realtime Database file system provider, refer to this [link](#)

Localization in Angular File manager component

The file manager can be localized to any culture by defining the texts and messages of the file manager in the corresponding culture. The default locale of the file manager is `en`(English). The following table represents the default texts and messages of the file manager in `en` culture.

KEY	Text/Message
----	----
NewFolder	New folder
Upload	Upload
Delete	Delete
Rename	Rename
Download	Download
Cut	Cut
Copy	Copy

Paste	Paste
SortBy	Sort by
Refresh	Refresh
Item-Selection	item selected
Items-Selection	items selected
View	View
Details	Details
SelectAll	Select all
Open	Open
Tooltip-NewFolder	New folder
Tooltip-Upload	Upload
Tooltip-Delete	Delete
Tooltip-Rename	Rename
Tooltip-Download	Download
Tooltip-Cut	Cut
Tooltip-Copy	Copy
Tooltip-Paste	Paste
Tooltip-SortBy	Sort by
Tooltip-Refresh	Refresh
Tooltip-Selection	Clear selection
Tooltip-View	View
Tooltip-Details	Details
Tooltip-SelectAll	Select all
Name	Name
Size	Size
DateModified	Modified
DateCreated	Date created
Path	Path
Created	Created
Modified	Modified
Location	Location
Type	Type
Permission	Permission

Ascending	Ascending
Descending	Descending
None	None
View-LargeIcons	Large icons
View-Details	Details
Search	Search
Button-Ok	OK
Button-Cancel	Cancel
Button-Yes	Yes
Button-No	No
Button-Create	Create
Button-Save	Save
Header-NewFolder	Folder
Content-NewFolder	Enter your folder name
Header-Rename	Rename
Content-Rename	Enter your new name
Header-Rename-Confirmation	Rename Confirmation
Content-Rename-Confirmation	If you change a file name extension
Are you sure you want to change it?	
Header-Delete	Delete File
Content-Delete	Are you sure you want to delete this file?
Header-Multiple-Delete	Delete Multiple Files
Content-Multiple-Delete	Are you sure you want to delete these {0} files?
Header-Folder-Delete	Delete Folder
Content-Folder-Delete	Are you sure you want to delete this folder?
Header-Duplicate	File exists
Content-Duplicate	already exists. Are you sure you want to replace it?
Header-Upload	Upload Files
Error	Error
Validation-Empty	The file or folder name cannot be empty.
Validation-Invalid	The file or folder name {0} contains invalid characters. Please use a different name.
Valid file or folder names cannot end with a dot or space, and cannot contain any of the following	
characters: \\/:.*?\"<>\\|	
Validation-NewFolder-Exists	A file or folder with the name {0} already exists.

|Validation-Rename-Exists|Cannot rename {0} to {1}| destination already exists.|

|Folder-Empty|This folder is empty|

|File-Upload|Drag files here to upload|

|Search-Empty|No results found|

|Search-Key|Try with different keywords|

|Filter-Empty|No results found|

|Filter-Key|Try with different filter|

|Sub-Folder-Error|The destination folder is the subfolder of the source folder|

|Same-Folder-Error|The destination folder is the same as the source folder.|

|Access-Denied|Access Denied|

|Access-Details|You don't have permission to access this folder|

|Header-Retry|File Already Exists|

|Content-Retry|A file with this name already exists in this folder. What would you like to do?|

|Button-Keep-Both|Keep both|

|Button-Replace|Replace|

|Button-Skip|Skip|

|ApplyAll-Label|Do this for all current items|

|KB|KB|

|Access-Message|{0} is not accessible. You need permission to perform the {1} action.|

|Network-Error|NetworkError: Failed to send on XMLHttpRequest: Failed to load|

|Server-Error|ServerError: Invalid response from|

The below example shows adding the German culture locale(de-DE)

APP.COMPONENT.TS

```
import { BrowserModule } from '@angular/platform-browser'
import { NgModule } from '@angular/core'
import { FileManagerModule, NavigationPaneService, ToolbarService,
DetailsViewService } from '@syncfusion/ej2-angular-filemanager'
import { Component } from '@angular/core';
import { L10n } from '@syncfusion/ej2-base';
@Component({
  imports: [FileManagerModule, ],
  providers:[ NavigationPaneService, ToolbarService, DetailsViewService],
  standalone: true,
  selector: 'app-root',
  styleUrls: ['./app.component.css'],
  template: `<ejs-filemanager id='file-manager'
[ajaxSettings]='ajaxSettings' [locale]='locale'>
</ejs-filemanager>`
})
export class AppComponent {
```

```
public ajaxSettings?: object;
public locale?: string;
public baseUrl: string = 'https://ej2-aspcore-
service.azurewebsites.net/';
public ngOnInit(): void {
  L10n.load({
    'de': {
      'filemanager': {
        "NewFolder": "Neuer Ordner",
        "Upload": "Hochladen",
        "Delete": "Löschen",
        "Rename": "Umbenennen",
        "Download": "Herunterladen",
        "Cut": "Schnitt",
        "Copy": "Kopieren",
        "Paste": "Einfügen",
        "SortBy": "Sortiere nach",
        "Refresh": "Aktualisierung",
        "Item-Selection": "Artikel ausgewählt",
        "Items-Selection": "Elemente ausgewählt",
        "View": "Aussicht",
        "Details": "Einzelheiten",
        "SelectAll": "Wählen Sie Alle",
        "Open": "Öffnen",
        "Tooltip-NewFolder": "Neuer Ordner",
        "Tooltip-Upload": "Hochladen",
        "Tooltip-Delete": "Löschen",
        "Tooltip-Rename": "Umbenennen",
        "Tooltip-Download": "Herunterladen",
        "Tooltip-Cut": "Schnitt",
        "Tooltip-Copy": "Kopieren",
        "Tooltip-Paste": "Einfügen",
        "Tooltip-SortBy": "Sortiere nach",
        "Tooltip-Refresh": "Aktualisierung",
        "Tooltip-Selection": "Auswahl aufheben",
        "Tooltip-View": "Aussicht",
        "Tooltip-Details": "Einzelheiten",
        "Tooltip-SelectAll": "Wählen Sie Alle",
        "Name": "Name",
        "Size": "Größe",
        "DateModified": "Geändert",
        "DateCreated": "Datum erstellt",
        "Path": "Pfad",
        "Modified": "Geändert",
        "Created": "Erstellt",
        "Location": "Ort",
        "Type": "Art",
        "Permission": "Genehmigung",
        "Ascending": "Aufsteigend",
        "Descending": "Absteigend",
        "None": "Keiner",
        "View-LargeIcons": "Große Icons",
        "View-Details": "Einzelheiten",
        "Search": "Suche",
        "Button-Ok": "OK",
        "Button-Cancel": "Stornieren",
        "Button-Yes": "Ja",
```

```

        "Button-No": "Nein",
        "Button-Create": "Erstellen",
        "Button-Save": "Sparen",
        "Header-NewFolder": "Mappe",
        "Content-NewFolder": "Geben Sie Ihren Ordnernamen ein",
        "Header-Rename": "Umbenennen",
        "Content-Rename": "Geben Sie Ihren neuen Namen ein",
        "Header-Rename-Confirmation": "Bestätigung umbenennen",
        "Content-Rename-Confirmation": "Wenn Sie eine
Dateinamenerweiterung ändern, wird die Datei möglicherweise instabil. Möchten
Sie sie wirklich ändern?",
        "Header-Delete": "Datei löschen",
        "Content-Delete": "Möchten Sie diese Datei wirklich löschen?",
        "Header-Multiple-Delete": "Mehrere Dateien löschen",
        "Content-Multiple-Delete": "Möchten Sie diese {0} Dateien
wirklich löschen?",
        "Header-Folder-Delete": "Lösche Ordner",
        "Content-Folder-Delete": "Möchten Sie diesen Ordner wirklich
löschen?",
        "Header-Duplicate": "Datei / Ordner existiert",
        "Content-Duplicate": "{0} existiert bereits. Möchten Sie
umbenennen und einfügen?",
        "Header-Upload": "Daten hochladen",
        "Error": "Error",
        "Validation-Empty": "Der Datei - oder Ordnername darf nicht leer
sein.",
        "Validation-Invalid": "Der Datei- oder Ordnername {0} enthält
ungültige Zeichen. Bitte verwenden Sie einen anderen Namen. Gültige Datei-
oder Ordnernamen dürfen nicht mit einem Punkt oder Leerzeichen enden und
keines der folgenden Zeichen enthalten: \\ /: *? \" < > | ",
        "Validation-NewFolder-Exists": "Eine Datei oder ein Ordner mit
dem Namen {0} existiert bereits.",
        "Validation-Rename-Exists": "{0} kann nicht in {1} umbenannt
werden: Ziel existiert bereits.",
        "Folder-Empty": "Dieser Ordner ist leer",
        "File-Upload": "Dateien zum Hochladen hierher ziehen",
        "Search-Empty": "Keine Ergebnisse gefunden",
        "Search-Key": "Versuchen Sie es mit anderen Stichwörtern",
        "Filter-Empty": "keine Ergebnisse gefunden",
        "Filter-Key": "Versuchen Sie es mit einem anderen Filter",
        "Sub-Folder-Error": "Der Zielordner ist der Unterordner des
Quellordners.",
        "Same-Folder-Error": "Der Zielordner ist derselbe wie der
Quellordner.",
        "Access-Denied": "Zugriff verweigert",
        "Access-Details": "Sie haben keine Berechtigung, auf diesen
Ordner zuzugreifen.",
        "Header-Retry": "Die Datei existiert bereits",
        "Content-Retry": "In diesem Ordner ist bereits eine Datei mit
diesem Namen vorhanden. Was möchten Sie tun?",
        "Button-Keep-Both": "Behalte beides",
        "Button-Replace": "Ersetzen",
        "Button-Skip": "Überspringen",
        "ApplyAll-Label": "Mache das für alle aktuellen Artikel",
        "KB": "KB",
        "Access-Message": "{0} ist nicht zugänglich. Sie benötigen die
Berechtigung, um die Aktion {1} auszuführen.",

```

```

        "Network-Error": "NetworkError: Fehler beim Senden auf
XMLHttpRequest: Fehler beim Laden",
        "Server-Error": "ServerError: Ungültige Antwort von"
    }
}
});
this.ajaxSettings = {
    url: this.hostUrl + 'api/FileManager/FileOperations',
    getImageUrl: this.hostUrl + 'api/FileManager/GetImage',
    uploadUrl: this.hostUrl + 'api/FileManager/Upload',
    downloadUrl: this.hostUrl + 'api/FileManager/Download'
};
this.locale = 'de';
}
}

```

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 File Manager component

File Manager's UI virtualization allows you for the dynamic loading of a large number of directories and files in both the detailsView and largeIconsView without degrading its performance.

Module Injection

In order to use UI Virtualization, you 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 { FileManagerComponent, VirtualizationService } from '@syncfusion/ej2-angular-filemanager';
@NgModule({
  imports: [
    BrowserModule
  ],
  declarations: [AppComponent],
  bootstrap: [AppComponent],
  providers: [VirtualizationService]
})
export class AppModule { }

```

Enable Virtualization

The virtualization of the File Manager component is based on the height and width of the viewport. The items will be loaded in both [largelconsView](#) and [detailsView](#) based on the viewport size.

In order to enable `virtualization`, you must set the [enableVirtualization](#) property to true.

In the instance below, a sizable collection of files can be found in the folders "Documents" and "Text Documents".

APP.COMPONENT.TS

```
import { BrowserModule } from '@angular/platform-browser'
import { NgModule } from '@angular/core'
import { FileManagerModule, NavigationPaneService, ToolbarService,
DetailsViewService, VirtualizationService } from '@syncfusion/ej2-angular-
filemanager'
import { Component, ViewEncapsulation } from '@angular/core';
import { FileManagerComponent, NavigationPaneService, ToolbarService,
DetailsViewService, VirtualizationService } from '@syncfusion/ej2-angular-
filemanager';
/**
 * File Manager virtualization feature sample
 */
@Component({
imports: [FileManagerModule, ],
providers:[ NavigationPaneService, ToolbarService, DetailsViewService,
VirtualizationService ],
standalone: true,
selector: 'app-root',
styleUrls: ['./app.component.css'],
templateUrl: './default.html',
encapsulation: ViewEncapsulation.None,
providers: [ NavigationPaneService, ToolbarService, DetailsViewService,
VirtualizationService]
})
export class AppComponent {
public ajaxSettings?: object;
public view?: string;
public hostUrl: string = 'https://ej2-aspcore-
service.azurewebsites.net/';
public ngOnInit(): void {
this.ajaxSettings = {
url: this.hostUrl + 'api/FileManager/FileOperations',
getImageUrl: this.hostUrl + 'api/FileManager/GetImage',
uploadUrl: this.hostUrl + 'api/FileManager/Upload',
downloadUrl: this.hostUrl + 'api/FileManager/Download'
};
this.view = "Details";
}
onBeforeSend(args: any) {
args.ajaxSettings.beforeSend = function (args: any) {
args.httpRequest.setRequestHeader('Authorization',
'FileBrowser');
};
}
```

```

    beforeImageLoad(args: any) {
      args.imageUrl = args.imageUrl + '&rootName=' + 'FileBrowser';
    }
    beforeDownload(args: any) {
      args.data.rootFolderName = 'FileBrowser';
    }
  }
}

```

MAIN.TS

```

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

```

Limitations for Virtualization

- Programmatic selection using the **selectAll** method is not supported with virtual scrolling.
- The keyboard shortcut **CTRL+A** will only select the files and directories that are currently visible within the viewport, rather than selecting all files and directories in the entire directory tree.
- Selected file items are not maintained while scrolling, considering the performance of the component.

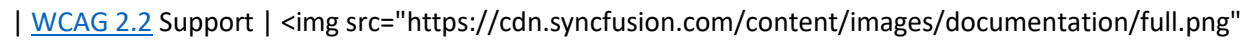
Accessibility in Angular File Manager component

The File Manager component followed the accessibility guidelines and standards, including [ADA](#), [Section 508](#), [WCAG 2.2](#) standards, and [WCAG roles](#) that are commonly used to evaluate accessibility.

The accessibility compliance for the File Manager component is outlined below.

| Accessibility Criteria | Compatibility |


| -- | -- |

| [WCAG 2.2](#) Support |  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 |  |  
  
| Axe-core Accessibility Validation |  |  
  
<style>  
.post .post-content img {  
display: inline-block;  
margin: 0.5em 0;  
}  
</style>  
  
<div> - All  
features of the component meet the requirement.</div>  
  
<div> - Some features of the component do not meet the requirement.</div>  
  
<div> - The component does not meet the requirement.</div>
```

WAI-ARIA attributes

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

Attributes	Purpose
---	---
<code>role</code>	Used to convey a significant and contextual message to the user.
<code>aria-disabled</code>	Indicates whether the File Manager component is in disabled state.
<code>aria-haspopup</code>	Indicates whether the toolbar item has a popup list or not.
<code>aria-orientation</code>	Indicates whether the File Manager element is oriented horizontally or vertically.
<code>aria-expanded</code>	Indicates whether the Treeview node has been expanded.
<code>aria-owns</code>	Contains the ID of the suggestion list to indicate popup as a child element.
<code>aria-activedescendent</code>	Holds the ID of the active list item to focus its descendant child element.
<code>aria-level</code>	Specifies the level of the element in Treeview Structure.
<code>aria-selected</code>	Indicates whether a particular node is in selected state.
<code>aria-placeholder</code>	Represents a hint (word or phrase) to the user about what to enter in the text field.
<code>aria-label</code>	Provides an accessible name for the element.
<code>aria-checked</code>	Indicates whether the checkbox is in checked state.

| **aria-labelledby** | Provides a label for the dialog. Typically, the "aria-labelledby" attribute will contain the id of the element used as the dialog's title. |

| **aria-describedby** | This attribute points to the Dialog element describing the one it's set on. |

| **aria-modal** | Indicates whether an element is a modal when display. |

| **aria-colcount** | Specifies the number of columns in full table. |

| **aria-colindexnt** | Defines the number of columns within a table in details view. |

| **aria-rowspan** | Defines the number of rows a cell spanned within a table in details view. |

| **aria-colspan** | Defines the number of columns a cell spanned within a table in details view. |

| **aria-sort** | Indicates whether items in the table are sorted in ascending or descending order. |

| **aria-grabbed** | When the folder/file item is chosen for dragging, the aria-grabbed attribute is set to "true." If it's set to "false," the element can be grabbed for drag-and-drop, but it won't be actively held. |

| **aria-busy** | This attribute is set to false when table content is loaded. |

| **aria-multiselectable** | Defines more than one item has been selected. |

Keyboard interaction

The File Manager component followed the **keyboard interaction** guidelines, making it easy for people who use assistive technologies (AT) and those who completely rely on keyboard navigation. The following keyboard shortcuts are supported by the File Manager component.

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

| --- | --- |

| **Page Down** | Scrolls down to the next folder or file and selects the first item when files are loaded. |

| **Page Up** | Scrolls up to previous folder and select the first item when files are loaded. |

| **Enter** | Selects the focused item and navigate through the child elements. |

| **Tab** | Focuses on the first element of toolbar and navigates through the next tab indexed element. |

| **Esc(Escape)** | Closes the image when it is in open state. |

| **Alt+N** | Creates a new folder dialog. |

| **F5** | Refresh the file manager element. |

| **Home** | Navigate through the first element of details view or large icons view. |

| **End** | Navigate through the last element of details view or large icons view. |

| **Move Left** | Scrolls left to the previous folder and select the first item when files are loaded |

| **Move Right** | Scrolls right to the previous folder and select the first item when files are loaded |

- | Alt+Enter | Shows the get details info for selected folder. |
- | Shift+Right | Allows multiselection. Select the file or folder at the right of the previously selected folder. |
- | Shift+Left | Allows multiselection. Select the file or folder at the left of the previously selected folder. |
- | Shift+Down | Allows multiselection. Select the file or folder till the focused index. |
- | Shift+Delete | Permanently deletes the selected file or folder in the file manager element. |
- | Delete | Deletes the selected file or folder in the file manager element. |
- | Shift+Up | Allows multiselection. Select the file or folder till the focused index. |
- | Ctrl+C | Copies the selected file or folder in the file manager element. |
- | Ctrl+V | Pastes the copied/cut file or folder in the file manager element. |
- | Ctrl+X | Cuts the selected file or folder in the file manager element. |
- | Ctrl+A | Select all the files or folders in the details view or large icons view. |
- | F2 | Creates a rename dialog for a selected file or folder in the file manager element. |
- | Shift+F10 | Opens the context menu for the selected file or folder in the file manager element. |
- | Ctrl+D | Downloads the list of selected files or folders in the file manager element. |
- | Ctrl+Shift+1 | Changes the file manager layout to details view. |
- | Ctrl+Shift+2 | Changes the file manager layout to large icons view. |

Ensuring accessibility

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

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

See also

- [Accessibility in Syncfusion Angular components](#)

Access control in Angular File manager component

The FileManager allows you to define access permissions for folders and files using a set of access rules to user(s).

- [Access Rules](#)
- [Permissions](#)

Access Rules

The FileAccessController allows you to define security permissions for folders and files using a set of folder or file access rules.

To set up access rules for folders (including their files and sub-folders) and individual files, use the SetRules() method in the controller. The following table represents the AccessRule properties available for file and folder:

Properties	Applicable for file	Applicable for folder	Description
---	---	---	---
Copy	Yes	Yes	Allows access to copy a file or folder.
Read	Yes	Yes	Allows access to read a file or folder.
Write	Yes	Yes	Allows permission to write a file or folder.
WriteContents	No	Yes	Allows permission to write the content of folder.
Download	Yes	Yes	Allows permission to download a file or folder.
Upload	No	Yes	Allows permission to upload to the folder.
Path	Yes	Yes	Specifies the path to apply the rules, which are defined.
Role	Yes	Yes	Specifies the role to which the rule is applied.
IsFile	Yes	Yes	Specifies whether the rule is specified for folder or file.

The following syntax represents the access Rules for Administrator using file or folder.

```
`typescript
//Administrator
//Access Rules for File
new AccessRule { Path = "/.", Role = "Administrator", Read = Permission.Allow, Write =
Permission.Allow, Copy = Permission.Allow, Download = Permission.Allow, IsFile = true },
// Access Rules for folder
new AccessRule { Path = "**", Role = "Administrator", Read = Permission.Allow, Write = Permission.Allow,
Copy = Permission.Allow, WriteContents = Permission.Allow, Upload = Permission.Allow, Download =
Permission.Deny, IsFile = false },
`
```

The following syntax represent the access Rules for Default user using file or folder.

```
`typescript
//Default User
//Access Rules for File
new AccessRule { Path = "/.", Role = "Default User", Read = Permission.Deny, Write = Permission.Deny,
Copy = Permission.Deny, Download = Permission.Deny, IsFile = true },
// Access Rules for folder
```

```
new AccessRule { Path = "*", Role = "Default User", Read = Permission.Deny, Write = Permission.Deny,
Copy = Permission.Deny, WriteContents = Permission.Deny, Upload = Permission.Deny, Download =
Permission.Deny, IsFile = false },
```

,

Permissions

It helps to explain how to apply security permission to file manager file or folder using access rules. The following table represent the value that determines the permission.

Value	Description
-------	-------------

---	---
-----	-----

Allow	Allows you to do read, write, copy, and download operations.
-------	--------------------------------------------------------------

Deny	Denies you to do read, write, copy, and download operations.
------	--------------------------------------------------------------

Use the **Role** property to apply created roles to the file manager. After that, the file manager displays folder or file and allow permission based on assigned roles.

The following syntax represent how to apply permission based on assigned roles

Permission denied for administrator to write a file or folder.

```
`typescript
```

```
// For file
```

```
new AccessRule { Path = "/", Role = "Administrator", Read = Permission.Allow, Write = Permission.Deny,
IsFile = true},
```

```
// For folder
```

```
new AccessRule { Path = "*", Role = "Administrator", Read = Permission.Allow, Write = Permission.Deny,
IsFile = false},
```

,

The following syntax represent how to allow or deny permission based on file or folder access rule.

Permission denied for writing except for particular file or folder.

```
`typescript
```

```
// Deny writing for particular folder
```

```
new AccessRule { Path = "/Documents", Role = "Document Manager", Read = Permission.Allow, Write =
Permission.Deny, Copy = Permission.Allow, WriteContents = Permission.Deny, Upload =
Permission.Deny, Download = Permission.Deny, IsFile = false },
```

```
// Deny writing for particular file
```

```
new AccessRule { Path = "/Pictures/Employees/Adam.png", Role = "Document Manager", Read =
Permission.Allow, Write = Permission.Deny, Copy = Permission.Deny, Download = Permission.Deny,
IsFile = true },
```

,

Permission denied for writing and uploading in root folder.

```
` typescript
```

```
// Folder Rule
```

```
new AccessRule { Path = "/", Role = "Document Manager", Read = Permission.Allow, Write =
Permission.Deny, Copy = Permission.Deny, WriteContents = Permission.Deny, Upload =
Permission.Deny, Download = Permission.Deny, IsFile = false },
```

The following example demonstrate the file manager rendered with access control support.

APP.COMPONENT.TS

```
import { BrowserModule } from '@angular/platform-browser'
import { NgModule } from '@angular/core'
import { FileManagerModule, NavigationPaneService, ToolbarService,
DetailsViewService } from '@syncfusion/ej2-angular-filemanager'
import { Component } from '@angular/core';
@Component({
  imports: [FileManagerModule, ],
  providers: [NavigationPaneService, ToolbarService, DetailsViewService],
  standalone: true,
  selector: 'app-root',
  styleUrls: ['./app.component.css'],
  template: `<ejs-filemanager id='file-manager'
[ajaxSettings]='ajaxSettings' [view]='view'>
  </ejs-filemanager>`
})
export class AppComponent {
  public ajaxSettings?: object | any;
  public view?: string;
  public hostUrl: string = 'https://ej2-aspcore-
service.azurewebsites.net/';
  public ngOnInit(): void {
    this.ajaxSettings = {
      url: this.hostUrl + 'api/FileManagerAccess/FileOperations',
      getImageUrl: this.hostUrl + 'api/FileManagerAccess/GetImage',
      uploadUrl: this.hostUrl + 'api/FileManagerAccess/Upload',
      downloadUrl: this.hostUrl + 'api/FileManagerAccess/Download'
    };
    this.view = "Details";
  }
}
```

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

Adding custom item to context menu in Angular File manager component

The context menu can be customized using the [contextMenuSettings](#), [menuOpen](#), and [menuClick](#) events.

The following example shows adding a custom item in the context menu.

The [contextMenuSettings](#) is used to add new menu item. The [menuOpen](#) event is used to add the icon to the new menu item. The [menuClick](#) event is used to add an event handler to the new menu item.

APP.COMPONENT.TS

```
import { BrowserModule } from '@angular/platform-browser'
import { NgModule } from '@angular/core'
import { FileManagerModule, NavigationPaneService, ToolbarService,
DetailsViewService } from '@syncfusion/ej2-angular-filemanager'
import { Component } from '@angular/core';
import { MenuOpenEventArgs, MenuClickEventArgs } from '@syncfusion/ej2-
filemanager';
@Component({
  imports: [FileManagerModule, ],
  providers:[ NavigationPaneService, ToolbarService, DetailsViewService],
  standalone: true,
  selector: 'app-root',
  styleUrls: ['./app.component.css'],
  template: `<ejs-filemanager id='filemanager'
[ajaxSettings]='ajaxSettings' [contextMenuSettings]='contextMenuSettings'
(menuOpen)='menuOpen($event)' (menuClick)='menuClick($event)'>
</ejs-filemanager>`
})
export class AppComponent {
  public ajaxSettings?: object;
  public contextMenuSettings?: object;
  public baseUrl: string = 'https://ej2-aspcore-
service.azurewebsites.net/';
  public ngOnInit(): void {
    this.ajaxSettings = {
      url: this.baseUrl + 'api/FileManager/FileOperations',
      getImageUrl: this.baseUrl + 'api/FileManager/GetImage',
      uploadUrl: this.baseUrl + 'api/FileManager/Upload',
      downloadUrl: this.baseUrl + 'api/FileManager/Download'
    };
    this.contextMenuSettings = {
      file: ['Custom', 'Open', '|', 'Delete', 'Rename', '|',
'Details'],
      folder: ['Custom', 'Open', '|', 'Delete', 'Rename', '|',
'Details', 'Custom'],
      layout: ['Custom', 'SortBy', 'View', 'Refresh', '|', 'NewFolder',
'Upload', '|', 'Details', '|', 'SelectAll'],
      visible: true
    };
  }
  menuOpen(args: MenuOpenEventArgs | any) {
    for(var i=0;i<args.items.length;i++) {
      if (args.items[i].text === 'Custom') {
        args.items[i].iconCss= 'e-icons e-fe-tick';
      }
    }
  }
}
```

```

    }
  }
  // event for custom menu item
  menuClick(args: MenuClickEventArgs | any) {
    if (args.item.text === 'Custom') {
      alert('You have clicked custom menu item')
    }
  }
}

```

MAIN.TS

```

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

```

Adding custom item to toolbar in Angular File manager component

You can modify the items displayed in the toolbar by utilizing the [toolbarItems](#) API. To display both default and customized items, it's essential to assign a unique **name** to each item. Additionally, you have the flexibility to alter the default items by adjusting properties such as **tooltipText**, **iconCss**, **Text**, **suffixIcon** and more. This level of customization allows you to tailor the toolbar to your specific requirements and design preferences. The names used in the code example below serve as unique identifiers for default toolbar items, while custom items can be assigned any unique name value to distinguish them from the defaults.

For instance, here's an example of how to add a custom checkbox to the toolbar using the **template** property. Here we have modified the default **New Folder** item and added a custom toolbar item for selection.

APP.COMPONENT.TS

```

import { BrowserModule } from '@angular/platform-browser'
import { NgModule } from '@angular/core'
import { FileManagerModule, NavigationPaneService, ToolbarService,
  DetailsViewService } from '@syncfusion/ej2-angular-filemanager'
import { CheckBoxModule } from '@syncfusion/ej2-angular-buttons'
import { Component, ViewChild } from '@angular/core';
import { FileManagerComponent } from '@syncfusion/ej2-angular-filemanager';
import { CheckBoxComponent, ChangeEventArgs } from '@syncfusion/ej2-angular-
  buttons';
@Component({
  imports: [FileManagerModule, CheckBoxModule],
  providers:[ NavigationPaneService, ToolbarService, DetailsViewService],
  standalone: true,
  selector: 'app-root',
  styleUrls: ['./app.component.css'],
  template: `<ejs-filemanager id='files' #fileManager
[ajaxSettings]='ajaxSettings' >
    <e-toolbaritems>
      <e-toolbaritem name= "NewFolder" text= "Create folder"
prefixIcon= "e-plus" tooltipText= "Create folder" ></e-toolbaritem>
      <e-toolbaritem name= "Upload"></e-toolbaritem>
      <e-toolbaritem name= "SortBy"></e-toolbaritem>
    </e-toolbaritems>
  `
})
export class AppComponent {
  ajaxSettings = {
    toolbarItems: [
      { name: 'NewFolder', text: 'Create folder', prefixIcon: 'e-plus', tooltipText: 'Create folder' },
      { name: 'Upload' },
      { name: 'SortBy' }
    ]
  };
}

```



```

        <e-toolbaritem name= "Refresh"></e-toolbaritem>
        <e-toolbaritem name= "Cut"></e-toolbaritem>
        <e-toolbaritem name= "Copy"></e-toolbaritem>
        <e-toolbaritem name= "Paste"></e-toolbaritem>
        <e-toolbaritem name= "Delete"></e-toolbaritem>
        <e-toolbaritem name= "Download"></e-toolbaritem>
        <e-toolbaritem name= "Rename"></e-toolbaritem>
        <e-toolbaritem name= "Select">
            <ng-template #template>
                <ejs-checkbox #checkbox label="Select All"
[ checked]="false" (change)="onChange()" "></ejs-checkbox>
            </ng-template>
        </e-toolbaritem>
        <e-toolbaritem name= "Selection"></e-toolbaritem>
        <e-toolbaritem name= "View"></e-toolbaritem>
        <e-toolbaritem name= "Details"></e-toolbaritem>
    </e-toolbaritems>
</ejs-filemanager>`
}))
export class AppComponent {
    @ViewChild('fileManager')
    public fileManagerInstance?: FileManagerComponent;
    public checkbox: CheckBoxComponent;
    public ajaxSettings?: object;
    public hostUrl: string = 'https://ej2-aspcore-
service.azurewebsites.net/';
    public ngOnInit(): void {
        this.ajaxSettings = {
            url: this.hostUrl + 'api/FileManager/FileOperations',
            getImageUrl: this.hostUrl + 'api/FileManager/GetImage',
            uploadUrl: this.hostUrl + 'api/FileManager/Upload',
            downloadUrl: this.hostUrl + 'api/FileManager/Download'
        };
    };
    public onChange(args: ChangeEventArgs): void {
        if (args.checked) {
            (this.fileManagerInstance as FileManagerComponent).selectAll();
            (this.checkbox as CheckBoxComponent).label = 'Unselect All';
        }
        else {
            (this.fileManagerInstance as
FileManagerComponent).clearSelection();
            (this.checkbox as CheckBoxComponent).label = '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));

```

Enable disable toolbar item in Angular File manager component

The toolbar items can be enabled/disabled by specifying the items in [enableToolbarItems](#) or [disableToolbarItems](#) methods respectively.

The following example shows enabling and disabling toolbar items on button click.

APP.COMPONENT.TS

```
import { BrowserModule } from '@angular/platform-browser'
import { NgModule } from '@angular/core'
import { FileManagerModule, NavigationPaneService, ToolbarService,
DetailsViewService } from '@syncfusion/ej2-angular-filemanager'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component, ViewChild } from '@angular/core';
import { FileManager } from '@syncfusion/ej2-filemanager';
@Component({
  imports: [FileManagerModule, ButtonModule],
  providers: [NavigationPaneService, ToolbarService, DetailsViewService],
  standalone: true,
  selector: 'app-root',
  styleUrls: ['./app.component.css'],
  template: ` <button ejs-button id="enable" cssClass="e-success">Enable
New Folder toolbar item</button>
      <button ejs-button id="disable" cssClass="e-danger">Disable
New Folder toolbar item</button>
      <br />
      <br />
      <ejs-filemanager id='file-manager' #fileManager
[ajaxSettings]='ajaxSettings' [height]='height' (created)='onCreated($event)'
>
      </ejs-filemanager>`
})
export class AppComponent {
  @ViewChild('fileManager')
  public fileManagerInstance?: FileManager;
  public ajaxSettings?: object;
  public height?: number;
  public hostUrl: string = 'https://ej2-aspcore-
service.azurewebsites.net/';
  public ngOnInit(): void {
    this.ajaxSettings = {
      url: this.hostUrl + 'api/FileManager/FileOperations',
      getImageUrl: this.hostUrl + 'api/FileManager/GetImage',
      uploadUrl: this.hostUrl + 'api/FileManager/Upload',
      downloadUrl: this.hostUrl + 'api/FileManager/Download'
    };
    this.height = 330;
  };
  onCreated(args: any) {
    // Click event for enable button
    (document.getElementById("enable") as
HTMLElement).addEventListener('click', (event) => {
      // Enable new folder toolbar item
      (this.fileManagerInstance as
FileManager).enableToolbarItems(["newfolder"]);
    });
    // Click event for disable button
```

```
(document.getElementById("disable") as
HTMLElement).addEventListener('click', (event) => {
    // Disable new folder toolbar item
    (this.fileManagerInstance as
FileManager).disableToolbarItems(["newfolder"]);
});
}
```

MAIN.TS

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

Initialize filemanager using systemjs in Angular File manager component

File manager can also be initialized using **SystemJS** as follows

Installation and configuration

- To setup basic **Angular** sample use the following commands.

```
`sh
git clone https://github.com/angular/quickstart.git quickstart
cd quickstart
npm install
`
```

For more information, refer to [Angular sample setup](#).

- Install Syncfusion file manager packages using the below command.

```
`sh
npm install @syncfusion/ej2-angular-filemanager --save
`
```

The above package installs File Manager dependencies which are required to render the component in an Angular environment.

- Syncfusion **ej2-angular-filemanager** packages need to be mapped in **systemjs.config.js** configuration file.

```
`javascript
/
```

- System configuration for Angular samples

- Adjust as necessary for your application needs.

```
*/  
(function (global) {  
  System.config({  
    paths: {  
      // paths serve as alias  
      'npm:': 'node_modules/',  
      "syncfusion:": "node_modules/@syncfusion/", // syncfusion alias  
    },  
    // map tells the System loader where to look for things  
    map: {  
      // our app is within the app folder  
      'app': 'app',  
      // angular bundles  
      '@angular/core': 'npm:@angular/core/bundles/core.umd.js',  
      '@angular/common': 'npm:@angular/common/bundles/common.umd.js',  
      '@angular/compiler': 'npm:@angular/compiler/bundles/compiler.umd.js',  
      '@angular/platform-browser': 'npm:@angular/platform-browser/bundles/platform-browser.umd.js',  
      '@angular/platform-browser-dynamic': 'npm:@angular/platform-browser-dynamic/bundles/platform-browser-dynamic.umd.js',  
      '@angular/http': 'npm:@angular/http/bundles/http.umd.js',  
      '@angular/router': 'npm:@angular/router/bundles/router.umd.js',  
      '@angular/forms': 'npm:@angular/forms/bundles/forms.umd.js',  
      // syncfusion bundles  
      "@syncfusion/ej2-base": "syncfusion:ej2-base/dist/ej2-base.umd.min.js",  
      "@syncfusion/ej2-data": "syncfusion:ej2-data/dist/ej2-data.umd.min.js",  
      "@syncfusion/ej2-grids": "syncfusion:ej2-grids/dist/ej2-grids.umd.min.js",  
      "@syncfusion/ej2-layouts": "syncfusion:ej2-layouts/dist/ej2-layouts.umd.min.js",  
      "@syncfusion/ej2-navigations": "syncfusion:ej2-navigations/dist/ej2-navigations.umd.min.js",  
      "@syncfusion/ej2-inputs": "syncfusion:ej2-inputs/dist/ej2-inputs.umd.min.js",  
      "@syncfusion/ej2-compression": "syncfusion:ej2-compression/dist/ej2-compression.umd.min.js",  
      "@syncfusion/ej2-pdf-export": "syncfusion:ej2-pdf-export/dist/ej2-pdf-export.umd.min.js",  
      "@syncfusion/ej2-file-utils": "syncfusion:ej2-file-utils/dist/ej2-file-utils.umd.min.js",
```

```

"@syncfusion/ej2-popups": "syncfusion:ej2-popups/dist/ej2-popups.umd.min.js",
"@syncfusion/ej2-buttons": "syncfusion:ej2-buttons/dist/ej2-buttons.umd.min.js",
"@syncfusion/ej2-lists": "syncfusion:ej2-lists/dist/ej2-lists.umd.min.js",
"@syncfusion/ej2-splitbuttons": "syncfusion:ej2-splitbuttons/dist/ej2-splitbuttons.umd.min.js",
"@syncfusion/ej2-calendars": "syncfusion:ej2-calendars/dist/ej2-calendars.umd.min.js",
"@syncfusion/ej2-excel-export": "syncfusion:ej2-excel-export/dist/ej2-excel-export.umd.min.js",
"@syncfusion/ej2-dropdowns": "syncfusion:ej2-dropdowns/dist/ej2-dropdowns.umd.min.js",
"@syncfusion/ej2-filemanager": "syncfusion:ej2-filemanager/dist/ej2-filemanager.umd.min.js",
"@syncfusion/ej2-angular-base": "syncfusion:ej2-angular-base/dist/ej2-angular-base.umd.min.js",
"@syncfusion/ej2-angular-filemanager": "syncfusion:ej2-angular-filemanager/dist/ej2-angular-
filemanager.umd.min.js",
// other libraries
'rxjs':          'npm:rxjs',
'angular-in-memory-web-api': 'npm:angular-in-memory-web-api/bundles/in-memory-web-api.umd.js'
},
// packages tells the System loader how to load when no filename and/or no extension
packages: {
  app: {
    defaultExtension: 'js',
    meta: {
      './*.js': {
        loader: 'systemjs-angular-loader.js'
      }
    }
  },
  rxjs: {
    defaultExtension: 'js'
  }
}
});
})(this);
`

```

To render the file manager component, need to import file manager and its dependent component's styles as given below in `style.css`.

```
`css
@import '@syncfusion/ej2-base/styles/material.css';
@import '@syncfusion/ej2-navigations/styles/material.css';
@import '@syncfusion/ej2-layouts/styles/material.css';
@import '@syncfusion/ej2-dropdowns/styles/material.css';
@import '@syncfusion/ej2-inputs/styles/material.css';
@import '@syncfusion/ej2-lists/styles/material.css';
@import '@syncfusion/ej2-splitbuttons/styles/material.css';
@import '@syncfusion/ej2-popups/styles/material.css';
@import '@syncfusion/ej2-buttons/styles/material.css';
@import '@syncfusion/ej2-angular-filemanager/styles/material.css';
`
```

Note: If you want to refer the combined component styles,

please make use of our [CRG](#) (Custom Resource Generator) in your application.

Create a simple File Manager

Refer the following code to include the file manager in application .

- Create an **Angular** component with file manager. Add the FileManager component by using selector in template section of the **app.component.ts** file.

```
`HTML
```

```
<ejs-filemanager id='overview' [ajaxSettings]='ajaxSettings'>
</ejs-filemanager>
`
```

- Create an **Angular** module and include the above file manager component.
- In the module, declare the Component and Directives required to render the file manager.
- Bootstrap the application with the above module.

Refer to the following snippet to import the **FileManagerAllModule** in **app.module.ts** from the **@syncfusion/ej2-angular-filemanager**.

```
`Typescript
```

```
import { AppComponent } from './app.component';
import { HttpClientModule, JsonpModule } from '@angular/http';
import { BrowserModule } from '@angular/platform-browser';
import 'rxjs/add/operator/map';
import { NgModule } from '@angular/core';
```

```
import { FileManagerAllModule } from '@syncfusion/ej2-angular-filemanager';
@NgModule({
imports: [FileManagerAllModule, HttpClientModule, JsonpModule, BrowserModule],
declarations: [AppComponent],
bootstrap: [AppComponent]
})
export class AppModule { }
`
```

systemjs.config.js file should be configured as described in the [Installation and configuration](#) section.

Run the application

Use the npm run start command to run the application in the browser.

```
`sh
npm start
`
```

The following samples shows the file manager component in browser.

APP.COMPONENT.TS

```
import { BrowserModule } from '@angular/platform-browser'
import { NgModule } from '@angular/core'
import { FileManagerModule } from '@syncfusion/ej2-angular-filemanager'
import { Component } from '@angular/core';
@Component({
imports: [FileManagerModule, ],
standalone: true,
selector: 'app-root',
styleUrls: ['./app.component.css'],
template: `<ejs-filemanager id='default-filemanager' #filemanagerObj
[ajaxSettings]='ajaxSettings' [view]='view'>
</ejs-filemanager>`
})
export class AppComponent {
public view?: any;
public ajaxSettings?: object;
public hostUrl: string = 'https://ej2-aspcore-service.azurewebsites.net/';
public ngOnInit(): void {
this.ajaxSettings = {
url: this.hostUrl + 'api/FileManager/FileOperations',
getImageUrl: this.hostUrl + 'api/FileManager/GetImage',
uploadUrl: this.hostUrl + 'api/FileManager/Upload',
downloadUrl: this.hostUrl + 'api/FileManager/Download'
};
}
}
```

MAIN.TS

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

Customize custom thumbnail in Angular File manager component

The default appearance of the file manager can customize with your own icon by using [showThumbnail](#) property.

The following example demonstrate how to add a custom icon in largeicons view.

APP.COMPONENT.TS

```
import { BrowserModule } from '@angular/platform-browser'
import { NgModule } from '@angular/core'
import { FileManagerModule, NavigationPaneService, ToolbarService,
DetailsViewService } from '@syncfusion/ej2-angular-filemanager'
import { Component } from '@angular/core';
@Component({
  imports: [FileManagerModule, ],
  providers:[ NavigationPaneService, ToolbarService, DetailsViewService],
  standalone: true,
  selector: 'app-root',
  styleUrls: ['./app.component.css'],
  template: `<ejs-filemanager id='file-manager'
[ajaxSettings]='ajaxSettings' [showThumbnail]='showThumbnail'>
</ejs-filemanager>`
})
export class AppComponent {
  public ajaxSettings?: object;
  public showThumbnail?: boolean;
  public hostUrl: string = 'https://ej2-aspcore-
service.azurewebsites.net/';
  public ngOnInit(): void {
    this.ajaxSettings = {
      url: this.hostUrl + 'api/FileManager/FileOperations',
      getImageUrl: this.hostUrl + 'api/FileManager/GetImage',
      uploadUrl: this.hostUrl + 'api/FileManager/Upload',
      downloadUrl: this.hostUrl + 'api/FileManager/Download'
    };
    this.showThumbnail = false;
  }
}
```

MAIN.TS

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

Nested items in Angular File manager component

FileManager can be rendered inside the other components like Tab, Dialog, and more.

- [Adding file manager inside the dialog](#)
- [Adding file manager inside the tab](#)

Adding file manager inside the dialog

The following example shows the file manager component rendered inside the dialog. Click the browse button in the Uploader element to open the File Manager inside the Dialog control.

APP.COMPONENT.TS

```
import { BrowserModule } from '@angular/platform-browser'
import { UploaderModule } from '@syncfusion/ej2-angular-inputs'
import { DialogModule } from '@syncfusion/ej2-angular-popups'
import { ButtonModule, CheckBoxModule } from '@syncfusion/ej2-angular-buttons'
import { NgModule } from '@angular/core'
import { FileManagerModule, NavigationPaneService, ToolbarService,
DetailsViewService } from '@syncfusion/ej2-angular-filemanager'
import { Component, ViewChild, Inject } from '@angular/core';
import { UploaderComponent } from '@syncfusion/ej2-angular-inputs';
import { DialogComponent } from '@syncfusion/ej2-angular-popups';
import { FileManagerComponent, FileOpenEventArgs } from '@syncfusion/ej2-angular-filemanager';
import { EmitType } from '@syncfusion/ej2-base';
@Component({
  imports: [FileManagerModule, UploaderModule, DialogModule, ButtonModule,
  CheckBoxModule],
  providers: [NavigationPaneService, ToolbarService, DetailsViewService],
  standalone: true,
  selector: 'app-root',
  styleUrls: ['./app.component.css'],
  template: `<div class="sample-container">
    <div id='uploadFileManager' class="fileupload">
      <ejs-uploader #uploadObj
id='defaultfileupload'></ejs-uploader>
      <button ej-button id='openBtn'
(click)="btnClick()">File Browser</button>
    </div>
    <div id='target' class="control-section">
      <ejs-dialog #dialogObj id='dialog'
[visible]='visible' [header]='dialogHeader'
[animationSettings]='animationSettings' [showCloseIcon]='showCloseIcon'
(open)="dialogOpen()" (close)="dialogClose()"
[target]='target' [width]='dialogWidth'>
      <ejs-filemanager #filemanagerObj id='filemanager'
[ajaxSettings]='ajaxSettings' [toolbarSettings]='toolbarSettings'
[contextMenuSettings]='contextMenuSettings' [allowMultiSelection]='false'
(fileOpen)="onFileOpen($event)">
      </ejs-filemanager>
    </div>
  </div>`
})
export class AppComponent {
  @ViewChild('uploadObj')
  public uploadObj?: UploaderComponent;
  @ViewChild('dialogObj')
```

```

public dialogObj?: DialogComponent;
@ViewChild('filemanagerObj')
public filemanagerObj?: FileManagerComponent;
public dialogHeader = 'Select a file';
public animationSettings: Object = { effect: 'None' };
public showCloseIcon = true;
public target = '#target';
public visible = false;
public dialogWidth = '850px';
public ajaxSettings?: object;
public contextMenuSettings?: object;
public toolbarSettings?: object;
public hostUrl = 'https://ej2-aspcore-service.azurewebsites.net/';
public contextmenuItems: string[] = ['Open', '|', 'Cut', 'Copy',
'Delete', 'Rename', '|', 'Details'];
public btnClick: EmitType<object> = () => {
    this.dialogObj?.show();
    this.dialogOpen();
    (this.filemanagerObj as FileManagerComponent).path = '/';
    (this.filemanagerObj as FileManagerComponent).selectedItems = [];
    (this.filemanagerObj as FileManagerComponent).refresh();
}
// Uploader will be hidden, if Dialog is opened
public dialogOpen: EmitType<object> = () => {
    (document.getElementById('uploadFileManager') as
HTMLElement).style.display = 'none';
}
// Uploader will be shown, if Dialog is closed
public dialogClose: EmitType<object> = () => {
    (document.getElementById('uploadFileManager') as
HTMLElement).style.display = 'block';
}
// File Manager's fileOpen event function
public onFileOpen(args: FileOpenEventArgs): void {
    let file = (args as any).fileDetails;
    if (file.isFile) {
        args.cancel = true;
        if (file.size <= 0 ) { file.size = 10000; }
        (this.uploadObj as UploaderComponent).files = [{name: file.name,
size: file.size, type: file.type}];
        this.dialogObj?.hide();
    }
}
public ngOnInit(): void {
    this.ajaxSettings = {
        url: this.hostUrl + 'api/FileManager/FileOperations',
        getImageUrl: this.hostUrl + 'api/FileManager/GetImage',
        uploadUrl: this.hostUrl + 'api/FileManager/Upload',
        downloadUrl: this.hostUrl + 'api/FileManager/Download'
    };
    this.toolbarSettings = {
        items: ['NewFolder', 'Upload', 'Delete', 'Cut', 'Copy', 'Rename',
'SortBy', 'Refresh', 'Selection', 'View', 'Details']
    };
    this.contextMenuSettings = {
        file: this.contextmenuItems,
        folder: this.contextmenuItems
    };
}

```

```

    };
    (this.uploadObj as UploaderComponent).autoUpload = true;
  }
  public ngOnDestroy(): void {
    if ((document.querySelector('.sb-demo-section') as
    Element).classList.contains('upload-dialog')) {
      (document.querySelector('.sb-demo-section') as
    Element).classList.remove('upload-dialog');
    }
  }
}

```

MAIN.TS

```

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

```

Adding file manager inside the tab

The following example demonstrate that the file manager component is placed inside the content area of tab element.

APP.COMPONENT.TS

```

import { BrowserModule } from '@angular/platform-browser'
import { TabAllModule } from '@syncfusion/ej2-angular-navigations'
import { ButtonModule, CheckBoxModule } from '@syncfusion/ej2-angular-
buttons'
import { NgModule } from '@angular/core'
import { FileManagerModule, NavigationPaneService, ToolbarService,
DetailsViewService } from '@syncfusion/ej2-angular-filemanager'
import { Component } from '@angular/core';
import { FileManagerComponent } from '@syncfusion/ej2-angular-filemanager';
import { TabAllModule } from '@syncfusion/ej2-angular-navigations';
@Component({
  imports: [FileManagerModule, TabAllModule , ButtonModule, CheckBoxModule ],
  providers:[ NavigationPaneService, ToolbarService, DetailsViewService],
  standalone: true,
  selector: 'app-root',
  styleUrls: ['./app.component.css'],
  template: ` <div class="control-section e-tab-section">
    <div class="e-sample-resize-container">
      <!-- Render the Tab Component -->
      <ejs-tab id="tab_default" heightAdjustMode='Content'
[showCloseButton]='enableClose' >
        <e-tabitems>
          <e-tabitem [header]='headerText[0] '>
            <ng-template #content>
              <div class="cnt-text"
>Overview</div>

```

The file manager component contains a context menu for performing file operations, large-icons view for displaying the files and folders, and a breadcrumb for navigation. However, these basic functionalities can be extended by using the additional feature

```

modules like toolbar, navigation pane, and details view to simplify the
navigation and file operations within the file system.
        </ng-template>
    </e-tabitem>
    <e-tabitem [header]='headerText[1] '>
        <ng-template #content>
            <div class="cnt-text"
>Filemanager with Default Functionalities</div>
            <ejs-filemanager id='file-
manager' [ajaxSettings]='ajaxSettings'>
                </ejs-filemanager>
            </ng-template>
        </e-tabitem>
    </e-tabitems>
</ejs-tab>
</div>
</div>`
    })
export class AppComponent {
    public headerText: Object | any = [{ text: 'Overview' }, { text:
'FileManager' }];
    public ajaxSettings?: object;
    // Mapping Tab items showCloseButton property
    public enableClose: boolean = true;
    public hostUrl: string = 'https://ej2-aspcore-
service.azurewebsites.net/';
    public ngOnInit(): void {
        this.ajaxSettings = {
            url: this.hostUrl + 'api/FileManager/FileOperations',
            getImageUrl: this.hostUrl + 'api/FileManager/GetImage',
            uploadUrl: this.hostUrl + 'api/FileManager/Upload',
            downloadUrl: this.hostUrl + 'api/FileManager/Download'
        };
    };
}
}

```

MAIN.TS

```

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

```

Create the custom file provider using NodeJS

Here we manipulate the Azure Blob Storage to supply the necessary data for the File Manager. We achieve this by utilizing NodeJS to fetch the required data from the Azure blob storage.

NodeJS acts as the bridge between the File Manager component and Azure Blob Storage, allowing seamless communication and data retrieval. Through this integration, the File Manager can access and interact with the data stored in Azure Blob Storage, enabling smooth file management operations.

Prerequisites

- Valid Azure blob storage account. (accountName, accountKey, endpointSuffix)

- Node version 14 above.

[Introduction to Azure Blob Storage](#)

Azure Blob Storage is a cloud-based object storage service provided by Microsoft Azure. It is designed to store and manage unstructured data, also known as "blobs" in the cloud. Blobs can be any type of data, such as images, videos, documents, backups, logs, and more.

[Key concepts of Azure Blob Storage](#)

Containers: In Azure Blob Storage, data is organized into containers. Containers are logical units that can hold one or more blobs. Think of them as directories or folders that help organize the data.

Blobs: Blobs are the actual data objects stored in Azure Blob Storage.

By understanding the fundamental concepts and use cases of Azure Blob Storage, you will be well-prepared to proceed with setting up and interacting with it using NodeJS in the custom File Provider.

[Create NodeJS project](#)

Following the steps to create the NodeJS project.

Create a new directory for your project and run the following command to initialize a new NodeJS project. This will create a package.json file.

```
`ts
```

```
npm init
```

```
,
```

Install the following packages.

- express
- @azure/storage-blob
- archiver
- body-parser
- cors
- esm
- multer

Open your text editor or integrated development environment (IDE) and create the index.js file start writing your NodeJS code. This file will serve as the entry point of your application.

```
`ts
```

```
const express = require('express');
```

```
const app = express();
```

```
const port = 3000;
```

```
app.get('/', (req, res) => {
```

```
res.send('Hello, NodeJS!');
```

```
});
```

```
app.listen(port, () => {
```

```
console.log(Server running on http://localhost:${port});
});
`ts
```

To start your NodeJS application, simply run the following command in your terminal, pointing to the entry point file:

```
`ts
node index.js
`
```

Initialize container client

We need to first get the BlobServiceClient. By using the connection string, we can obtain the BlobServiceClient. So, format the connection string as shown below.

```
`ts
Const connectionString =
DefaultEndpointsProtocol=https;AccountName=${accountName};AccountKey=${accountKey};E
ndpointSuffix=${EndpointSuffix};
`
```

We can obtain the BlobServiceClient and the **containerClient** using this connection String and the BlobServiceClient. the **containerName** is the container from your Azure blob storage account that you need to access.

```
`ts
import { BlobServiceClient } from "@azure/storage-blob";
const blobServiceClient = BlobServiceClient.fromConnectionString(connectionString);
const containerClient = blobServiceClient.getContainerClient(containerName);
`
```

File actions

Need to provide the following action to creating a new folder, copying and moving of files or folders, deleting, uploading, and downloading the files or folders in the file system

Read

Specify the directory name that needs to be accessed.

```
`ts
const directoryName = 'Files';
`
```

Create the **app.post** method with URL **‘/fileManager’**.

To identify the action by use this condition **req.body.action === ‘read’**

The following table represents the request parameters of **read** operations.

Parameter	Type	Default	Explanation
-----------	------	---------	-------------

----	----	----	----
action	String	read	Name of the file operation.
path	String	-	Relative path from which the data has to be read.
showHiddenItems	Boolean	-	Defines show or hide the hidden items.
data	FileManagerDirectoryContent	-	Details about the current path (directory).

Example for request:

```
`ts
{
  action: "read",
  path: "/Videos/",
  showHiddenItems: false,
  data: [
    0:{
      name:"Videos",
      size:0,
      dateModified:"2023-09-14T14:28:27.000Z",
      dateCreated: "2023-09-14T11:16:57.000Z",
      hasChild:true,
      isFile:false,
      type:"Directory",
      filterPath:"/",
      fmicon: "e-fe-folder",
      fmiconClass: "e-fe-folder",
      fmid: "fetree0",
      fmmodified: "September 14, 2023 19:58"
    }
  ]
}
```

The following table represents the response parameters of **read** operations.

Parameter	Type	Default	Explanation
----	----	----	----
cwd	FileManagerDirectoryContent	-	Path (Current Working Directory) details.

|files|FileManagerDirectoryContent[]|-|Details of files and folders present in given path or directory.|

|error|[ErrorDetails](#)|-|Error Details|

The following table represents the contents of **FileManagerDirectoryContent** in the file manager request and response.

Parameter	Type	Default	Explanation	Is required
----	----	----	----	----
name	String	-	File name	Yes
dateCreated	String	-	Date in which file was created (UTC Date string).	Yes
dateModified	String	-	Date in which file was last modified (UTC Date string).	Yes
filterPath	String	-	Relative path to the file or folder.	Yes
hasChild	Boolean	-	Defines this folder has any child folder or not.	Yes
isFile	Boolean	-	Say whether the item is file or folder.	Yes
size	Number	-	File size	Yes
type	String	-	File extension	Yes
permission	AccessRules	-	File extension	Optional
caseSensitive	Boolean	-	Defines search is case sensitive or not.	Optional
action	String	read	Name of the file operation.	Optional
names	String[]	-	Name list of the items to be downloaded.	Optional
data	FileManagerDirectoryContent	-	Details of the download item.	Optional
uploadFiles	IList<IFormFile>	-	File that are uploaded.	Optional
newName	String	-	New name for the item.	Optional
searchString	String	-	String to be searched in the directory.	Optional
targetPath	String	-	Relative path where the items to be pasted are located.	Optional
targetData	FileManagerDirectoryContent	-	Details of the copied item.	Optional
renameFiles	String[]	-	Details of the renamed item.	Optional

The following table represents the **AccessRules** properties available for file and folder:

Properties	Applicable for file	Applicable for folder	Description
---	---	---	---
Copy	Yes	Yes	Allows access to copy a file or folder.
Read	Yes	Yes	Allows access to read a file or folder.
Write	Yes	Yes	Allows permission to write a file or folder.

WriteContents	No	Yes	Allows permission to write the content of folder.
Download	Yes	Yes	Allows permission to download a file or folder.
Upload	No	Yes	Allows permission to upload to the folder.
Path	Yes	Yes	Specifies the path to apply the rules, which are defined.
Role	Yes	Yes	Specifies the role to which the rule is applied.
IsFile	Yes	Yes	Specifies whether the rule is specified for folder or file.

Example for response:

```
`ts
{
  cwd:
  {
    filterPath: "/",
    hasChild: true,
    name: "Videos",
    size: 0,
    type: "File Folder"
  },
  files:[
    0:{
      dateCreated: "2023-09-14T11:16:57.000Z"
      dateModified: "2023-09-14T11:16:57.000Z"
      filterPath: "/Videos/"
      hasChild: false
      isFile: true
      name: "about.txt"
      size: 29
      type: ".txt"
    }
  ],
  error:null
}
```

Get image

Create the **app.get** method with URL **‘/fileManager/GetImage’**.

The following table represents the request parameters of **GetImage** operations.

Parameter	Type	Default	Explanation
path	String	-	Relative path to the image file

The req.query.path contains the exact path of the images. For example: `"/Jack.png"`.

Download the blob (image) from Azure Blob Storage using the blobClient and stores the result in the downloadResponse variable.

Pipe the readableStreamBody from the blob to the res response. It means the image data will be streamed from the Azure Blob Storage directly to the client's browser when the image URL is accessed.

Handle the exception if the image is not available in the given path.

Download

Create the **app.post** method with URL **‘/fileManager/Download’**.

The following table represents the request parameters of *download* operations.

Parameter	Type	Default	Explanation
action	String	download	Name of the file operation
path	String	-	Relative path to location where the files to download are present.
names	String[]	-	Name list of the items to be downloaded.
data	FileManagerDirectoryContent	-	Details of the download item.

Example for request:

```
`ts
{
  action: 'download',
  path: '/Downloads/Testing/',
  names: [ 'About.txt' ],
  data: [
    0:{
      name: 'About.txt',
      type: '.txt',
      isFile: true,
      size: 29,
      dateModified: '2023-09-14T06:03:52.000Z',
```

```
hasChild: false,
filterPath: '/Downloads/Testing/',
fmcreated: null,
fmmodified: 'September 14, 2023 11:33',
fmiconClass: 'e-fe-txt',
fmicon: 'e-fe-txt'
}
]
}
,
```

The **req.body.downloadInput** must be parsed to get the **downloadObj**. Download the blob from Azure Blob Storage using the blobClient.

Download the blob from Azure Blob Storage using the blobClient and Pipe the readableStreamBody to the response object.

Create the archive file to download the multiple Files, Folders and single folders, then pipe the archive to the response.

Upload

Create the **app.post** method with URL **'/fileManager/Upload'**.

The following table represents the request parameters of *Upload* operations.

Parameter	Type	Default	Explanation
----	----	----	----
action	String	Save	Name of the file operation.
path	String	-	Relative path to the location where the file has to be uploaded.
uploadFiles	IList<IFormFile>	-	File that are uploaded.

Example for request:

```
`ts
{
path: '/Pictures/',
action: 'save',
data: [
0:{
name: 'Pictures',
type: 'File Folder',
isFile: true,
```

```
size: 0,
dateModified: '2023-09-14T06:03:52.000Z',
hasChild: true,
filterPath: "",
fmid: 'fetree1',
},
filename: 'bird (2).jpg'
},
,
```

Multer is a popular middleware used to handle file uploads in Express-based web applications. Create the Multer config to store the upload files in buffer.

```
`ts
const multerConfig = {
storage: memoryStorage()
};
,
```

Need to handle the 3 cases here.

- Save
- Keep Both (action name will be **keepboth**)
- Replace (action name will be **replace**)

Create the **getBlockBlobClient** with the **req.body.filename**. If the blob does not exist, then upload the data to that blob. If the blob already exists, then create an error message containing "File Already Exists" and send the response.

[Create a new folder](#)

The following table represents the request parameters of *create* operations.

Parameter	Type	Default	Explanation
----	----	----	----
action	String	create	Name of the file operation.
path	String	-	Relative path in which the folder has to be created.
name	String	-	Name of the folder to be created.
data	FileManagerDirectoryContent	-	Details about the current path (directory).

Example for request:

```
`ts
```

```
action: "create",
data: [
  0:{
    filterPath: "/",
    hasChild: true,
    isFile: false,
    name: "files",
    nodeId: "fe_tree",
    size: 0,
    type: ""
  }
],
name: "Hello",
path: "/test/"
`
```

Check the existence of the folder, If the folder exists then send the error message containing “Folder already exists”. If it does not exist, then create the folder. Create the folder by creating the file in that folder’s path.

The following table represents the response parameters of *create* operations.

Parameter	Type	Default	Explanation
files	FileManagerDirectoryContent[]	-	Details of the created folder
error	ErrorDetails	-	Error Details

Example for response:

```
`ts
{
  cwd: null,
  files: [
    0:{
      dateCreated: "2023-09-14T10:52:25.000Z",
      dateModified: "2023-09-14T10:52:25.000Z",
      filterPath: null,
      hasChild: false,
      isFile: false,
```

```
name: "New",
size: 0,
type: "Directory"
}
],
details: null,
error: null
}
`
```

Rename

The following table represents the request parameters of *rename* operations.

Parameter	Type	Default	Explanation
----	----	----	----
action	String	rename	Name of the file operation.
path	String	-	Relative path in which the item is located.
name	String	-	Current name of the item to be renamed.
newName	String	-	New name for the item.
data	FileManagerDirectoryContent	-	Details of the item to be renamed.

Example for request:

```
`ts
{
  action: "rename",
  data: [
    0:{
      dateCreated: "2023-09-14T10:41:17.000Z",
      filterPath: "/Pictures/Nature/",
      hasChild: false,
      iconClass: "e-fe-image",
      isFile: true,
      name: "seaviews.jpg",
      size: 95866,
      type: ".jpg"
    }
  ]
}
```

```
],  
  newName: "seaview.jpg",  
  name: "seaviews.jpg",  
  path: "/Pictures/Nature/"  
}  
`
```

Renaming can be done by copy the folder or file from the source blob instance to target blob instance. If the file exists, then send the error message as response.

The following table represents the response parameters of *rename* operations.

Parameter	Type	Default	Explanation
files	FileManagerDirectoryContent[]	-	Details of the renamed item.
error	ErrorDetails	-	Error Details

Example for response:

```
`ts  
{  
  cwd:null,  
  files:[  
    0:{  
      name:"seaview.jpg",  
      size:95866,  
      dateModified:"2023-09-14T11:16:57.000Z",  
      dateCreated:"2023-09-14T10:41:17.000Z",  
      hasChild:false,  
      isFile:true,  
      type:".jpg",  
      filterPath:"/Pictures/Nature/"  
    }  
  ],  
  error:null,  
  details:null  
}  
`
```

Delete

The following table represents the request parameters of *delete* operations.

Parameter	Type	Default	Explanation
----	----	----	----
action	String	delete	Name of the file operation.
path	String	-	Relative path where the items to be deleted are located.
names	String[]	-	List of the items to be deleted.
data	FileManagerDirectoryContent	-	Details of the item to be deleted.

Example for request:

```
`ts
{
  action: "delete",
  path: "/",
  names: ["bird.jpg"],
  data: [
    0:{
      dateModified: "2023-09-14T09:12:53.000Z",
      filterPath: "/",
      hasChild: false,
      iconClass: "e-fe-image",
      isFile: true,
      name: "bird.jpg",
      size: 102182,
      type: ".jpg"
    }
  ]
}
```

To delete the file, directly get the file instance and delete the file. To delete the folder, we need to get all files inside that folder and delete all those files.

Handle the null exception if file or folder is not available.

The following table represents the response parameters of *delete* operations.

Parameter	Type	Default	Explanation
----	----	----	----

|files|FileManagerDirectoryContent[]|-|Details about the deleted item(s).|
|error|[ErrorDetails](#)|-|Error Details|

Example for response:

```
`ts
{
  cwd: null,
  details: null,
  error: null,
  files: [
    0:{
      dateModified: "2023-09-14T09:12:53.000Z",
      filterPath: "/",
      hasChild: false,
      iconClass: "e-fe-image",
      isFile: true,
      name: "bird.jpg",
      size: 102182,
      type: ".jpg"
    }
  ]
}
```

Details

The following table represents the request parameters of *details* operations.

Parameter	Type	Default	Explanation
----	----	----	----
action	String	details	Name of the file operation.
path	String	-	Relative path where the items are located.
names	String[]	-	List of the items to get details.
data	FileManagerDirectoryContent	-	Details of the selected item.

Example:

```
`ts
{
```

```
action: "details",
path: "/FileContents/",
names: ["bird.jpg"],
data: [
0:{
dateModified: "2023-09-14T09:12:53.000Z",
filterPath: "/",
hasChild: false,
iconClass: "e-fe-image",
isFile: true,
name: "bird.jpg",
size: 102182,
type: ".jpg"
}
]
}
```

To get the file and folder details, iterate the **req.body.names** to get the details of files and folders. If the data is file, then get the file instance and get the properties using the **getProperties** method. If the data is Folder, then get the blobs details under that folder using **listBlobsFlat** method. Get the required properties and send final response. Handled the null exception if the file or folder is not available.

The following table represents the response parameters of *details* operations.

Parameter	Type	Default	Explanation
details	FileManagerDirectoryContent	-	Details of the requested item(s).
error	ErrorDetails	-	Error Details

Example:

```
`ts
{
cwd:null,
files:null,
error:null,
details:
{
```

```
created: "2023-09-15T06:04:12.000Z"
isFile: true
location: "Files/bird.jpg"
modified: "2023-09-15T06:04:12.000Z"
multipleFiles: false
name: "bird.jpg"
size: "100.0 KB"
}
}
、
```

Search

The following table represents the request parameters of *search* operations.

Parameter	Type	Default	Explanation
----	----	----	----
action	String	search	Name of the file operation.
path	String	-	Relative path to the directory where the files should be searched.
showHiddenItems	Boolean	-	Defines show or hide the hidden items.
caseSensitive	Boolean	-	Defines search is case sensitive or not.
searchString	String	-	String to be searched in the directory.
data	FileManagerDirectoryContent	-	Details of the searched item.

Example for request:

```
`ts
{
  action: "search",
  path: "/asia/",
  searchString: "nature",
  showHiddenItems: false,
  caseSensitive: false,
  data: [
    0:{
      filterPath: "/",
      hasChild: true,
      name: "asia",
```

```
size: 0,  
type: "File Folder",  
fmid: "fetree1"  
}  
]  
}  
`
```

Replace the '*' in the **req.body.searchString** and assign the result to new variable. Get all blobs under this directory and check that path contains the search string

The following table represents the response parameters of *search* operations.

Parameter	Type	Default	Explanation
----	----	----	----
cwd	FileManagerDirectoryContent	-	Path (Current Working Directory) details.
files	FileManagerDirectoryContent[]	-	Files and folders in the searched directory that matches the search input.
error	ErrorDetails	-	Error Details

Example for response:

```
`ts  
{  
  cwd:  
  {  
    name:"asia",  
    size:0,  
    dateModified:"2023-09-14T14:28:27.000Z",  
    dateCreated:"2023-09-14T11:16:57.000Z",  
    hasChild:true,  
    isFile:false,  
    type:"File Folder",  
    filterPath:"/"  
  },  
  files:[  
    0: {  
      dateModified: "2023-09-15T06:22:00.000Z",  
      filterPath: "/asia/",
```

```
hasChild: false,
isFile: true,
name: "about.txt",
size: 42,
type: ".txt"
}
],
error:null,
details:null
}
,
```

Copy and move

The following table represents the request parameters of *copy* operations.

Parameter	Type	Default	Explanation
----	----	----	----
action	String	copy	Name of the file operation.
path	String	-	Relative path to the directory where the files should be copied.
names	String[]	-	List of files to be copied.
targetPath	String	-	Relative path where the items to be pasted are located.
data	FileManagerDirectoryContent	-	Details of the copied item.
targetData	FileManagerDirectoryContent	-	Details of the copied item.
renameFiles	String[]	-	Details of the renamed item.

Example for request:

```
`ts
{
  action: "copy",
  path: "/",
  names: ["bird.jpg"],
  renameFiles: [],
  targetPath: "/asia/",
  targetData: {
    filterPath: "/",
    hasChild: true,
```

```
name: "asia",
size: 0,
type: "File Folder",
fmid: "fetree1",
},
data: [
0:{
dateCreated: "2023-09-15T06:04:12.000Z",
dateModified: "2023-09-15T06:04:12.000Z",
filterPath: "/",
hasChild: false,
isFile: true,
name: "bird.jpg",
size: 102182,
type: ".jpg",
fmcreated: "September 15, 2023 11:34",
fmhtmlAttr: {class: "e-large-icon", title: "bird.jpg"},
fmiconClass: "e-fe-image",
fmimageAttr: {alt: "bird.jpg"},
fmimageUrl: "http://localhost:3000/GetImage?path=%2Fbird.jpg&time=1694760243307",
fmmodified: "September 15, 2023 11:34",
}
]
}
```

Action name will be **move** for move action.

The following table represents the response parameters of *copy* operations.

Parameter	Type	Default	Explanation
----	----	----	----
cwd	FileManagerDirectoryContent	-	Path (Current Working Directory) details.
files	FileManagerDirectoryContent[]	-	Details of copied files or folders
error	ErrorDetails	-	Error Details

Example for response:

```
`ts
{
  cwd:null,
  files:[
    0:{
      dateCreated: "2023-09-15T06:55:03.000Z"
      dateModified: "2023-09-15T06:55:03.000Z"
      filterPath: "/asia/"
      hasChild: false
      isFile: true
      name: "bird.jpg"
      previousName: null
      size: 102182
      type: ".jpg"
    }
  ],
  error:null,
  details:null
}
```

Need to handle two cases.

- Directory copy and move.
- File copy and move.

Create the **isRename** variable to store the request is rename or not. If the **isRename** is false then check the existence of the folders, and if folder is existing, then send the error message. If **isRename** is true, then don't check the existence of the folder.

To move or copy the folders you need to get all the blobs from that folder and create the new path for each blob and copy the data from the old path to the new path. To move or copy the files copy the data from the source blob client to target client. If the action is move then delete the old blob.

Note: To get the complete project, refer to this [link](#)

Perform custom sorting in Angular FileManager component

The FileManager component provides a way to customize the default sort action for the LargedIconsView by defining the [sortComparer](#) property and for sorting individual columns in the DetailsView by defining the [sortComparer](#) property in the [columns](#) property.

Note: To achieve natural sorting like Windows Explorer, you can import the `SortComparer` function from the '@syncfusion/ej2-angular-filemanager'. If you want to perform your own custom sorting, you can define your own [Link to the Video](#) function.

The following example demonstrates how to define custom sort comparer function to achieve natural sorting behavior for the LargeIconsView and name column in DetailsView.

APP.COMPONENT.TS

```
import { Component } from '@angular/core';
import { sortComparer } from '@syncfusion/ej2-angular-filemanager';
@Component({
  selector: 'app-root',
  styleUrls: ['./app.component.css'],
  template: `<ejs-filemanager id='overview' [ajaxSettings]='ajaxSettings'
[sortComparer]='sortComparer'
[detailsViewSettings]='detailsViewSettings'></ejs-filemanager>`
})
export class AppComponent {
  public ajaxSettings?: object;
  public sortComparer?: object;
  public detailsViewSettings?: object;
  public hostUrl: string = 'https://ej2-aspcore-
service.azurewebsites.net/';
  public ngOnInit(): void {
    this.ajaxSettings = {
      url: this.hostUrl + 'api/NaturalSorting/FileOperations',
      getImageUrl: this.hostUrl + 'api/NaturalSorting/GetImage',
      uploadUrl: this.hostUrl + 'api/NaturalSorting/Upload',
      downloadUrl: this.hostUrl + 'api/NaturalSorting/Download'
    };
    this.sortComparer = sortComparer;
    this.detailsViewSettings = {
      columns: [
        {field: 'name', headerText: 'File Name', minWidth: 120,
width: 'auto', customAttributes: { class: 'e-fe-grid-name' },template:
'${name}', sortComparer : sortComparer},
        {field: 'size', headerText: 'File Size',minWidth: 50, width:
'110', template: '${size}'},
        { field: '_fm_modified', headerText: 'Date
Modified',minWidth: 50, width: '190'}
      ]
    };
  }
}
```

APP.MODULE.TS

```
import { AppComponent } from './app.component';
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { FileManagerModule, NavigationPaneService, ToolbarService,
DetailsViewService } from '@syncfusion/ej2-angular-filemanager';
@NgModule({
  imports: [FileManagerModule, BrowserModule],
  declarations: [AppComponent],
```



```
providers:[ NavigationPaneService, ToolbarService, DetailsViewService],
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);
```

Floating Action Button

Getting started with Angular Floating action button component

This section explains how to create a simple Floating Action Button and demonstrate the basic usage of the Floating Action Button component in an Angular environment.

To get started quickly with Angular Floating Action Button component, you can check out this video:

Dependencies

The list of dependencies required to use the Floating Action Button component in your application is given as follows:

```
`js
|-- @syncfusion/ej2-angular-buttons
|-- @syncfusion/ej2-angular-base
|-- @syncfusion/ej2-base
|-- @syncfusion/ej2-buttons
\`
```

Setup Angular Environment

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

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

Create an Angular Application

Start a new Angular application using below Angular CLI command.

```
`
ng new my-app
`
```

```
cd my-app
`
```

Installing Syncfusion Buttons package

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

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

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

Ivy Library Distribution package

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

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

```
`bash
```

```
npm install @syncfusion/ej2-angular-buttons --save
```

```
,
```

Angular Compatibility Compiled Package(ngcc)

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

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

```
`bash
```

```
npm install @syncfusion/ej2-angular-buttons@ngcc --save
```

```
,
```

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

```
`bash
```

```
@syncfusion/ej2-angular-buttons:"20.3.0.47-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 Floating Action Button module

Import Floating Action Button module into Angular application(`app.module.ts`) from the package

`@syncfusion/ej2-angular-buttons`.

```
`typescript
```

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

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

```
// Imported Syncfusion Floating Action Button module from buttons package.
```

```
import { FabModule } from '@syncfusion/ej2-angular-buttons';
import { AppComponent } from './app.component';
@NgModule({
  imports: [BrowserModule, FabModule], // Registering EJ2 Floating Action Button Module.
  declarations: [AppComponent],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Adding Syncfusion Floating Action Button Component

Modify the template in `app.component.ts` file to render the Floating Action Button component.

```
`typescript
import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
  template: `<!-- To Render Floating Action Button -->
<button ej-s-fab id='fab'></button>`
})
export class AppComponent { }
```

Adding CSS reference

Add Floating Action Button component's styles as given below in `style.css`.

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

Running the application

Run the application in the browser using the following command:

```
ng serve
```

The below example shows a basic Floating Action Button component,

APP.COMPONENT.TS

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

```
import { BrowserModule } from '@angular/platform-browser'
import { FabModule } from '@syncfusion/ej2-angular-buttons'
import { Component } from '@angular/core';
@Component({
  imports: [

    FabModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<div id="targetElement" style="position:relative;min-height:350px;border:1px solid;">
    </div>
    <!-- To Render Floating Action Button -->
    <button ej2-fab id='fab' content='Add'
target='#targetElement'></button>`
})
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));
```

Click event

The floating action button control triggers the `onclick` event when you click on the floating action button. You can use this event to perform the required action.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FabModule } from '@syncfusion/ej2-angular-buttons'
import { Component } from '@angular/core';
@Component({
  imports: [

    FabModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<div id="targetElement" style="position:relative;min-height:350px;border:1px solid;">
    </div>
    <!-- To Render Floating Action Button. -->
    <button ej2-fab id='fab' iconCss= 'e-icons e-edit' content=
'Edit' (click) ="onclick()" target= '#targetElement'></button>`
})
export class AppComponent {
  onclick() {
    alert("Edit is clicked!");
  };
}
```

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 Floating action button component

You can customize the icon and text of Angular Floating Action Button(FAB) using [iconCss](#) and [content](#) properties.

FAB with icon

You can show icon only in Floating Action Button by setting [iconCss](#) property. You can show tooltip on hover to show additional details to end-user by setting [title](#) attribute.

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

@Component({
  selector: 'app-root',
  template: `<!-- To Render Floating Action Button with icon only -->
<button ejs-fab id='fab' iconCss= 'fab-icons fab-icon-people'></button>`
})
export class AppComponent { }
`
```

FAB with icon and text

You can show icon along with text in Floating Action Button by setting [iconCss](#) and [content](#) properties.

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

@Component({
  selector: 'app-root',
  template: `<!-- To Render Floating Action Button with icon and text -->
<button ejs-fab id='fab' iconCss= 'fab-icons fab-icon-people' content= 'Contacts'></button>`
})
export class AppComponent { }
`
```

Icon position

You can change the position of icon when showing along with content by setting [iconPosition](#) property. By default, the icon is positioned on the left side together with text.

```
`typescript
```

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

@Component({
  selector: 'app-root',
  template: `<!-- To Render Floating Action Button with icon position -->
<button ejs-fab id='fab' iconCss= 'fab-icons fab-icon-people' content= 'Contacts' iconPosition=
'Right'></button>`
})
export class AppComponent { }
`
```

Below example demonstrates a FAB with icon and text.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FabModule } from '@syncfusion/ej2-angular-buttons'
import { Component } from '@angular/core';
@Component({
  imports: [
    FabModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<div id="targetElement" style="position:relative;min-
height:350px;border:1px solid;">
</div>
<!-- To Render Floating Action Button -->
<button ejs-fab id='fab' iconCss= 'fab-icons fab-icon-people'
content= 'Contacts' iconPosition= 'Right' target= '#targetElement'></button>`
})
export class AppComponent { }
```

MAIN.TS

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

INDEX.CSS

```
/* Represents the styles for loader */
#loader {
  color: #008cff;
  height: 40px;
  left: 45%;
  position: absolute;
  top: 45%;
  width: 30%;
```

```

}
/* Represents the styles for fab icon */
@font-face {
  font-family: 'fab-icons';
  src: url(data:application/x-font-ttf;charset=utf-
8;base64,AAEAAAAKAIAAAwAgTlMvMj0kSSoAAAEoAAAAVmNtYXCcV5yuAAABlAAAAFRnbHlmHl6s
lgAAAFQAAASQaGVhZCG5vSMAAADQAAANmhoZWEHowNkAAAArAAAACRobXR4E6AAAAAAAYAAAAUUb
G9jYQ GKAYwAAAHoAAAADG1heHABEGDDAAABCAAAACBuYW1l0KnKeQAABoQAAAI9cG9zdBh6gIAAAA
jEAAAArWABAAADUv9qAFoEAAAA//QD9AABAAAAAAAAAAAAAAAAAABQABAAAAAQAAZG1HNV8PPPU
ACwPoAAAAAN9TvCUAAAAA3108JQAAAAAD9AP0AAAACAACAAAAAAAAAAAAEAAAFALcAAwAAAAAAAgAA
AAoACgAAAP8AAAAAAAAAAQPtAZAABQAAAnoCvAAAAIwCegK8AAAB4AAxAQIAAAIABQMAAAAAAAAA
AAAAAAAAAAAAAAAAAUGZFZABA5wTnDANS/2oAWgP0AJYAAAABAAAAAAAAABAAAAAPoAAAD6AAAA+
gAAAPoAAAAAACAACAAAAAwAAABQAAwABAAAAFAAEAEAAAAAKAAgAAgAC5wTnCOcK5wz//wAA5wTnCOc
K5wz//wAAAAAAAAAAAAEACgAKAAoACgAAAAEAAgADAAQAAAAAABgA5AFyAkgaAQAAAAAD6gPqAAsA
AAEzESEVIREjESElIQHDegGu/lJ6/lIBrgPr/lJ6/lIBrnoAAgAAAAADkWP0AHQAtgAAJRUFjFSElI
zU/HjUjDxUvFSMVHx0DER8PPw8RLw8PDgHRIQF3ihISEhIRERAQDxAODg4NDQwLCwsJCQkHBwYGBA
QDAGJXAGIDBAQFBQYHBWgIEhUWFxoaHB4eHx8eHhwaGhcWFRIICAcHBgUFBAQDAGJXAGIDBAQGBgc
ICakKCgsMDA0NDg8OEBAQEREREhMSdgECBQYICgoMDQ8PEBEREhMTEhEREa8PDQwKCgQHBQQAQIF
BggJCgWNDhAQERETExMTEhEQDw8NDAsJBwYfArhbUVFbAgMDBAUFBgYHCagICQoKCgsLDawMDQ0OD
Q4PDg8PDxANDAsMCwwLCgsKCgkSEQ8NDa0HBgQBAQQGBwMDQ8REgkKCgsKCwwLDAsMDRAPDw8ODw
4NDg0NDawMCwsKCgoJCAgIBwYGBQUEAwMCpP64EA8ODg0NCwsJCQcHBAQCAQECEBAUGCAkJCwwMBw0
ODg8BUBAPDg4NDQsLCQkHBgUEAgEBAGQFBgcJCQsLDQ0ODg8AAAAAAwAAAAADxgPoABAAIqBmAAAB
HgIUDgIiLgIPgIyAR4CFA4CIi4CND4CMicOAhUUFhcOAQcuASMiDgIVFbYXDgMVMzQ+AjIeAhUzN
C4CJz4BNTQ+AjIeAhUzNC4CJz4BNTQuAiIBYBkgFRUkMTcwJBuVJDA3AakYJBuVJDE3MCQVFSQwN2
kkNiArJic9FBxWLyLjNiArJiI2JxVdIDZJUkk2IEMVJzYiJisgNklSSTYgQxUnNiImKyA2SVICCws
kMTcwJBuVJDA3MSQVAYYLJDE3MCQVFSQwNzEkFTMQNkkpL1YcFD0nJisgNkkpL1YcETM+RyYpSTYg
IDZJKSZHPjMRHFYvKuk2ICA2SSkmRz4zERxVMclJNiAAAAADAAAAAAP0A/QAPwB/ALUAACUfDz8PL
w8PDgUfDz8PLw8PDgMzEw8CFR8OITUhLwQ3IT8GEz8CNS8GIScjAsgBAQIEBAUFBwcICakJCgoKCg
oKCQkICAcHBQUEBAIBAQBAGQEBQUHBWgICQkKCgoKCgoJCQgIBwYGBQQEAgH+CwEBAGQEBQUHBWg
ICQkKCgoKCgoJCQgIBwYGBQQEAgEBAGQECBAQFBQcHCAGJCQoKCgoKCgkJCAGHBgYFBAQCAclktUgI
AQICBAQFBQcHCAGJCQkKCwJb/bsDAwIBASwBcQ8NDawKCAi8AwQCAgMFBWgJCv0VK6ZwCgoKCQkIC
AcGBgUEBAIBAQBAGQEBQUHBWgICQkKCgoKCgoJCQgIBwYGBQQEAgEBAGQECBAQFBQcHCAGJCQoKCg
oKCgkJCAGHBgYFBAQCAQEBAQIEBAUFBwcICakJCgoKCgoKCQkICAcHBQUEBAIBAQBAGQEBQUHBWg
ICQkKCgMW/on3JgWKCgoJCQgIBwYGBQQEAgEBZAEBAwIjVAECBQUHCQoBUAMHBRAKQgHBQMCZAAA
AAAAEGDeAAEAAAAAAAAAAAAEAAAAAAAAEACQABAAEAAAAAAAAIABwAKAAEAAAAAAAAAMACQARAAEAA
AAAAAQCAaAAEAAAAAAAAUACwAjAAEAAAAAAAAAYACQAUAAEAAAAAAAAoALAA3AAEAAAAAAAAsAEgBjAA
MAAQQAIAAAAGB1AAMAAQQJAEEAgB3AAMAAQQJAAIADgCJAAMAAQQJAAMAEgCXAAMAAQQJAAQAEgC
pAAMAAQQJAUAUFgC7AAMAAQQJAAYAEgDRAAMAAQQJAAoAWADjAAMAAQQJAAsAJAE7IEZhYi1JY29u
c1JlZ3VsYXJGYWItSWNVbnNGYWItSWNVbnNWZXJzaW9uIDEuMEZlZ3VsYXJ29uc0ZvbWVzZ2VzZXJhd
GVkIHVzaW5nIFN5bmNmdXNpb24gTWV0cm8gU3R1ZG1vd3d3LnN5bmNmdXNpb24uY29tACAARgBhAG
IALQBjAGMAbwBuAHMAUGB1AGcAdQBsaGEAcgBGAGEAYgAtAEkAYwBvAG4AcwBGAGEAYgAtAEkAYwB
vAG4AcwBWAGUAcgBzAGkAbwBuACAAMQAuADAARgBhAGIALQBjAGMAbwBuAHMARgBvAG4AdAAgAGcA
ZQBuAGUAcgBhAHQAZQBkACAAADQBzAGkAbgBnACAAUwB5AG4AYwBmAHUAcwBpAG8AbgAgAE0AZQB0A
HIAbwAgAFMAdAB1AGQAAQBVaHcAdwB3AC4AcwB5AG4AYwBmAHUAcwBpAG8AbgAuAGMAbwBtAAAAAA
IAAAAAAAAAACgAAAAAAAAAAAAAAAAAAAAAAAAAAAAABQECAQMBAEFAQYAA2FkZANTaWMGcGVvcGx
lCHNob3BwaW5nAAAA) format('true-type');
  font-weight: normal;
  font-style: normal;
}
[class^="fab-icon-"],
[class*=" fab-icon-"] {
  font-family: 'fab-icons' !important;
  speak: none;
  font-size: 55px;
  font-style: normal;
  font-weight: normal;
  font-variant: normal;
}

```

```

text-transform: none;
line-height: inherit;
-webkit-font-smoothing: antialiased;
-moz-osx-font-smoothing: grayscale;
}
.fab-icon-people:before {
  content: "\e70a";
}

```

Styles in Angular Floating action button component

This section explains the different styles of Floating Action Button.

FAB styles

The Angular Floating Action Button supports the following predefined styles that can be defined using the [cssClass](#) property. You can customize by replacing the `cssClass` property with the below defined class.

cssClass	Description
-----	-----
e-primary	Used to represent a primary action.
e-outline	Used to represent an appearance of button with outline.
e-info	Used to represent an informative action.
e-success	Used to represent a positive action.
e-warning	Used to represent an action with caution.
e-danger	Used to represent a negative action.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FabModule } from '@syncfusion/ej2-angular-buttons'
import { Component } from '@angular/core';
@Component({
  imports: [
    FabModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<div id="targetElement" style="position:relative;min-height:350px;border:1px solid;">
    </div>
    <!-- To Render Floating Action Button with applied warning
style -->
    <button ejs-fab id='fab' iconCss= 'e-icons e-edit' cssClass=
'e-warning' target='#targetElement'></button>`
})
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));
```

Predefined Floating Action Button styles provide only the visual indication. So, Floating Action Button [content](#) property should define the Floating Action Button style for the users of assistive technologies such as screen readers.

Styles customization

To modify the Floating Action Button appearance, you need to override the default CSS of Floating Action Button component. Please find the list of CSS classes and its corresponding section in Floating Action Button component. Also, you have an option to create your own custom theme for the components using our [Theme Studio](#).

| CSS Class | Purpose of Class |

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

|.e-fab.e-btn|To customize the FAB.|

|.e-fab.e-btn:hover|To customize the FAB on hover.|

|.e-fab.e-btn:focus|To customize the FAB on focus.|

|.e-fab.e-btn:active|To customize the FAB on active.|

|.e-fab.e-btn-icon|To customize the style of FAB icon.|

Show text on hover

By using [cssClass](#), you can customize the Floating Action Button to show text on hover with applied transition effect. For detailed information, refer `index.css` file below.

The content will behave the same , when the `enableHtmlSanitizer` is enabled. Since we are adding only the valid tags in content, sanitizing the content will not affect it.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FabModule } from '@syncfusion/ej2-angular-buttons'
import { Component } from '@angular/core';
@Component({
  imports: [
    FabModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<div id="targetElement" style="position:relative;min-height:350px;border:1px solid;">
    </div>
    <!-- To Render Floating Action Button -->
    <button ejs-fab id='fab' iconCss= 'e-icons e-edit'
content='<span class="text-container"><span
class="textEle">Edit</span></span>' cssClass= 'fab-hover'
target='#targetElement'></button>`
```

```
  })  
  export class AppComponent { }
```

MAIN.TS

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

INDEX.CSS

```
/* Represents the styles for loader */  
#loader {  
  color: #008c8f;  
  height: 40px;  
  left: 45%;  
  position: absolute;  
  top: 45%;  
  width: 30%;  
}  
/* start of onhover customization */  
.e-fab.e-btn.fab-hover {  
  padding: 6px 0px 10px 10px;  
}  
.fab-hover .text-container {  
  overflow: hidden;  
  width: 0;  
  margin: 0;  
  transition: width .5s linear 0s, margin .2s linear .5s;  
}  
.fab-hover:hover .text-container {  
  width: 35px;  
  margin: 0 5px;  
  transition: width .5s linear .2s, margin .2s linear 0s;  
}  
/* end of onhover customization */
```

Positions in Angular Floating action button component

The floating action button can be positioned anywhere on the [target](#) using the [position](#) property. If the [target](#) is not defined, then FAB is positioned based on the browser viewport.

The position values of Floating Action Button are as follows:

- TopLeft
- TopCenter
- TopRight
- MiddleLeft
- MiddleCenter
- MiddleRight
- BottomLeft
- BottomCenter

- BottomRight

```
`typescript
```

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

@Component({
  selector: 'app-root',
  template: `<!-- To Render Floating Action Button in BottomLeft position. -->
<button ejs-fab id='fab' content='Add' position='BottomLeft'></button>`
})
export class AppComponent { }
`
```

Below example demonstrates different supported positions of FAB.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FabModule } from '@syncfusion/ej2-angular-buttons'
import { Component } from '@angular/core';
@Component({
  imports: [
    FabModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<div id="target" style="position:relative;min-height:350px;border:1px solid;"></div>
  <!-- To Render Floating Action Button in TopLeft position. -->
  <button ejs-fab id="fab1" iconCss='fab-icons fab-icon-people'
position='TopLeft'
  target='#target'></button>
  <!-- To Render Floating Action Button in TopCenter position. -->
  <button ejs-fab id="fab2" iconCss='fab-icons fab-icon-people'
position='TopCenter'
  target='#target'></button>
  <!-- To Render Floating Action Button in TopRight position. -->
  <button ejs-fab id="fab3" iconCss='fab-icons fab-icon-people'
position='TopRight'
  target='#target'></button>
  <!-- To Render Floating Action Button in MiddleLeft position. -->
  <button ejs-fab id="fab4" iconCss='fab-icons fab-icon-people'position='MiddleLeft'
  target='#target'></button>
  <!-- To Render Floating Action Button in MiddleCenter position. -->
  <button ejs-fab id="fab5" iconCss='fab-icons fab-icon-people'
position='MiddleCenter'
  target='#target'></button>
  <!-- To Render Floating Action Button in MiddleRight position. -->
```

```

    <button ejs-fab id="fab6" iconCss='fab-icons fab-icon-people'
    position='MiddleRight'
        target='#target'></button>
    <!-- To Render Floating Action Button in BottomLeft position. -->
    <button ejs-fab id="fab7" iconCss='fab-icons fab-icon-people'
    position='BottomLeft'
        target='#target'></button>
    <!-- To Render Floating Action Button in BottomCenter position. -->
    <button ejs-fab id="fab8" iconCss='fab-icons fab-icon-people'
    position='BottomCenter'
        target='#target'></button>
    <!-- To Render Floating Action Button in BottomRight position. -->
    <button ejs-fab id="fab9" iconCss='fab-icons fab-icon-people'
    position='BottomRight'
        target='#target'></button>`
  })
  export class AppComponent { }

```

MAIN.TS

```

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

```

INDEX.CSS

```

/* Represents the styles for loader */
#loader {
  color: #008cff;
  height: 40px;
  left: 45%;
  position: absolute;
  top: 45%;
  width: 30%;
}
/* Represents the Styles for fab icon */
@font-face {
  font-family: 'fab-icons';
  src: url(data:application/x-font-ttf;charset=utf-
8;base64,AAEAAAKAIAAAwAgTlMvMjltSgIAAAEoAAAVmNtYXDNyOfNAAABYAAAAFZnbHlmkfZL
RAAAAKgAABBoaGVhZCF5f3EAAADQAAANmhoZWEIUQQTAARAAAACRobXR4SAAAAAAYAAAAABib
G9jYTveNR4AAAIgAAAAJmlheHABIQIXAAABCAAAACBuYl1kwSegQAAGTAAAAIxcG9zdKKfPWYAAB
tkAAAAzQABAAAEAAAAAFwEAAAAAAD9AABAAAAAAAAAAAAAAAAAAAAEgABAAAAQAAbzQqW18PPPU
ACwQAAAAAAN8znUAAAAAA3zOdQAAAAAAD9AP0AAAACAACAAAAAAAAAAAAEAAAASAgSABQAAAAAAAgAA
AAoACgAAAP8AAAAAAAAAAQQAaZAABQAAAokCzAAAAI8CiQLMAAAB6wAyAQgAAAIABQMAAAAAAAAA
AAAAAAAAAAAAAAAAAUGZFZABA5wDnEAQAAAAAXAQAAAAAABAAAAAABAAAAAQAABAAAAEAAAABA
AAAAQAAAAEAAAABAAAAAQAABAAAAEAAAABAAAAAQAABAAAAEAAAABAAAAAQAABAAAAEAAA
AAAAAQAABAAAAAQAABAAAAQAABAAAAQAABAAAAEAAEOCQ//8AAOCa//8AAABAAQAAAABAAIA
AwAEAAUABgAHAAGACQAKAAsADAANAA4ADwAQABEAAAAAAAAABpAPmBFQEggSaBK4FQAXsBrgG1AfcC
BAI5gj8CR4KPA0AAAAABAAAAAAD9APoAD0AewC5AXAAACUfCRUPDiMvDjU/Dh8CAR8LFQ80Iy8PPw
4XAR8LHQEPDi8OPQE/DDMXJw8KHQEFagUvCSsBDw4VHw4zPwcBDwEdAR8NPw41Lw4jDwkBNzUvASU
fCTM/DT0BLw4rAQ8BA5AGBgYFBAQDAgIBAQECAwMFBAUGBgCHBwgHCACIBwCHBgYGBQQEAwICAQEB
AgMDBQQGBQYHBwCKCwoJCv0nBwcGBgYFBAQDAwIBAQECAgQEBQUFBgCHBwCHCACICACHBgYGBQQEA
wMBAQEBAQIDAQEBgUGBwkICQoLatwHBwGBQUFBAMCAGEBAGMBAUFBgYGBwgHBwgIBwCIBwYGBQ
UFBAMCAGEBAGMBAUFBQCQCQkJCQxMDAsKCQgHBgUDAgEBAQP+HwYHCAGICQkKDQ4ODQ4NDawMCwo

```

Copyright © 2001 -2024 Syncfusion Inc.

```

CAGGBgUEAwEBAQMEBgcJCgsNDQ4QEBEREhISEhAQDg4NCwoJBWYFAvgBAQMEBQYGCAGKCgsLDA0QB
gUDAQEBAGICAwMEHjUZGRcXFRUTEhAPDgwKAQMMDAWNDg4dHh8gIQ0MCwoJCQgHBWUFBAMCAQEBAG
MFBgYICQoLCw0NDg4PBgcHCAgJCQkKCgsLCwsLDBIRERAQDg0NCwoJBWYEAJWEXISERAPDg4NDaw
KCgkICQYHBQQFBAUEBAQDAgWXCw0NDg4QERIUFRRYYGhkYFxEUTEhIQDw4NDQsLCwoCBQgEBQQGBQUF
AwQLCAkKCwsMDQ0PDw8RERISFhYVFRMSEQ8ODQsJBWUEAQEEBQcKCg0OEBASFBQVFncTExEREASPD
Q0MDAOKCQkJBQcFBAUFBAUEAwMDDBYMDA0ODw8REhQVFhkaDQ0MDAsKFBMQEA8KDAWMDQ4ODhAPEB
ARERISFRQUFBMSEREQDw8NDAsKCQwLCgoJCQgIBgYFBQMCAgEDBQgJCwwOEBESEXQVFgAAAAEAAAA
AAvAD9AAkAAABERSBES8PIw8OAQ/t9QECawUFBwgICgoLDAWNDQ7WDg0NDawLCgoICAcFBQMCA238
nwEW/uoDYQ4NDQ0LCwsJCQgHBQUdAgEBAGMFBQcICQkLCwsNDQ0AAAAAAWAAAAAD9AP0AD8AfWc1A
AA1Hw8/Dy8PDw4FHW8/Dy8PDw4DMxMPAhUfDiE1IS8ENyE/BhM/AjUvBiEnIwLIAQECBAQFBQcHCA
gJCQoKCgoKCgkJCAGHBWUFBQAQEBAGIEBAUFBWcICAKJCgoKCgoKCQkICAcGBgUEBAIB/gSBAQI
EBAUFBWcICAKJCgoKCgoKCQkICAcHBQUEBAIBAGIEBAUFBWgICQkKCgoKCgoJCQgIBWYGBQQE
AgHJZLVICAECAGQEBQUHBWgICQkJCgsCW/27AwMCAQEsAXEPDQwMCggIvAMEAgIDBQcICQr9FSumc
AoKCgkJCAGHBgYFBAQCAQEBAGIEBAUFBWcICAKJCgoKCgoKCQkICAcHBQUEBAIBAGIEBAUHBW
gICQkKCgoKCgoJCQgIBWYGBQQEAgEBAQECEBAQFBQcHCAgJCQoKCgoKCgkJCAGHBWUFBQAQEBAGI
EBAUFBWcICAKJCgoDFv6DdyYMCgoKCQkICAcGBgUEBAIBAWQBAQMCCVQBAgUFBWkKAVADBwUQCgkI
BwUDAmQAAAAACAAAAAAO2A/QAAgAFAAAJAQsBCQEDRF0+ATcDbPySahL+XQMm/HcCCAHgAAQAAAAA
ygD9AADAACACwAPAAABESMIREjEQEzESMBMxEjAu5e/uBeAUPT0/6D09MDUfyOA3L8jgNy/FMD6P
wYA+gAAWAAAAADuAP0ABEAZgD6AAALDwgvBxMzHwCVMx8SDwIfCCU/CS8CPxEzNT8HBxUPFBuAg8
NFwUfdjsBPw4FNy8MPwI1LyUjDw4CHQEBAGMEBAUFBQUBQDQDAWICKAQECAYGBQQBAjQLFBAQERER
CAGJCAcFBQDQDAgEBAQ0CAGICBAUGBwkL/ZUMCQcGBQQCAGEBAG0BAQECAWQFBQcICQcQEBAQDw4WO
AECBAUGAwciYh8UFBYLCwsLCwoKDQkIBWUFAwMBAQIGBWEDAwMEBAYGBxAPDhIRBQFXAQICBAQEBg
YGBWgHCQgJCQkJCAGIBWcGBgUFBQDAgEBWQUXEW8PDWYGBAQEAgICAQWCAQICBAQGBggJCw0LCws
MCwwLFhUTEwEBAwMFBQYGCACJCQkKCgsYCgoKCQgJBWgGBgUFBACIYwEFBAQEAWIBAQEBAgMEBAQG
A1IBAGUFBWgECUMDCACKDA8RCgsSERERERARERAhIqc4HBIQEAA4ODQ0OAQ8NDQ4OEBASEhUtpyIhE
BEQERARERESChIQDQoJBWg/CQgIBWUDAwIoBQoICw4ICQoLDA0OFxQUFBMTExMSEyUnRm0qGwwKCg
kICAcLCQYHBT4BCQkICAGHBgcFBQUdAwICAGICBAQFBQYGBWcICAGNAT4HCACKDQgICQoKDA0dL58
nJRMSExMTExQUFBQSDQwMCgkICAwJBWYMCgsKCQkJBWcGBgQEAWEBAGQEDAwQGBQcHCAkJCQoKAAAC
AAAAAP0A/QAQAEAAABFQ8PLw8/Dx8OARUPBS8EDWcdAR8DDWYrAQ8FHQEfBjsSBHwUPBB8IPwQfB
h0MBHwU7Aj8FPQE/BR8EPwgvBD8GOWE/BT0BLwYrAS8FPwQvBysBDwMvBj0BLwUrAg8FASgBAwUHCA
oMDQ4QEBISEXQUFBQTEhIQEA4NDAoIBWUDAEQDBQcICgWNDhAQEhITFBQUFBMSEhAQDg0MCggHBQP
+6A4cHA0ODVoDBAQFBUDA1ICAGICAGJbCA4GBgUEBH0GBQQDAWICAQEDAwQEBAV9BAkMBWcIWWMC
AQEBAGIDUGIEBQQFBAQDWg0aDg0ODw8CAGQDBQQFCAyFBAQCAwEOHBWNDg1aAwQEBQQFBAJSAGIBA
QEBAGJbCA4GBgUEBH0GBQQDAWICAQEDAwQEBAV9BAkMBWcIWWMCAQEBAGIDUQMEBAUFBQDWg0aDg
0ODw8CAGMEBAUFCAUEBQMEAGICAAOKFBMSEhAQDg0MCggHBQMBAQMFBWgKDA0OEBASEhMUFBUQEXI
SEBAODQWKCAcFAWEBawUHCAoMDQ4QEBISEXQBx30ECQWHBwhbAgIBAQEBAGJSAGQEBQUEBANXDRoO
DQ4PDWICAwQEBQVwBgUEAwMCAg4cHA0ND1oEBAUEBQFAlICAGEBAQECALsIDgYGBQQEfQYFBAMDA
gICAwMDBQQFfQJDAcHCFsCAGEBAQECak4DBAQFBQQEAlONgG0ODg8PagIDBAQFBHEGBAUDAwICDh
wcDQ0NWwQEBQQFBAUDUGICAGICALsIDgYGBQQEfQYFBAMDAgICAgMDBQAAAAAABIA3gABAAAAAAA
AAAEAAAAABAAAAAABAAgAAQABAAAAAAACAACACQABAAAAAAADAAGAEAAABAAAAAAAEAAgAGAABAAA
AAAFAAAsAIAABAAAAAAGAAgAKwABAAAAAAAKACwAMwABAAAAAALABIAxwADAAEECQAAAAIACQADA
AEECQABABAAcWADAAEECQACAA4AgwADAAEECQADABAAKQADAAEECQAEABAAQADAAEECQAFABYAsQ
ADAAEECQAGABAAxwADAAEECQAKAFgAlwADAAEECQALACQBLyBzYilpY29uc1JlZ3VzYXJzYilpY29
uc3NiLWljB25zVmVyc2lvbiAxLjBzYilpY29uc0ZvbnQgZ2VuZXJhdGVkIHVzaW5nIFN5bmNmdXNp
b24gTWV0cm8gU3RlZGlvd3d3LnN5bmNmdXNpb24uY29tACAAcWBiAC0AaQBJAG8AbgBzAFIAZQBNA
HUAbABhAHIAcWBiAC0AaQBJAG8AbgBzAHMAYgAtAGkAYwBvAG4AcwBWAGUAcgBzAGkAbwBuACAAMQ
AuADAACwBiAC0AaQBJAG8AbgBzAEYAbwBuAHQAIAbNAGUAbgBIAHIAIYQB0AGUAZAAGAHUAcWBPAG4
AZwAgAFMAEQBuAGMAZgB1AHMAAQBVAG4AIABNAGUAdABYAG8AIABTAHQAdQBkAGkAbwB3AHCAAdwAu
AHMAAQBuAGMAZgB1AHMAAQBVAG4ALgBjAG8AbQAAAAACAAAAAaAAAAAaAAAAAaAAAAAaAAAAAa
AAAAAABIBAGEDAQQBQEGAQcBCAEJAQoBCwEMAQ0BDgEPARABEQESARMABXNoYXJlCHNoYXJlLT
AxBGVkaXQHZWRpdC0wMQNhZGQhZXBhcnJvdwVoZWZydANTYXAMdm9pY2Utc2VhcmNoCWZhdmd9yaXR
lcwtjaGF0LXB1cnNvbGhib29rbWFyYXZhaG9wcGluZWwRwBGF5BXBhdXNlCHJlbWluZGVyCHNldHRp
bmdzAAAAAA=) format('trueType');
font-weight: normal;
font-style: normal;
}
[class^="fab-icon-"],

```

```
[class*=" fab-icon-"] {
  font-family: 'fab-icons' !important;
  speak: none;
  font-size: 55px;
  font-style: normal;
  font-weight: normal;
  font-variant: normal;
  text-transform: none;
  line-height: 1;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
}
.fab-icon-people:before {
  content: "\e70a";
}
```

Custom position

You can define the custom position of the Floating Action Button by override the **top**, **left**, **right**, and **bottom** CSS properties using [cssClass](#). For detailed information, refer [index.css](#) file below.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FabModule } from '@syncfusion/ej2-angular-buttons'
import { Component } from '@angular/core';
@Component({
  imports: [

    FabModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<div id="targetElement" style="position:relative;min-height:350px;border:1px solid;">
    </div>
    <!-- To Render Floating Action Button -->
    <button ejs-fab id='fab' iconCss= 'e-icons e-edit' cssClass=
'custom-position' target='#targetElement'></button>`
})
export class AppComponent { }
```

MAIN.TS

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

INDEX.CSS

```
/* Represents the styles for loader */
#loader {
  color: #008cff;
  height: 40px;
```

```
left: 45%;
position: absolute;
top: 45%;
width: 30%;
}
/* Represents the styles for button positioning */
#fab.e-fab.e-btn.custom-position {
  left: 40px;
  top: 40px;
  bottom: unset;
  right: unset;
}
```

Events in Angular Floating action button component

This section explains the available events in Floating Action Button component.

created

Event triggers after the creation of Floating Action Button.

`typescript

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

@Component({
  selector: 'app-root',
  template: `<!-- To Render Floating Action Button. -->
<button ejs-fab id='fab' iconCss= 'e-icons e-edit' content= 'Edit' (created)="onCreate()"></button>`
})

export class AppComponent {
  onCreate() {
    //Your required action here
  };
}
```

onclick

Event triggers when the Floating Action Button is clicked.

`typescript

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

@Component({
  selector: 'app-root',
  template: `<!-- To Render Floating Action Button. -->
<button ejs-fab id='fab' iconCss= 'e-icons e-edit' content= 'Edit' (click)="onclick()"></button>`
})
```



```
export class AppComponent {
  onclick() {
    //Your required action here
  };
}
```

Below example demonstrates the click event of the Floating Action Button.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FabModule } from '@syncfusion/ej2-angular-buttons'
import { Component } from '@angular/core';
@Component({
  imports: [
    FabModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<div id="targetElement" style="position:relative;min-height:350px;border:1px solid;">
    </div>
    <!-- To Render Floating Action Button. -->
    <button ejs-fab id='fab' iconCss= 'e-icons e-edit' content=
'Edit' (click) ="onclick()" target= '#targetElement'></button>`
})
export class AppComponent {
  onclick() {
    alert("Edit is clicked!");
  };
}
```

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 Floating action button component

The Floating action button component followed the accessibility guidelines and standards, including [ADA](#), [Section 508](#), [WCAG 2.2](#) standards, and [WCAG roles](#) that are commonly used to evaluate accessibility.

The accessibility compliance for the Floating action button component is outlined below.

| Accessibility Criteria | Compatibility |

| -- | -- |

| [WCAG 2.2](#) Support | |

| [Section 508](#) Support | |

| Screen Reader Support | |

| Right-To-Left Support | |

| Color Contrast | |

| Mobile Device Support | |

| Keyboard Navigation Support | |

| [Accessibility Checker](#) Validation | |

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

<style>

.post .post-content img {

display: inline-block;

margin: 0.5em 0;

}

</style>

<div> - All features of the component meet the requirement.</div>

<div> - Some features of the component do not meet the requirement.</div>

<div> - The component does not meet the requirement.</div>

WAI-ARIA attributes

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

| Attributes | Purpose |

| --- | --- |

| `aria-label` | Provides an accessible name for the icon only floating action button. |

Keyboard interaction

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

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

| --- | --- |

| **Space** | When the floating action button has focus, pressing the space key changes the state of the floating action button. |

Ensuring accessibility

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

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

See also

- [Accessibility in Syncfusion Angular components](#)