

USER GUIDE

Essential Studio

for EJ2 Angular

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

Syncfusion Angular UI Components (Essential JS 2)	27
Components list	27
table	27
How to best read this user guide	29
Getting help	29
See also	29
System requirements for Angular components.....	30
Angular version	30
Node.js	30
Browser support	30
Angular supported versions	30
See also	30
Browser compatibility in Angular.....	31
Required Polyfills	31
Ways to add Polyfills	31
Polyfill with CDN	31
Polyfill with NPM.....	31
NPM packages for Syncfusion Angular UI Components	32
See Also	32
Installation and Upgrade.....	32
Installation of Syncfusion Controls	32
Install by using NPM CLI.....	32
Install by using package.json.....	33
Download Javascript -EJ2 Installer.....	33
Download the Trial Version	33
Installation using Web Installer	36
Overview	36
Installation	37
Uninstallation.....	48
Installation using offline installer.....	58
Installing with UI	58
Installing in Silent Mode	66
Installing Syncfusion JavaScript – EJ2 Mac Installer.....	67
Steps to resolve the warning message in Catalina OS or later	67
Step-by-Step Installation.....	68

License key registration in samples	72
Linux Installer	72
Download Syncfusion JavaScript Linux Installer	72
Installing Syncfusion JavaScript Linux installer	76
Common Installation Errors	77
Unlocking the License Installer using the Trial Key	77
License has Expired	77
Unable to find a valid license or trial	78
Unable to Install because of Another Installation	79
Unable to Install due to Controlled Folder Access	80
Licensing	82
Syncfusion Licensing Overview	82
Privacy Assurance	82
Difference between unlock key and license key	82
Registering Syncfusion license keys in Build server	82
See Also	83
Generate Syncfusion EJ2-Angular License key	83
Claim License key	83
See Also	85
Register Syncfusion License key in EJ2-Angular application	85
Register Syncfusion License key in the project	86
Register Syncfusion License key using the npx command	86
Activate the license	90
See Also	91
Syncfusion Licensing Errors	91
Licensing errors	91
Licensing errors from version 16.2.0* to 20.3.0*	94
Licensing troubleshoot in Angular	98
Is an internet connection required for license validation	98
Upgrade from the trial version after purchasing a license	98
Where can I get a License key	98
Will the registered license key expire	99
When to generate new license key while upgrading	99
License registration for multiple developers on your project	99
Can I use the same key for all the web apps under the project	99

Does the license registration access any resources or data	99
License & Downloads shows the "Essential Studio Enterprise Edition Binary with Test Studio" and the "Project License". Which license to use.....	100
If I registered the license key in both the application and the license text file	100
Reactivating license once after updating the package version while using npx.....	100
Potential causes of licensing errors in applications.	100
Getting Started.....	102
Getting Started with Angular CLI	102
Prerequisites	102
Setting up an Angular project	102
Create a new application	103
Installing Syncfusion Angular packages	103
Adding Syncfusion Angular components	104
Run the application	105
Syncfusion components-based styles	107
Adding feature Modules to Syncfusion Angular components	108
Syncfusion Angular components showcase samples.....	111
See also	111
Getting Started with Angular Standalone component	111
Create a new application	111
Installing Syncfusion Angular packages	112
Adding Syncfusion Angular components	112
Adding CSS reference.....	114
Run the application	114
Getting Started ASP.NET Core with Angular using Project Template.....	115
Prerequisites	115
Create an application.....	115
Installing Syncfusion Grid Package.....	116
Adding Grid Module.....	116
Adding Syncfusion Component.....	117
Adding CSS reference.....	118
Run the application	118
See also	119
Getting Started with Angular CLI as Front end in ASP.NET MVC	119
Prerequisites	119

Create an ASP.NET MVC Web application	119
Create Angular CLI application.....	122
Configuring ASP.NET MVC application.....	124
Run the Application.....	126
Getting started with Ionic and Angular.....	126
Prerequisites	126
Create an Application.....	127
Installing Syncfusion Grid package.....	127
Adding Grid Module.....	127
Adding Syncfusion component	128
Adding CSS Reference	129
Running the Application.....	129
Getting started with Angular and Electron	130
Prerequisites	130
Setup Angular environment.....	130
Installing Syncfusion Menu package	130
Adding the Menu module	130
Adding Syncfusion Menu component.....	131
Adding CSS reference.....	132
Create main.js file	133
Update index.html	134
Update package.json.....	134
Update tsconfig.json	134
Running the application.....	134
See also	135
Appearance	135
Angular Theme in Syncfusion Components	135
Installation	135
Common Variables.....	137
Suggested link	149
Size Mode for Syncfusion Angular Components.....	149
Size mode for application	150
Change size mode for application at runtime.....	150
Change size mode for a component at runtime	151
See also	152

Icons in Angular Appearance component.....	153
Referring icons in Angular application	153
Steps to use icons library	154
Available Icons	157
Theme studio in Angular Appearance component.....	158
Customizing Theme Color from Theme Studio	158
Import Previously Changed Settings into the Theme Studio	165
Material 3 Theme with CSS Variables	168
Material 3 - Syncfusion Angular Components	169
How To	173
Loading themes without internet access.....	173
Common.....	174
Accessibility in Syncfusion Angular Components	174
Accessibility overview	174
Accessibility standards.....	175
Accessibility compliance	175
Ensuring accessibility	176
Accessibility support for specific components.....	176
table	176
Animation support in Syncfusion Angular Components.....	178
Animation effects.....	178
Animation duration.....	179
Animation delay	180
Enable or disable animation globally	181
Deployment	181
CDN	181
Packages.....	182
Drag and Drop for Angular components.....	182
Draggable	182
Droppable	184
Templates in Syncfusion Angular Components	185
Syncfusion Angular Components - Security	187
Security Vulnerabilities	187
Security Considerations	187
Globalization	191

Globalization	191
Internationalization.....	191
Getting started with Localization	203
Angular Reactive Form Validator	205
Creating Angular Reactive Form with Syncfusion Angular UI Validator	206
Right-To-Left support in Syncfusion Angular Components.....	207
Enable RTL for all components	207
Enable RTL for an individual component	208
Schematics	209
Creating Angular application.....	209
Syncfusion package initialization	209
Adding specific modules from multiple component package	209
Invalid and misspelled module names.....	210
State Persistence in Syncfusion Angular components.....	210
State Persistence supported components and properties	211
How To	214
Update Syncfusion npm package.....	214
How to use SCSS in Angular CLI.....	215
How to use Custom CSS File in Angular Application	216
How to resolve Content Security Policy (CSP) errors.....	217
Troubleshooting.....	217
Compatibility Issues with Syncfusion Angular Packages and Latest Angular CLI.....	217
Content Security Policy	218
Frameworks and Feature	219
Tree Shaking.....	219
Tree Shaking in Angular	219
Using Syncfusion components with Tree Shaking	219
Implementing Tree Shaking in an Angular Application.....	219
Bundle size for Syncfusion Angular Grid component	220
See also	220
Ahead-of-Time (AOT) Compilation in Angular	220
Why use AOT compilation.....	220
Using Syncfusion components with AOT	220
Implementation of AOT Compilation in an Angular Application	221
See also	221

Angular Universal: Server-side Rendering in Angular Frameworks	221
What is Angular Universal.....	221
Why use Server-side Rendering	221
Create an Angular Universal application	222
See also	222
Angular Lazy Loading	223
Lazy Loading.....	223
Creating a Syncfusion component in Angular.....	223
Testing.....	225
Component Testing in Jasmine/Karma Environment for Angular	225
Angular Cypress Testing.....	229
Getting Started for Selenium E2E Testing using Protractor in Angular	233
Upgrade.....	236
Release History	236
Syncfusion Angular supported versions.....	236
Angular version compatibility.....	236
Syncfusion version information	237
See also	237
Upgrading Syncfusion JavaScript (Essential JS2).....	237
Upgrading to the Latest Version	238
Upgrade from Trial Version to License Version	238
Visual Studio Code Integration	238
Visual Studio Code Integration	238
Overview	238
Download and Installation.....	239
Prerequisites	239
Install through the Visual Studio Code Extensions	239
Install from the Visual Studio Code Marketplace	240
Visual Studio Code Extensions	240
Create project	240
Run the application	244
Add Syncfusion Angular component in the Angular application	245
Add a Syncfusion Angular component.....	245
Configure Angular application with Syncfusion	246
Visual Studio Integration.....	248

Visual Studio Integration.....	248
Overview	248
Visual Studio Extensions	249
Create project	249
Convert Project	251
Upgrade Project in Angular Visual studio integration	254
3D Chart	256
Getting started with Angular 3D Chart component.....	256
Setup angular environment	256
Create an Angular application	256
Installing Syncfusion 3D Chart package	256
Registering 3D Chart module	257
Module injection.....	259
Populate chart with data	260
Add 3D Chart title	261
Enable legend.....	262
Add data label.....	263
Enable tooltip.....	265
Working with data in Angular 3D Chart control	266
Local data	266
Remote data.....	267
Binding data using ODataAdaptor	268
Empty points	269
Dimensions in Angular 3D Chart control.....	271
Size for container	271
Size for chart	273
Category axis in Angular 3D Chart control.....	275
Range	276
Indexed category axis.....	277
Numeric axis in Angular 3D Chart control.....	278
Range	279
Range padding	280
Label format.....	286
Grouping separator.....	288
Custom label format	289

DateTime axis in Angular 3D Chart control.....	290
DateTime axis.....	290
DateTime category axis.....	291
Label format.....	298
Custom label format	299
Logarithmic axis in Angular 3D Chart control	300
Range	302
Logarithmic base.....	303
Logarithmic interval	304
Axis labels in Angular 3D Chart control.....	305
Smart axis labels.....	305
Edge label placement.....	309
Maximum labels.....	310
Axis customization in Angular 3D Chart control	311
Title	311
Title rotation	312
Tick lines customization	313
Grid lines customization	314
Multiple axis.....	315
Inversed axis.....	317
Opposed position	318
Multiple panes in Angular 3D Chart control	319
Rows.....	319
Columns	322
Chart Types	325
Column Chart in Angular 3D Chart control	325
Stacked column chart in Angular 3D Chart control.....	331
100% Stacked column chart in Angular 3D Chart control.....	335
Bar Chart in Angular 3D Chart control	339
Stacked bar chart in Angular 3D Chart control	344
100% Stacked bar chart in Angular 3D Chart control	349
Data labels in Angular 3D Chart control	352
Position	353
Template	355
Text mapping	356

Format.....	357
Margin.....	358
Customization	359
Customizing specific label	360
Legend in Angular 3D Chart control.....	362
Position and alignment	362
Legend customization	367
Series selection through legend.....	374
Collapsing legend item.....	375
Legend title	376
Arrow page navigation.....	378
Legend item padding	379
Tooltip in Angular 3D Chart control.....	380
Default tooltip.....	381
Fixed tooltip	382
Format the tooltip.....	383
Tooltip template	384
Customize the appearance of tooltip	385
Selection in Angular 3D Chart control.....	386
Point.....	387
Series.....	388
Cluster	389
Selection type.....	390
Selection during initial loading.....	391
Selection through legend.....	393
Print and Export in Angular 3D Chart control	394
Print.....	394
Export.....	395
Appearance in Angular 3D Chart control.....	396
Custom color palette.....	396
Data point customization.....	398
Point level customization.....	399
Chart area customization.....	400
Animation.....	402
Chart rotation.....	403

Title	404
Accessibility in Angular 3D Chart control.....	410
WAI-ARIA.....	410
Keyboard navigation	410
Accordion	411
Getting started with Angular Accordion component	411
Dependencies.....	411
Setup Angular Environment.....	411
Create an Angular Application	411
Installing Syncfusion Accordion package	412
Registering Accordion Module.....	412
Adding CSS reference.....	413
Add Accordion component	413
Initialize the Accordion using Items	415
Initialize the Accordion using HTML elements.....	418
See Also	420
Expand mode in Angular Accordion component	420
Single.....	420
Multiple.....	422
See Also	423
Accessibility in Angular Accordion component.....	423
ARIA attributes.....	424
Keyboard interaction	425
Ensuring accessibility	425
See also	425
Style in Angular Accordion component	425
Customizing Accordion	425
Customizing the list items.....	426
Customizing Accordion's header.....	426
Customizing Accordion's expand and collapse icons.....	426
Customizing the hover state of Accordion control	426
Customizing selected item of Accordion control.....	426
How To	427
Set the nested accordion in Angular Accordion component.....	427
Load content through post in Angular Accordion component	429

Set custom animation in Angular Accordion component	430
Expand or collapse accordion items on checkbox click in Accordion component.....	433
To keep single pane open always in Angular Accordion component	436
Create wizard using accordion in Angular Accordion component.....	438
Load accordion with data source in Angular Accordion component.....	443
Load accordion items dynamically in Angular Accordion component.....	444
Customize expand collapse actions in Angular Accordion component	446
Render accordion content using angular content in Accordion component.....	448
Integrate treeview inside the accordion in Angular Accordion component.....	450
Ej1 api migration in Angular Accordion component.....	451
Accessibility and Localization.....	451
AjaxSettings.....	452
Animation.....	452
Items	453
Common.....	456
Accumulation Chart	457
Getting started with Angular Accumulation chart component	457
Setup Angular Environment.....	457
Create an Angular Application	458
Installing Syncfusion AccumulationChart package	458
Registering AccumulationChart Module	459
Pie dough nut in Angular Accumulation chart component	461
Pie Chart.....	461
Radius Customization.....	462
Pie Center.....	463
Various Radius Pie Chart	464
Doughnut Chart.....	465
Start and End angles	466
Color & Text Mapping	467
Hide pie or doughnut border	468
Multi-level pie chart.....	469
Pyramid in Angular Accumulation chart component.....	474
Mode.....	474
Size	475
Gap Between the Segments.....	476

Explode.....	477
Customization	478
Funnel in Angular Accumulation chart component	479
Size	480
Neck Size	480
Gap between the segments	481
Explode.....	482
Smart Data Label.....	483
Customization	484
See Also	485
Data label in Angular Accumulation chart component.....	485
Positioning.....	486
DataLabel rotation	487
Smart labels.....	488
Format.....	488
DataLabel template.....	490
Connector Line	491
Text Mapping	492
Customization	492
Text wrap	493
Show percentages in data labels of pie chart	494
Grouping in Angular Accumulation chart component.....	496
Broken slice	497
Group Mode.....	498
Customization	499
Empty points in Angular Accumulation chart component.....	500
Customization	501
Annotation in Angular Accumulation chart component.....	502
Region	503
Co-ordinate Units.....	504
Alignment.....	505
Tooltip in Angular Accumulation chart component.....	506
Header.....	507
Format.....	508
Tooltip template	509

Fixed tooltip	510
Customization	511
Tooltip mapping name	512
To customize individual tooltip.....	513
Legend in Angular Accumulation chart component	514
Position and Alignment.....	515
Legend Reverse	516
Legend Shape	516
Legend Size.....	517
Legend Item Size	518
Enable Animation	519
Paging for Legend.....	521
Legend Text Wrap	521
Legend Title.....	522
Arrow Page Navigation	523
Legend Item Padding	524
Centerlabel.....	525
Hover text	526
Customization	527
Title and sub title in Angular Accumulation chart component.....	528
Title Customization	529
SubTitle	531
SubTitle Customization	532
Chart print in Angular Accumulation chart component	533
Print.....	533
Export.....	534
Appearance in Angular Accumulation chart component	535
Custom Color Palette	535
Animation.....	536
Accessibility in Angular Accumulation chart component	539
WAI-ARIA attributes.....	540
Keyboard interaction	540
Ensuring accessibility	541
See also	541
Ej1 api migration in Angular Accumulation chart component.....	541

Accumulation Chart	541
Annotations.....	543
Series.....	544
DataLabel	549
Legend.....	550
Methods.....	552
Events.....	553
How To	555
Percentage tool tip in Angular Chart component.....	555
Click data in Angular Chart component	556
AppBar	558
Getting started with Angular Appbar component.....	558
Dependencies.....	558
Setup Angular environment.....	558
Create an Angular application	558
Installing Syncfusion AppBar Package.....	559
Adding AppBar module	559
Adding Syncfusion AppBar component	560
Adding CSS reference.....	560
Running the application	561
Size and color in Angular Appbar component	561
Size	561
Color.....	564
Accessibility in Angular AppBar component.....	566
Keyboard interaction	567
Ensuring accessibility	567
See also	568
Position in Angular Appbar component.....	568
Top AppBar	568
Bottom AppBar	569
Sticky AppBar	570
Design in Angular Appbar component.....	571
Spacer.....	571
Separator.....	572
Media Query	573

Designing AppBar with Menu	574
Designing AppBar with Buttons	575
Designing AppBar with SideBar.....	576
Style and appearance in Angular Appbar component.....	578
CssClass	578
HtmlAttributes	579
AutoComplete	580
Getting started with Angular Auto complete component.....	580
Dependencies.....	580
Setup angular environment	580
Create a new application	580
Installing Syncfusion AutoComplete package.....	581
Registering AutoComplete module.....	581
Adding CSS reference.....	582
Adding AutoComplete component	582
Binding data source	583
Running the application	583
Configure the popup list	584
Two-way binding.....	585
See Also	586
Data binding in Angular Auto complete component.....	586
Bind to local data	586
Bind to remote data.....	589
Data binding using Async pipe	590
See Also	592
Value binding in AutoComplete Component.....	592
Primitive Data Types	592
Object Data Types	593
Templates in Angular Auto complete component.....	594
Item template	594
Group template.....	595
Header template	597
Footer template	598
No records template	599
Action failure template	600

See Also	601
Virtualization in AutoComplete Component	601
Binding local data	601
Binding remote data	602
Grouping	604
Grouping in Angular Auto complete component	605
Customization	606
See Also	606
Filtering in Angular Auto complete component	606
Change the filter type	606
Filter item count.....	607
Limit the minimum filter character	609
Case sensitive filtering	610
Diacritics Filtering.....	611
See Also	611
Localization in Angular Auto complete component	612
Loading translations.....	612
See Also	613
Style in Angular Auto complete component.....	613
Customizing the appearance of wrapper element	613
Customizing the dropdown icon's color	613
Customizing the focus color	614
Customizing the outline theme's focus color	614
Customizing the disabled component's text color	614
Customizing the float label element's focusing color	614
Customizing the color of the placeholder text	615
Customizing the text selection color.....	615
Customizing the background color of focus, hover, and active item's	615
Customizing the appearance of pop-up element	616
Adding mandatory asterisk to placeholder and float label.....	616
Accessibility in Angular Auto complete component.....	617
WAI-ARIA attributes.....	618
Keyboard interaction	618
Ensuring accessibility	620
See also	620

Form support in Angular Auto complete component	620
Template-Driven Forms	620
Reactive Forms.....	621
How To	622
Autofill in Angular Auto complete component.....	622
Icon support in Angular Auto complete component	624
Custom search in Angular Auto complete component.....	624
Filter in Angular Auto complete component	626
Suggestion in Angular Auto complete component.....	627
Ej1 api migration in Angular Auto complete component	628
DataBinding.....	629
Filtering	629
Placeholder	630
Popup.....	630
CSS.....	631
Grouping	632
Localization	632
Template	632
Sorting.....	632
Accessibility.....	633
Selection.....	633
Miscellaneous	633
Common.....	634
Avatar	635
Getting started with Angular Avatar component	635
Setting up angular project	635
Dependencies.....	636
Installing Syncfusion Layouts package	636
Adding style sheet to the application	637
Add Avatar into application	637
Run the application	637
Types in Angular Avatar component.....	638
Avatar size	638
Avatar types	639
How To	640

Avatar customization in Angular Avatar component.....	640
Integrate avatar into listview in Angular Avatar component	645
Integrate avatar into badge in Angular Avatar component.....	646
Badge	649
Getting started with Angular Badge component.....	649
Setting up angular project	649
Dependencies.....	650
Installing Syncfusion notifications Package	650
Adding style sheet to the application	651
Add Badge into application.....	651
Run the application	651
Types in Angular Badge component	652
Badge styles	652
Badge types.....	654
How To	661
Badge customization in Angular Badge component.....	661
Integrate badge into listview in Angular Badge component	663
Dynamic badge content in Angular Badge component	665
Barcode	666
Getting started with Angular Barcode component.....	666
Setup Angular Environment.....	666
Create an Angular Application	666
Installing Syncfusion Barcode Generator package.....	667
Adding Syncfusion Barcode Generator package.....	667
Registering Barcode Generator Module	668
Adding Barcode Generator control.....	668
Adding QR Generator control	669
Adding Datamatrix Generator control	669
BarcodeGenerator in Angular Barcode component	670
Code39	670
Code39 Extended	671
Code 11	671
Codabar	672
Code 32	673
Code 93	674

Code 93 Extended	674
Code 128	674
Customizing the Barcode color	675
Customizing the Barcode dimension	676
Customizing the text	677
Qrcodegenerator in Angular Barcode component	677
QR Code	677
Customizing the Barcode color	678
Customizing the Barcode dimension	679
Customizing the text	679
Datamatrixgenerator in Angular Barcode component	680
Data Matrix	680
Customizing the Barcode color	681
Customizing the Barcode dimension	682
Customizing the text	682
Export in Angular Barcode component.....	683
Export.....	683
Breadcrumb	685
Getting started with Angular Breadcrumb component.....	685
Dependencies.....	685
Setup Angular environment.....	686
Create an Angular application	686
Installing Syncfusion Breadcrumb Package.....	686
Adding Breadcrumb module	687
Adding Syncfusion Breadcrumb component	687
Adding CSS reference.....	688
Running the application	688
Add Items to the Breadcrumb Component	688
Enable or Disable Navigation	689
Data binding in Angular Breadcrumb component	690
Items as tag directive	690
Items based on current Url	691
Static URL	692
Customize text when generated items using Url.....	692
Icons in Angular Breadcrumb component	693

Loading icon in BreadcrumbItem	693
Icon Position.....	696
Icon Only	697
Show icon only for first item.....	698
Navigations in Angular Breadcrumb component.....	698
URL	698
Enable navigation for last Breadcrumb item	700
Open URL in new page or tab	701
Templates in Angular Breadcrumb component.....	702
Item Template.....	702
Separator Template	703
Customize Specific Item Template.....	704
Overflow in Angular Breadcrumb component.....	705
Overflow Mode	705
Collapsed.....	706
Menu.....	707
Wrap.....	708
Scroll.....	709
Hidden.....	710
None.....	711
Accessibility in Angular Breadcrumb component.....	711
WAI-ARIA attributes.....	712
Keyboard interaction	712
Ensuring accessibility	712
See also	712
Bullet Chart	713
Getting started with Angular Bullet chart component	713
Setup Angular Environment.....	713
Create an Angular Application	713
Installing Syncfusion BulletChart package	713
Registering Bullet Chart Module.....	714
Module Injection.....	715
Bullet Chart with Data.....	716
Add Bullet Chart Title.....	717
Ranges.....	718

Enable Tooltip	719
Bullet chart dimensions in Angular Bullet chart component.....	720
Size for Container	720
Size for Bullet Chart.....	721
Axis customization in Angular Bullet chart component.....	723
MajorTickLines and MinorTickLines Customization.....	723
Tick Placement	724
Label Format	725
Custom Label Format	727
Label Placement.....	728
Opposed Position	729
Category Label	729
Category Label Customization	730
Data binding in Angular Bullet chart component	731
Local Data.....	731
Ranges in Angular Bullet chart component	733
Color Customization.....	734
Value bar in Angular Bullet chart component	735
Types of actual bar	736
Actual bar customization	737
Comparative bar in Angular Bullet chart component.....	738
Types of target bar	739
Target bar customization	740
Title in Angular Bullet chart component.....	741
Title	741
Subtitle	742
Title and SubTitle Position	743
Title Customization	746
SubTitle Customization	747
Customization in Angular Bullet chart component.....	748
Orientation.....	748
Right-to-left (RTL).....	749
Animation.....	750
Theme	751
Data label in Angular Bullet chart component.....	752

Data Label Customization	753
Tool tip in Angular Bullet chart component.....	755
Default Tooltip	755
Tooltip Template.....	756
Tooltip Customization	757
Accessibility in Angular Bullet chart component	758
WAI-ARIA attributes.....	759
Keyboard interaction	759
Ensuring accessibility	759
See also	759
ButtonGroup	759
Getting started with Angular Button group component	759
Dependencies.....	759
Setup Angular environment.....	759
Create an Angular application	760
Installing Syncfusion ButtonGroup package	760
Adding Button module.....	761
Adding Syncfusion ButtonGroup component.....	761
Adding CSS reference.....	762
Running the application	762
Orientation.....	763
Types and styles in Angular Button group component.....	763
ButtonGroup types	763
ButtonGroup styles	764
See Also	765
Selection in Angular Button group component	765
Selection.....	765
Nesting	767
See Also	769
Accessibility in Angular Button group component	769
Keyboard interaction	770
Ensuring accessibility	771
See also	771
How To	771
Create buttongroup with icons in Angular Button group component	771

Create buttongroup with rounded corner in Button group component.....	772
Disable in Angular Button group component	772
Enable ripple in Angular Button group component.....	773
Enable rtl in Angular Button group component.....	774
Form submit in Angular Button group component	775
Initialize buttongroup using util function in Button group component	776
Show buttongroup selected state on initial render in Button group component.....	778
Button	778
Getting started with Angular Button component.....	778
Dependencies.....	779
Setup Angular environment.....	779
Create an Angular application	779
Installing Syncfusion Button package	779
Adding Button module.....	780
Adding Syncfusion Button component	780
Adding CSS reference.....	781
Running the application.....	781
Change Button type	782
Types and styles in Angular Button component.....	782
Button styles	782
Button types.....	783
Icons.....	786
Button size	788
See Also	789
Accessibility in Angular Button component.....	789
WAI-ARIA attributes.....	790
Keyboard interaction	790
Ensuring accessibility	790
See also	790
How To	790
Add link to a button in Angular Button component	790
Create a block button in Angular Button component	791
Customize button appearance in Angular Button component.....	792
Customize input and anchor elements in Angular Button component	792
Repeat button in Angular Button component	793

Right to left in Angular Button component	795
Set the disabled state in Angular Button component.....	796
Tooltip for button in Angular Button component	796
Ej1 api migration in Angular Button component	797
Properties.....	797
Methods	798
Events.....	799

Syncfusion Angular UI Components (Essential JS 2)

Syncfusion Angular UI (Essential JS 2) is a collection of modern TypeScript based true Angular Components. It has support for Ahead Of Time (AOT) compilation and Tree-Shaking. All the components are developed from the ground up to be lightweight, responsive, modular and touch friendly.

Components list

The Syncfusion Angular UI components are listed below.

<style>

table

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

font-weight: bold!important;

letter-spacing: 0.7px;

}

}

</style>

GRIDS	DATA VISUALIZATION	CALENDARS	DROPDOWNS
DataGrid	Charts	Scheduler	AutoComplete
Pivot Table	3D Chart	Gantt Chart	ListBox
TreeGrid	3D Circular Chart	Calendar	ComboBox
Spreadsheet	Accumulation Chart	DatePicker	Dropdown List
FILE VIEWERS & EDITORS	Stock Chart	DateRangePicker	Multiselect DropDown
Document Editor	Circular Gauge	DateTime Picker	DropDown Tree
In-place Editor	Linear Gauge	TimePicker	Mention
Image Editor	Diagram Component	INPUTS	NAVIGATION
RichTextEditor	HeatMap Chart	TextBox	Accordion
PDF Viewer	Map	TextArea	Carousel
Word Processor	Range Selector	Input Mask	Context Menu
LAYOUT	Smith Chart	Numeric TextBox	Menu Bar
Dialog	Sparkline Charts	RadioButton	Ribbon
ListView	Barcode	CheckBox	Sidebar
Predefined Dialogs	TreeMap	Color Picker	Tabs
Tooltip	Bullet Chart	File Upload	Toolbar
Splitter	Kanban	Range Slider	TreeView
Dashboard	BUTTONS	Toggle Switch Button	File Manager
Card	Button	Signature	Stepper
Avatar	ButtonGroup	Rating	Breadcrumb
Timeline	Dropdown Menu	FORMS	Pager
	Progress Button	Query Builder	AppBar
	SplitButton		NOTIFICATION
	Chips		Toast
	Floating Action Button		Progress Bar

	Speed Dial		Spinner
			Badge
			Skeleton
			Message

How to best read this user guide

- The best way to get started would be to read the "Getting Started" section of the documentation for the component that you would like to start using first.

The "Getting Started" guide gives just enough information that you need to know before starting to write code. This is the only section that we recommend reading end-to-end before starting to write code, all other information can be referred as needed.

- Now that you are familiar with the basics of using the component, the next step would be to start integrating the component into your application.

A good starting point would be to refer to the code snippets in the [online sample browser](#) which contains hundreds of code samples, it is very likely that you will find a code sample that resembles your intended usage scenario.

- Another valuable resource is the API reference which provides detailed information on the object hierarchy as well as the settings available on every object.

Getting help

- If you are still not able to find the information that you are looking for in the self-help resources mentioned above then please contact us by creating a support ticket in [our support site](#) or ask your query in Stack Overflow with tag `syncfusion-ej2`.
- Don't see what you need? Please request it in our [feedback portal](#).

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

See also

- [Product Development Life Cycle](#)
- [Getting started with Syncfusion Angular Components](#)
- [Getting started with Syncfusion Angular SystemJS](#)
- [Getting started Syncfusion Angular with ASP.NET Core](#)
- [Getting started Syncfusion Angular with ASP.NET MVC](#)
- [Getting started with Syncfusion Angular Treeshaking](#)

System requirements for Angular components

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

Angular version

To get started with Syncfusion Angular UI components, make sure the Angular version is 6 or later.

Use the below command to check the Angular CLI version.

```
`bash
ng --version
`
```

Node.js

Angular requires an [active LTS or maintenance LTS](#) version of Node.js.

For more information on installing Node.js, see [nodejs.org](#). To check the version of Node.js running on the system, run `node -v` in a terminal window.

Browser support

The Syncfusion Angular UI components are supported only in modern browsers. Refer to the [browser compatibility](#) section for more information.

Angular supported versions

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

Version	Syncfusion Angular components version
-----	-----
Angular v17	23.2.4 and above
Angular v16	21.1.39 and above
Angular v15	20.4.38 and above
Angular v14	20.2.36 and above
Angular v13	19.4.38 and above
Angular v12	19.3.43 and above
Angular v11	18.4.31 and above
Angular v10	18.2.55 and above
Angular v9	17.4.51 and above
Angular v8	17.1.50 and above
Angular v7	16.3.32 and above

See also

- [Upgrade guide](#)
- [Setting up the local environment and workspace](#)

Browser compatibility in Angular

The Syncfusion Angular UI components are supported only in modern browsers. This includes the following versions.

	Chrome		Firefox		Opera		Edge		IE		Safari		iOS		Android		Windows Mobile	
	-----		-----		-----		-----		-----		-----		-----		-----		-----	
	63+		58+		50+		13+		11+		9+		9+		4.4+		IE 11+	

Required Polyfills

A polyfill is a piece of code (usually JavaScript on the Web) used to provide modern functionality on older browsers that do not natively support it.

The Syncfusion Angular UI components are supported in IE 11 browser with ES6 Promise polyfill.

Ways to add Polyfills

There are two ways to add polyfill in application,

Polyfill with CDN

You can add ES6 Promise polyfill using below CDN in your HTML file.

```
`typescript
<!-- Automatically provides/replaces Promise if missing or broken. -->
<script src="https://cdn.jsdelivr.net/npm/es6-promise@4/dist/es6-promise.js"></script>
<script src="https://cdn.jsdelivr.net/npm/es6-promise@4/dist/es6-promise.auto.js"></script>
<!-- Minified version of es6-promise-auto below. -->
<script src="https://cdn.jsdelivr.net/npm/es6-promise@4/dist/es6-promise.min.js"></script>
<script src="https://cdn.jsdelivr.net/npm/es6-promise@4/dist/es6-promise.auto.min.js"></script>
`
```

Polyfill with NPM

ES6 Promise polyfill can also be installed in npm.

You can use below command to install via npm.

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

You can use like below in your code.

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

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

NPM packages for Syncfusion Angular UI Components

Starting with v18.4.0.30 (Volume 4, 2020), the Syncfusion Angular UI components are separately available in individual [NPM packages](#). The NPM packages are segregated based on the component usage and its namespace.

See Also

- [Installation with NPM CLI](#)
- [Download JavaScript – EJ2 Installer](#)
- [Product Development Life Cycle](#)
- [Update NPM Packages](#)

Installation and Upgrade

<!-- markdownlint-disable MD024 -->

Installation of Syncfusion Controls

Install by using NPM CLI

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

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

By default, Syncfusion Angular packages(>=20.2.36) supports [Angular Ivy distribution](#). The package are compatible with Angular version 12 and above. To install the package use the below command,

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

```
`bash
```

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

Angular compatibility compiled package (ngcc)

For Angular version below 12, you can use the `ngcc` tagged packages of the Syncfusion Angular components. To install the package use the below command,

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

```
`bash
```

```
npm install @syncfusion/ej2-angular-grids@ngcc --save
```


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

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

If the ngcc tag is not specified while installing the package for Angular versions below 12, the Ivy package will be installed with warning.

Install by using package.json

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

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

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

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

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

Download Javascript -EJ2 Installer

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

- Trial Installer
- Licensed Installer

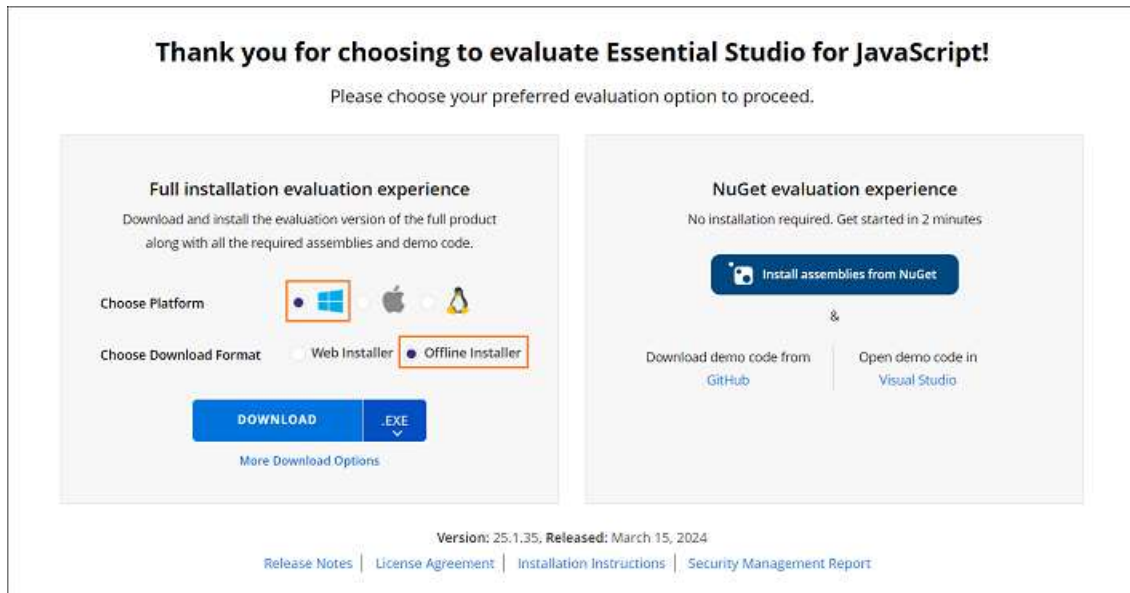
Download the Trial Version

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

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

Download Free Trial Setup

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

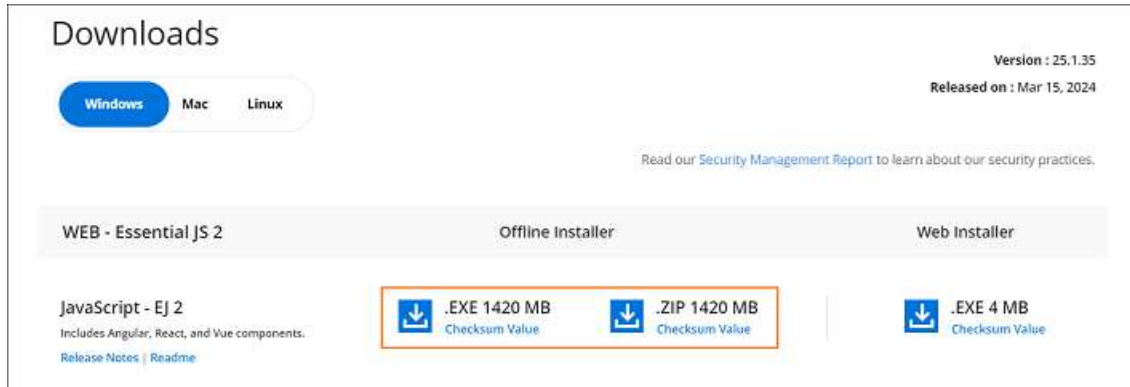


3. With a trial license, only the latest version's trial installer can be downloaded.
4. After downloading, the Syncfusion JavaScript - EJ2 trial installer can be unlocked using either the trial unlock key or the Syncfusion registered login credential. More information on generating an unlock key can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

Trial Downloads and Unlock Keys



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



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

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

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



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

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

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

Trial Downloads and Unlock Keys



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

Download the License Version

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



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

Installation using Web Installer

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

Overview

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

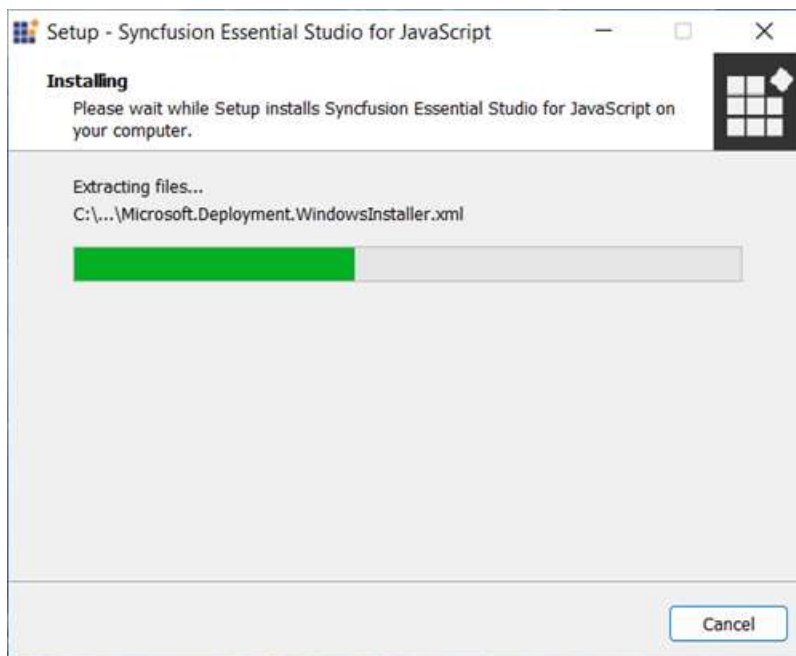
The frameworks listed below are supported in this installer.

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

Installation

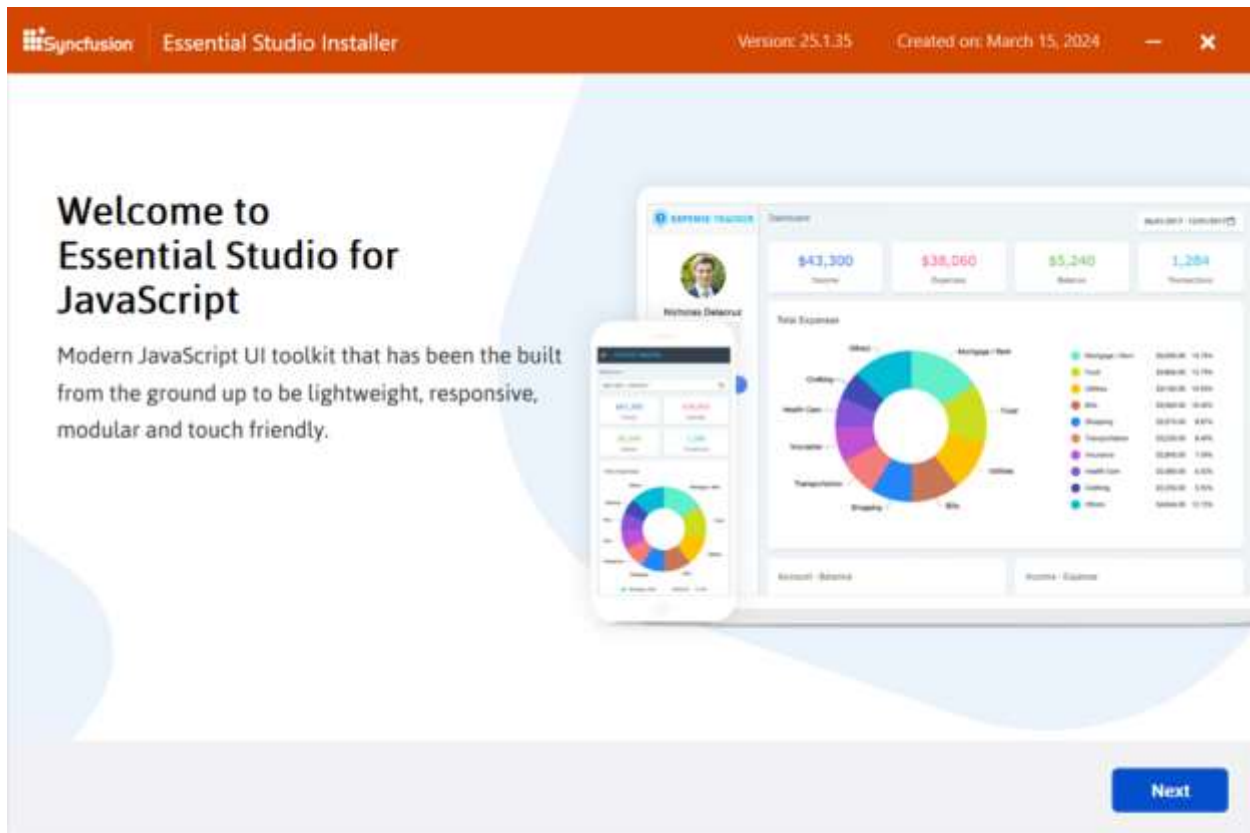
The steps below show how to install Essential Studio JavaScript – EJ2 Web Installer.

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



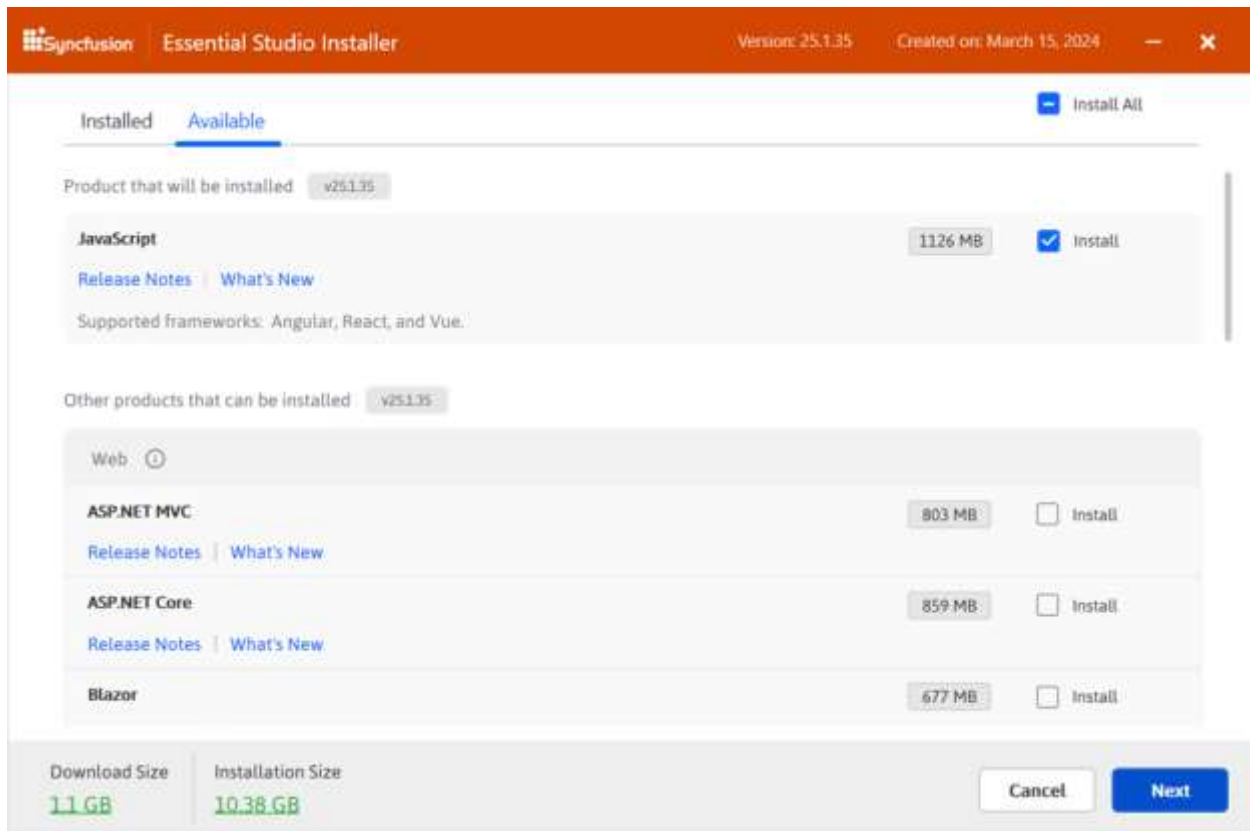
Note: The installer wizard extracts the syncfusionejs2webinstaller_{version}.exe dialog, which displays the package's unzip operation.

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



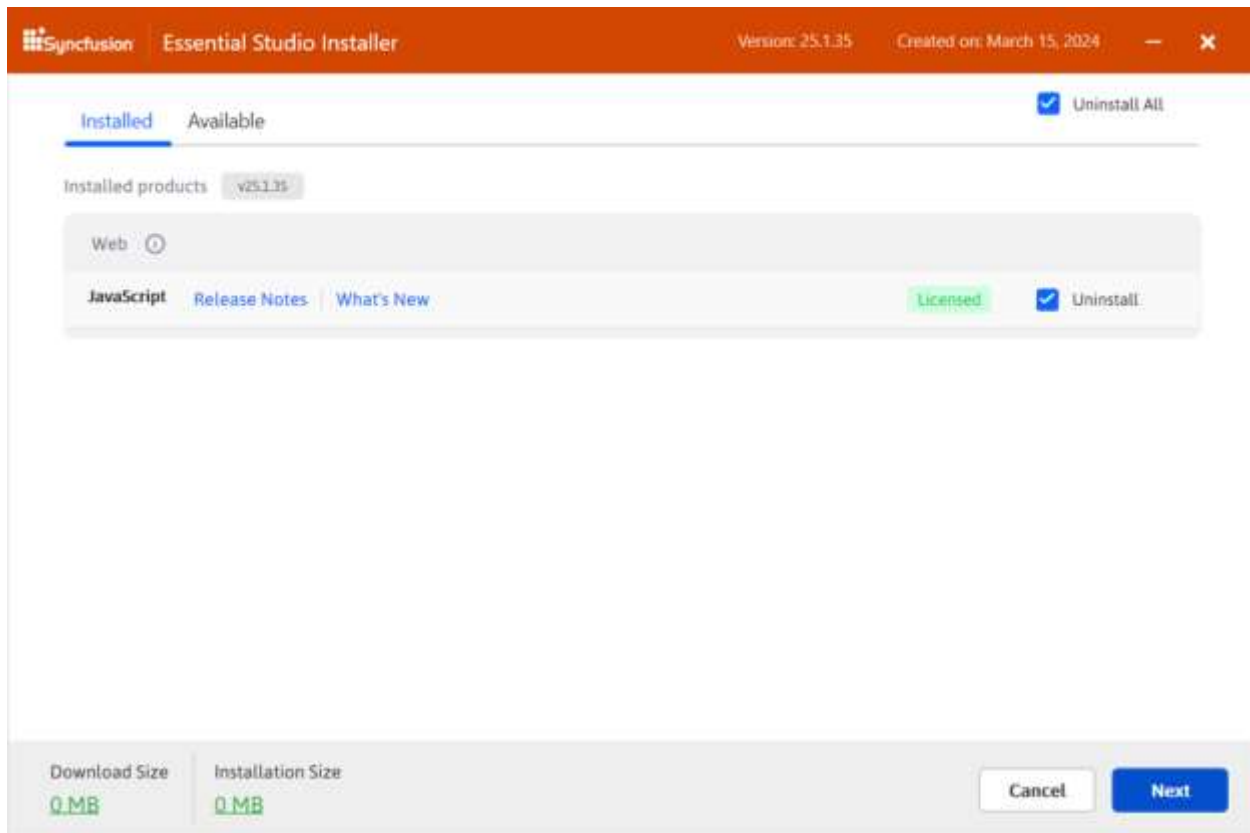
3. The Platform Selection Wizard will appear. From the **Available** tab, select the products to be installed. Select the Install All checkbox to **install all** products.

Available



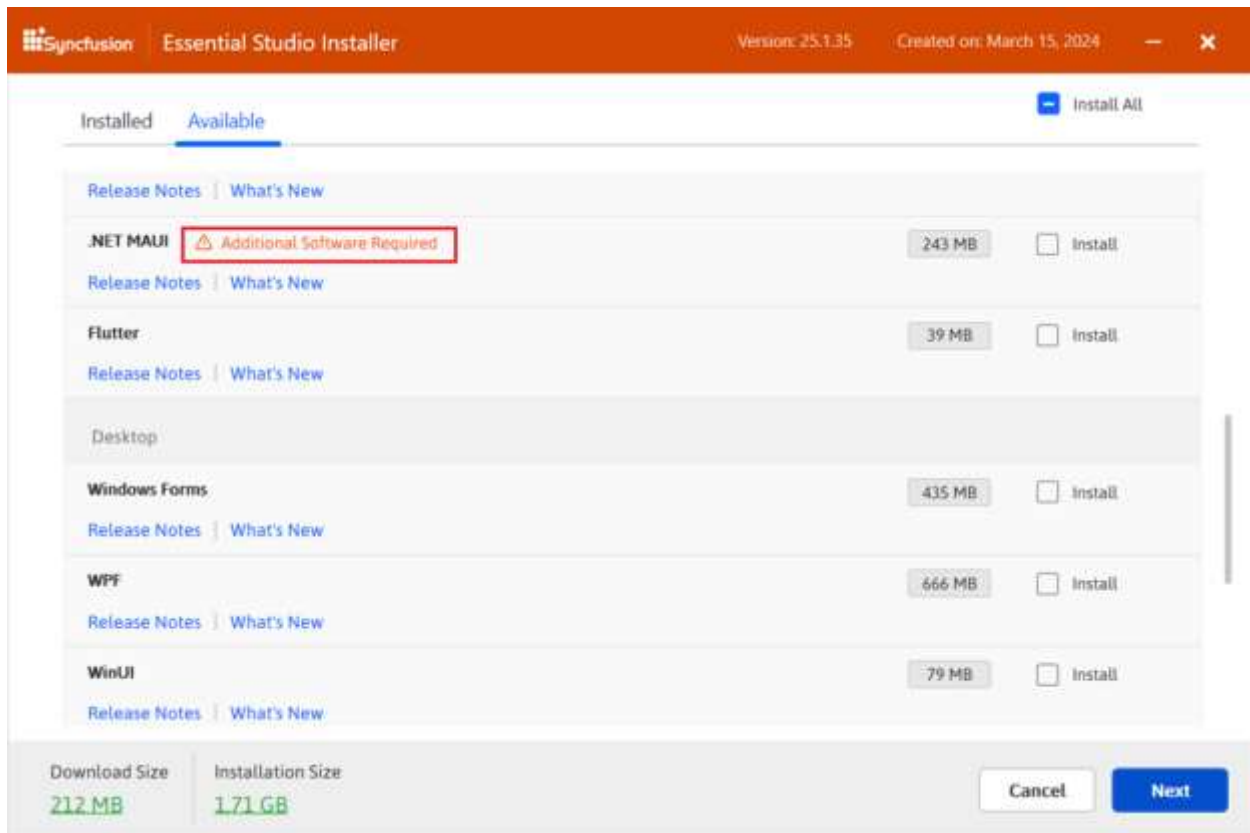
If you have multiple products installed in the same version, they will be listed under the **Installed** tab. You can also select which products to uninstall from the same version. Click the Next button.

Installed

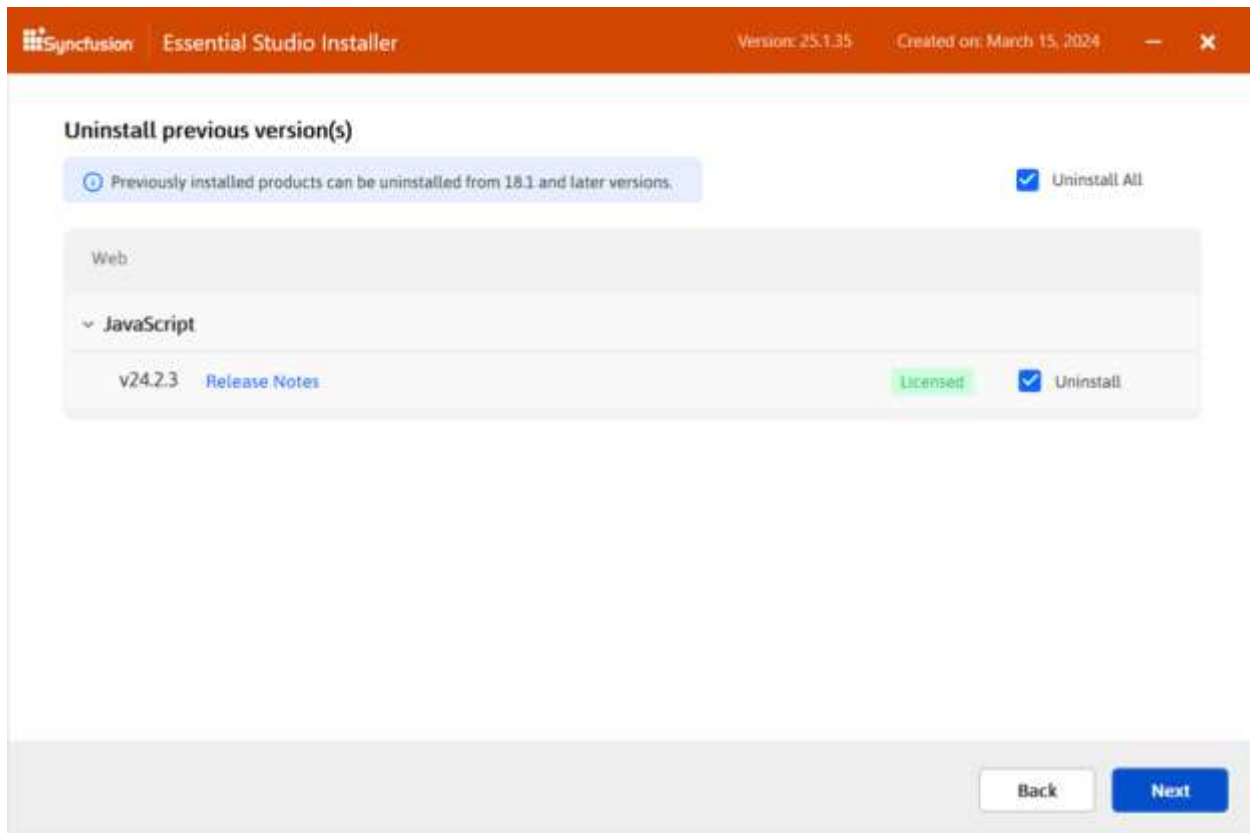


Important: If the required software for the selected product isn't already installed, the **Additional Software Required** alert will appear. You can, however, continue the installation and install the necessary software later.

Required Software

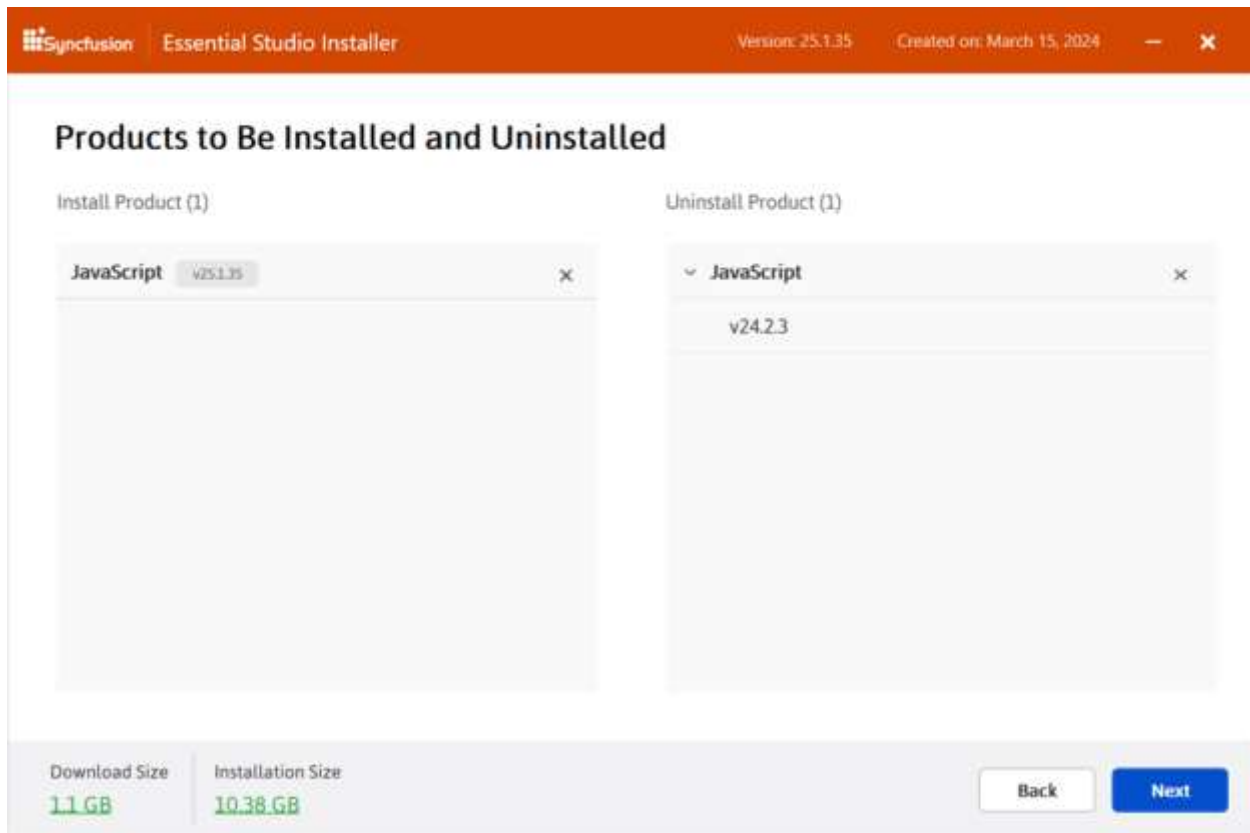


4. If previous version(s) for the selected products are installed, the Uninstall previous version wizard will be displayed. You can see the list of previously installed versions for the products you've chosen here. To remove all versions, check the **Uninstall All** checkbox. Click the Next button.

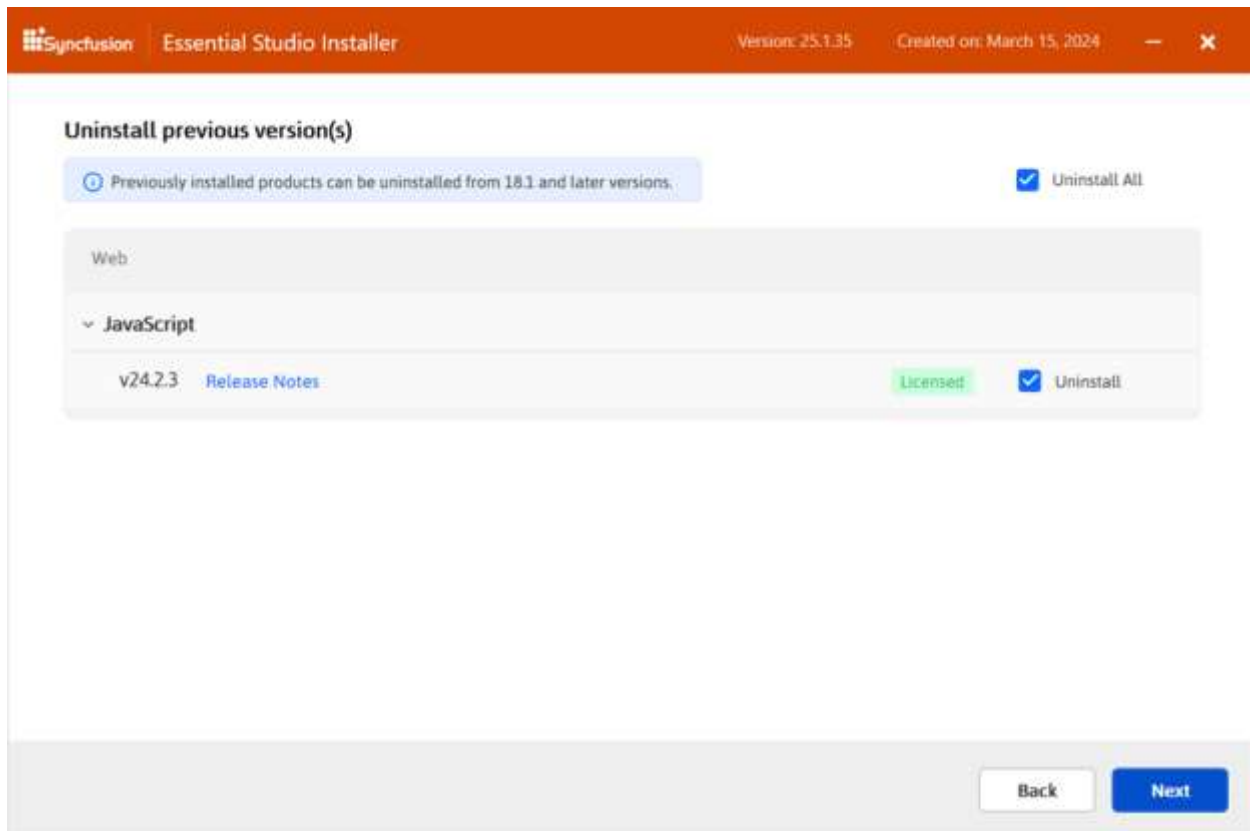


Note: From the 2021 Volume 1 release, Syncfusion has provided option to uninstall the previous versions from 18.1 while installing the new version.

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



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



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

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

Configuration

Download Location
 [Browse](#)

Installation Location
 [Browse](#)

Demos Location
 [Browse](#)

☒ I Agree to the [License Terms](#) and [Privacy Policy](#)

Additional Settings

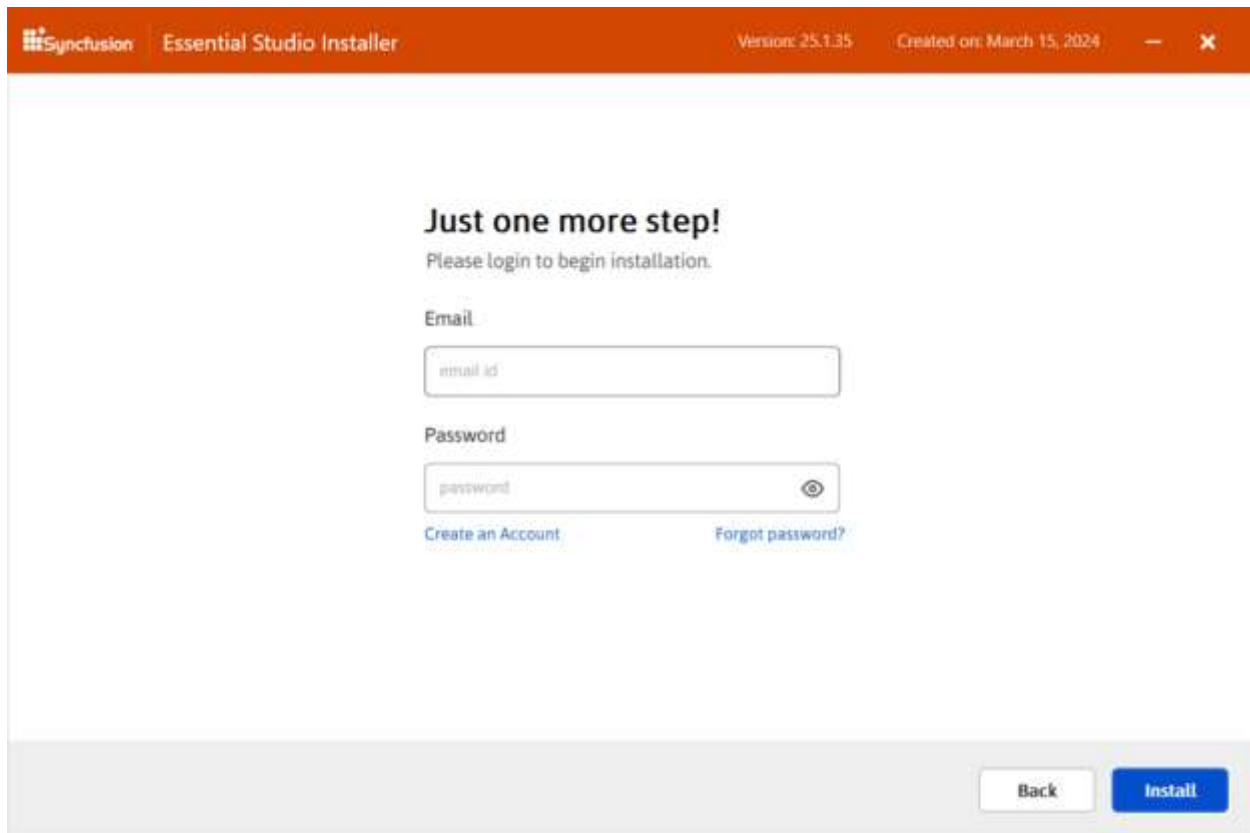
- ☒ Install Demos
 - ☒ JavaScript
- ☒ Configure Syncfusion Extensions in Visual Studio
 - ☒ JavaScript
- ☒ Create Desktop Shortcut
- ☒ Create Start Menu Shortcut

Download Size	Installation Size
1.1 GB	10.38 GB

[Back](#) [Next](#)

Additional settings

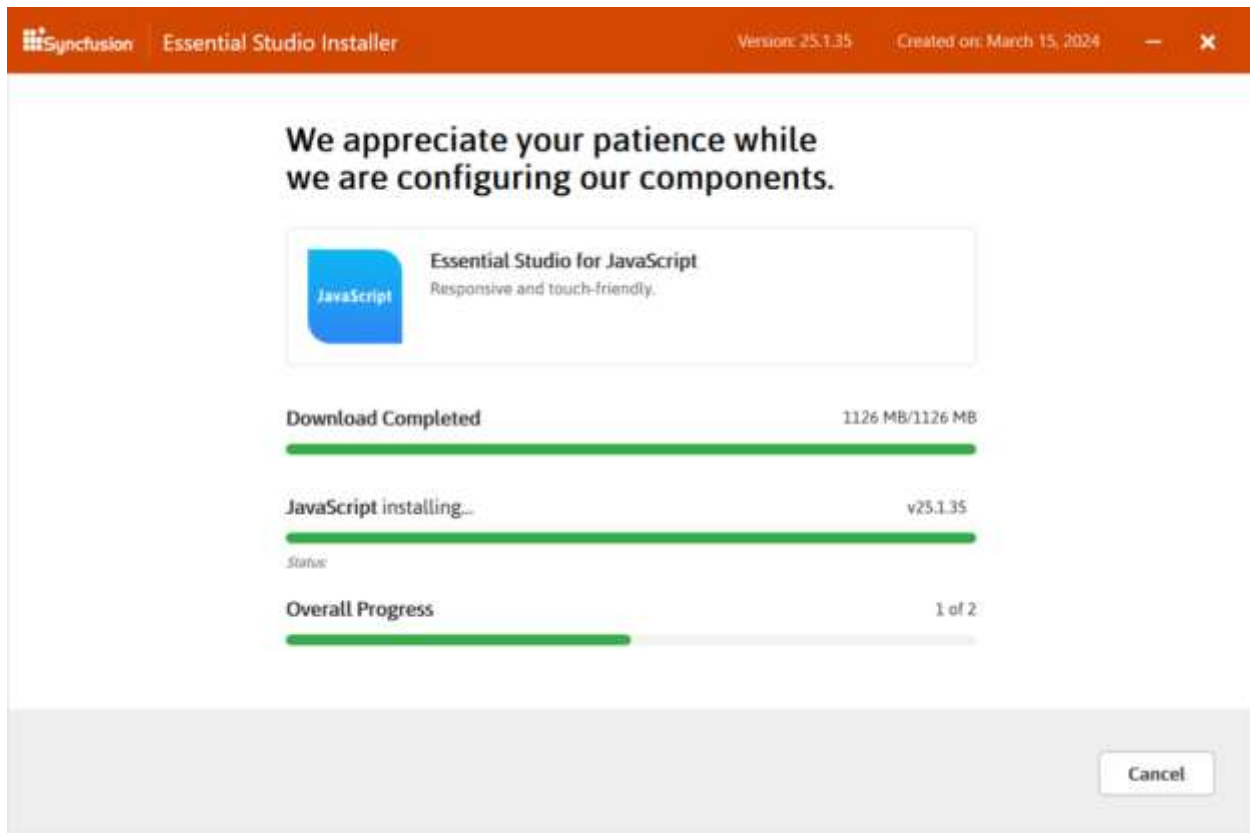
- Select the **Install Demos** check box to install Syncfusion samples, or leave the check box unchecked, if you do not want to install Syncfusion samples
 - Select the **Configure Syncfusion Extensions controls in Visual Studio** checkbox to configure the Syncfusion Extensions in Visual Studio or clear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



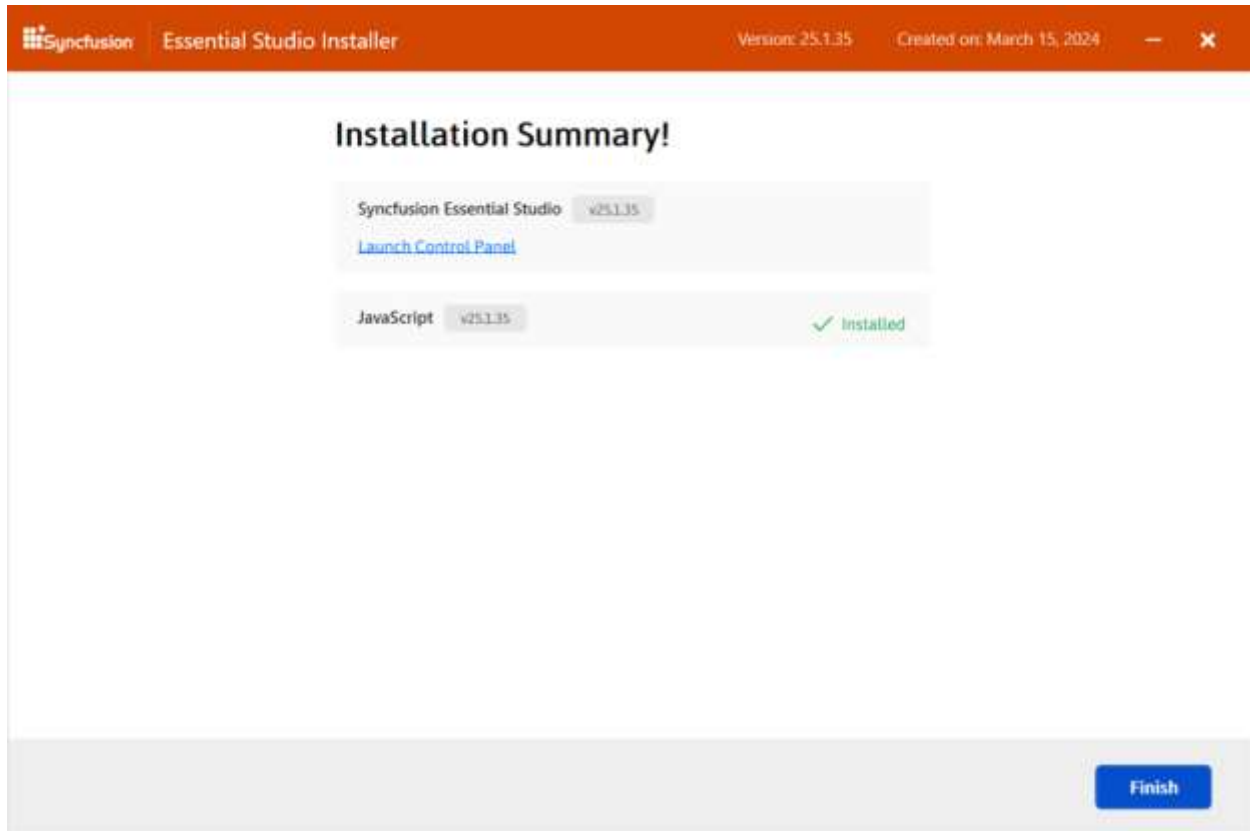
The screenshot shows the 'Syncfusion Essential Studio Installer' window. The title bar is orange and contains the Syncfusion logo, the text 'Essential Studio Installer', the version 'Version: 25.1.35', the creation date 'Created on: March 15, 2024', and window control buttons. The main content area has a white background with the heading 'Just one more step!' and the instruction 'Please login to begin installation.' Below this are two input fields: 'Email' with a placeholder 'email id' and 'Password' with a placeholder 'password' and a toggle eye icon. At the bottom of the form are two links: 'Create an Account' and 'Forgot password?'. At the bottom right of the window are two buttons: 'Back' and 'Install'.

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

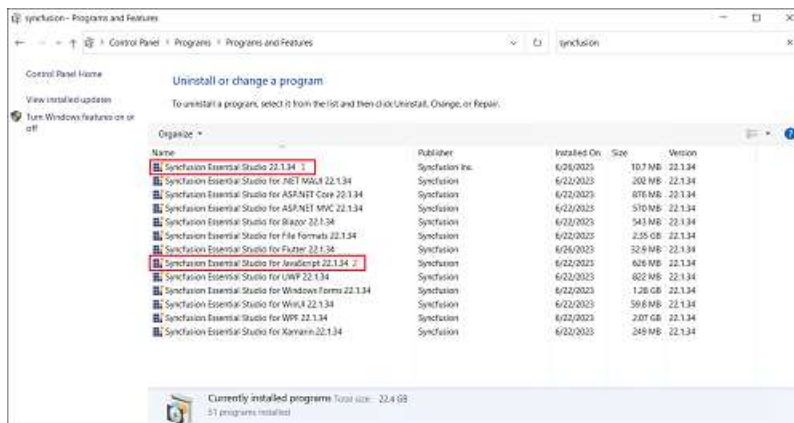
10. The download and installation/uninstallation progress will be displayed as shown below.



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



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



Uninstallation

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

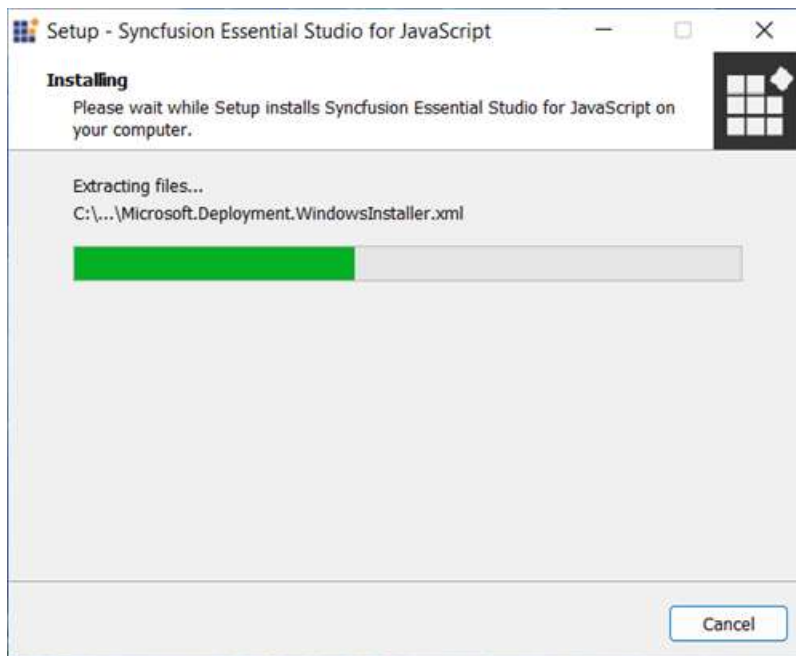
- Uninstall the JavaScript – EJ2 using the Syncfusion JavaScript – EJ2 web installer
- Uninstall the JavaScript – EJ2 from Windows Control Panel

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

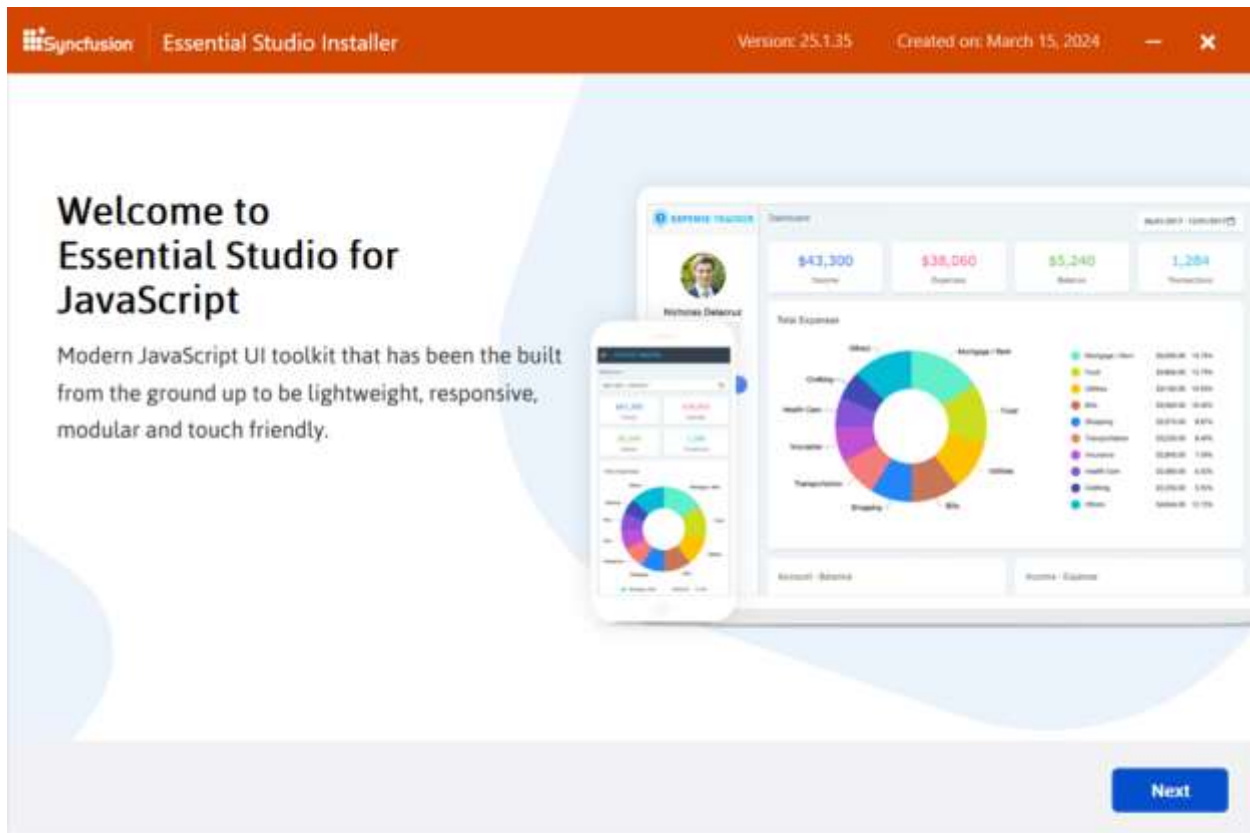
Option 1: Uninstall the JavaScript–EJ2 using the Syncfusion JavaScript–EJ2 web installer

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

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

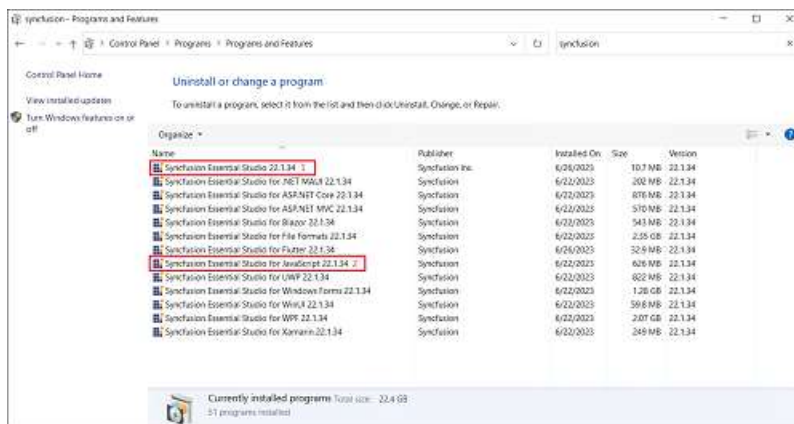


The Syncfusion JavaScript – EJ2 Web Installer’s welcome wizard will be displayed. Click the Next button



Option 2: Uninstall the JavaScript–EJ2 from Windows Control Panel

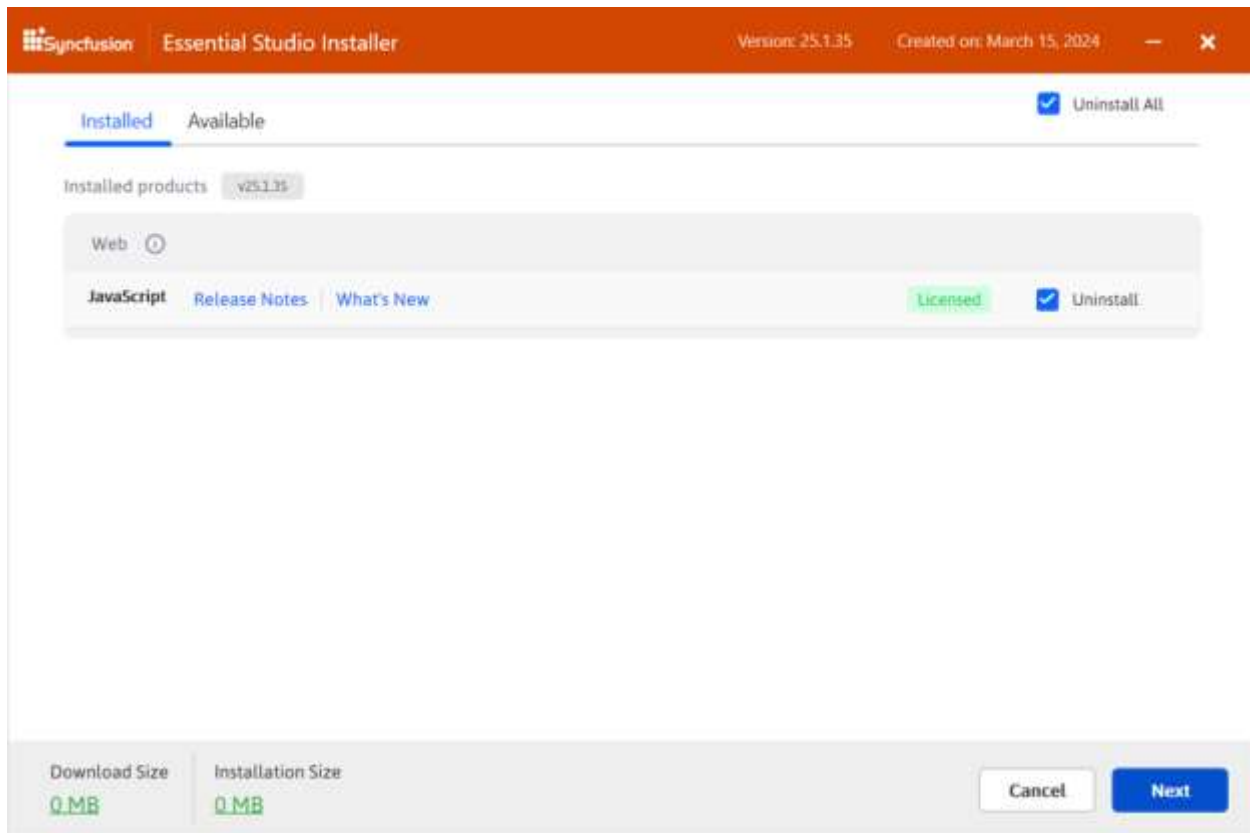
You can uninstall all the installed products by selecting the **Syncfusion Essential Studio {version}** entry (element 1 in the below screenshot) from the Windows control panel, or you can uninstall JavaScript – EJ2 alone by selecting the **Syncfusion Essential Studio for JavaScript – EJ2 {version}**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



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

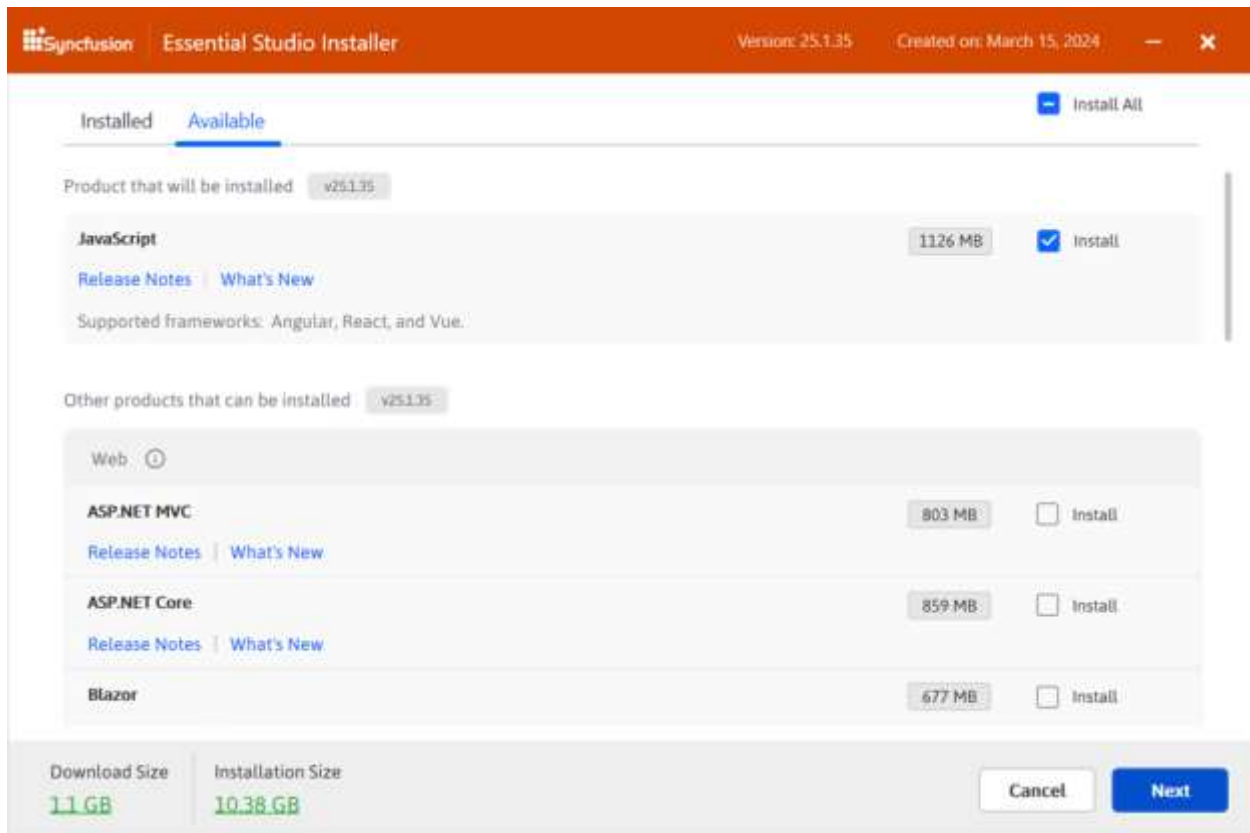
1. The Platform Selection Wizard will appear. From the **Installed** tab, select the products to be uninstalled. To select all products, check the **Uninstall All** checkbox. Click the Next button.

Installed

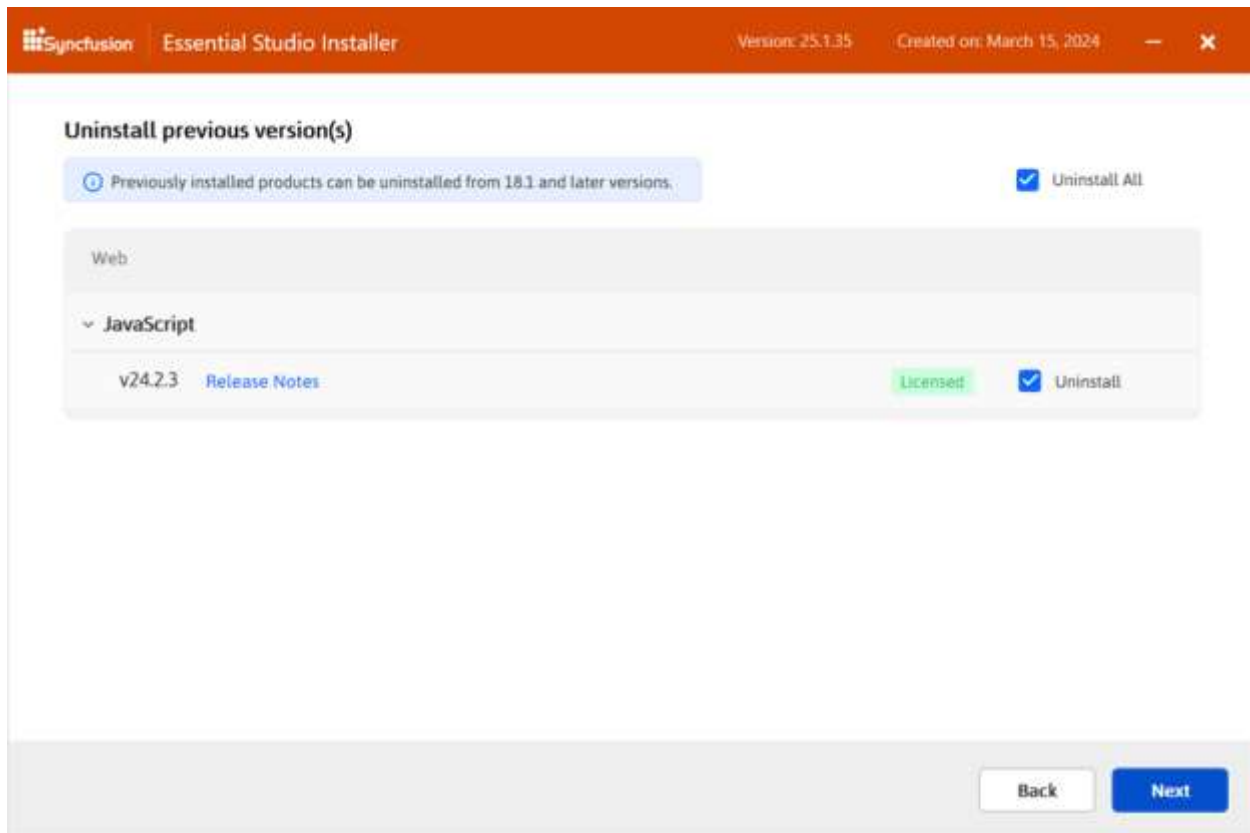


You can also select the products to be installed from the **Available** tab. Click the Next button.

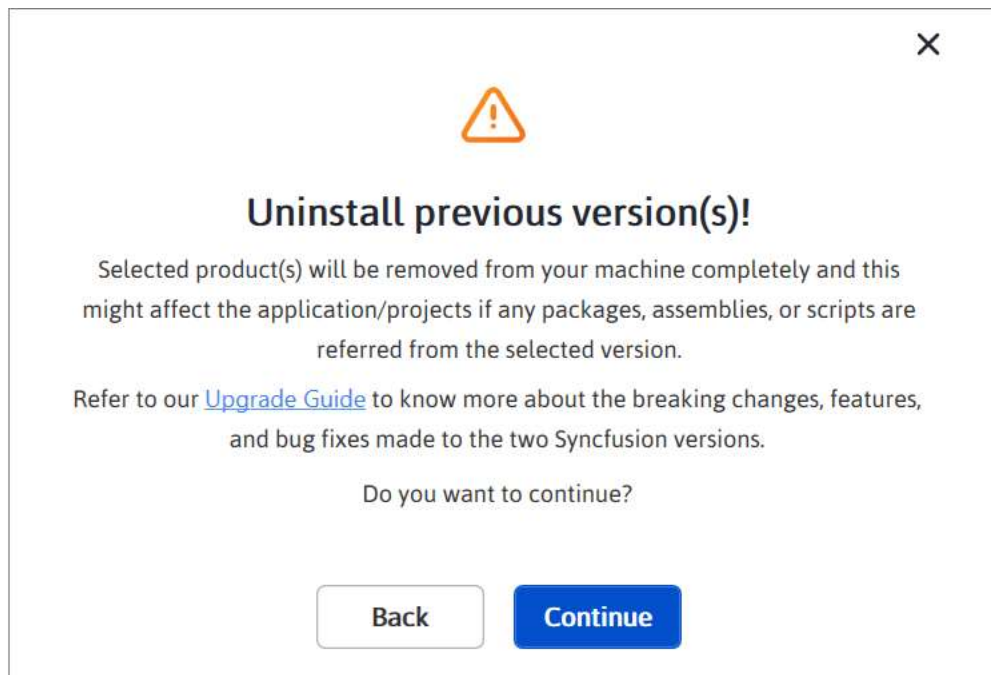
Available



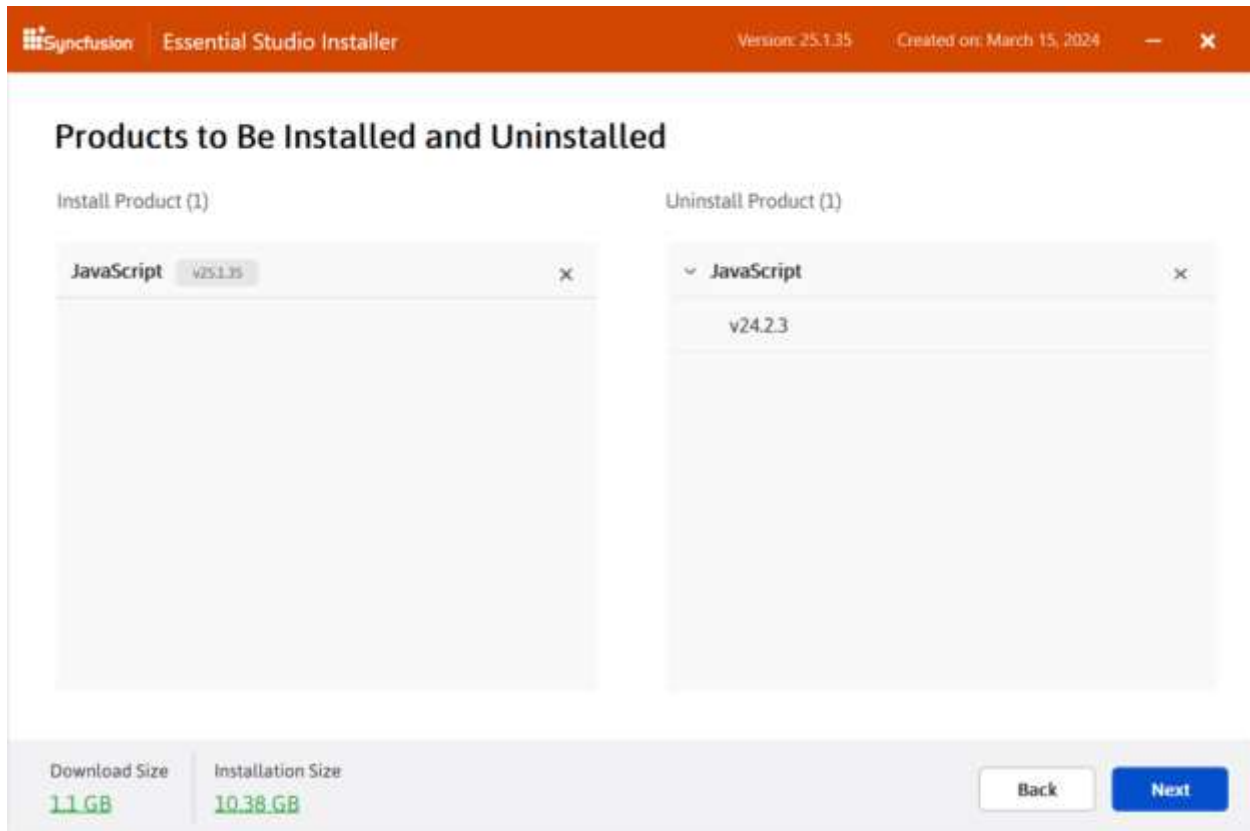
2. If any other products selected for installation, Uninstall previous version wizard will be displayed with previous version(s) installed for the selected products. Here you can view the list of installed previous versions for the selected products. Select **Uninstall All** checkbox to select all the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



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



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



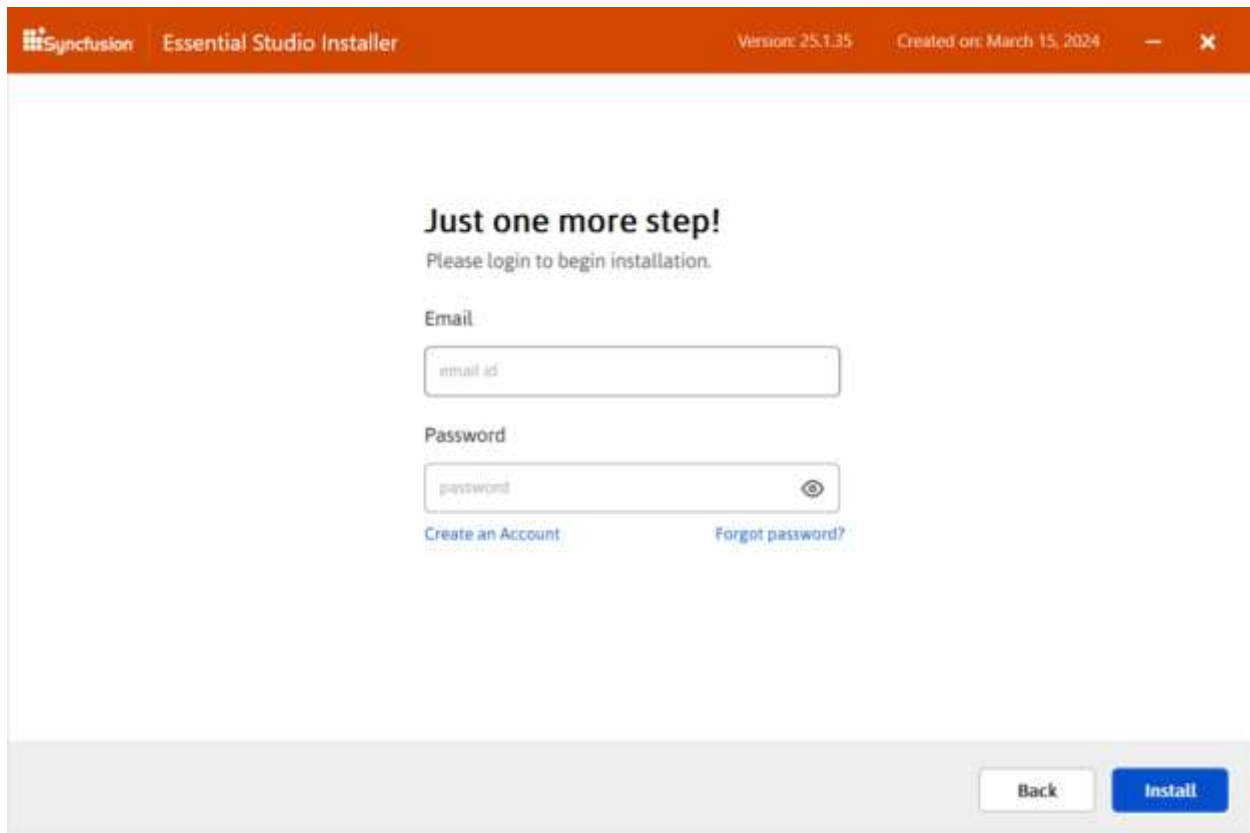
Note: By clicking the **Download Size** and **Installation Size** links, you can determine the approximate size of the download and installation

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

The screenshot shows the 'Configuration' window of the Syncfusion Essential Studio Installer. The window has an orange header bar with the Syncfusion logo, the title 'Essential Studio Installer', and metadata: 'Version: 25.1.35' and 'Created on: March 15, 2024'. The main content area is divided into two columns. The left column contains three sections: 'Download Location' with a text box showing 'C:\ProgramData\Syncfusion\25.1.35\Downloads\' and a 'Browse' button; 'Installation Location' with a text box showing 'C:\Program Files (x86)\Syncfusion\Essential Studio\' and a 'Browse' button; and 'Demos Location' with a text box showing 'C:\Users\Public\Documents\Syncfusion\' and a 'Browse' button. Below these is a checked checkbox for 'I Agree to the [License Terms](#) and [Privacy Policy](#)'. The right column is titled 'Additional Settings' and contains a list of options, all of which are checked: 'Install Demos', 'JavaScript', 'Configure Syncfusion Extensions in Visual Studio', 'JavaScript', 'Create Desktop Shortcut', and 'Create Start Menu Shortcut'. At the bottom, there is a summary table with 'Download Size' (1.1 GB) and 'Installation Size' (10.38 GB). To the right of the table are 'Back' and 'Next' buttons.

Download Size	Installation Size
1.1 GB	10.38 GB

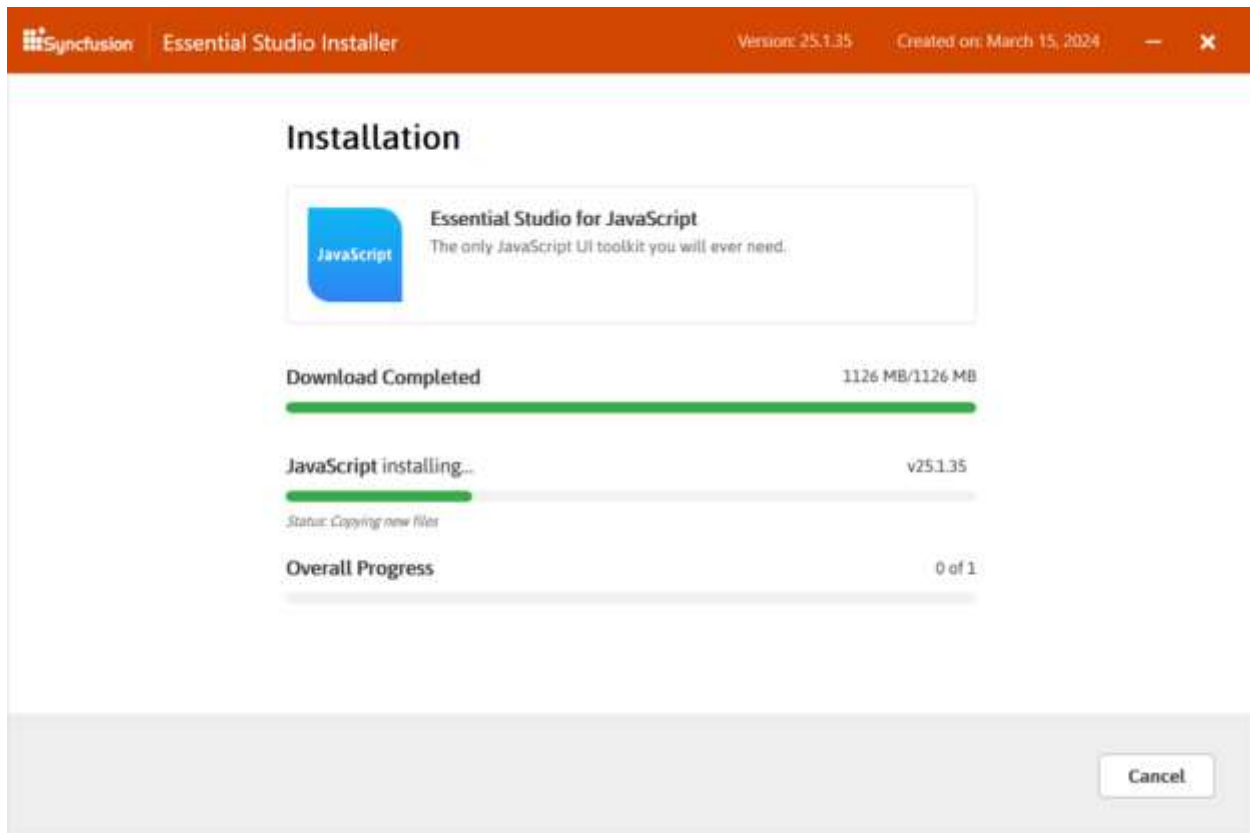
6. After reading the License Terms and Conditions, check the **I agree to the License Terms and Privacy Policy** check box. Click the Next button.
7. The login wizard will appear. You must enter your Syncfusion email address and password. If you do not already have a Syncfusion account, you can create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



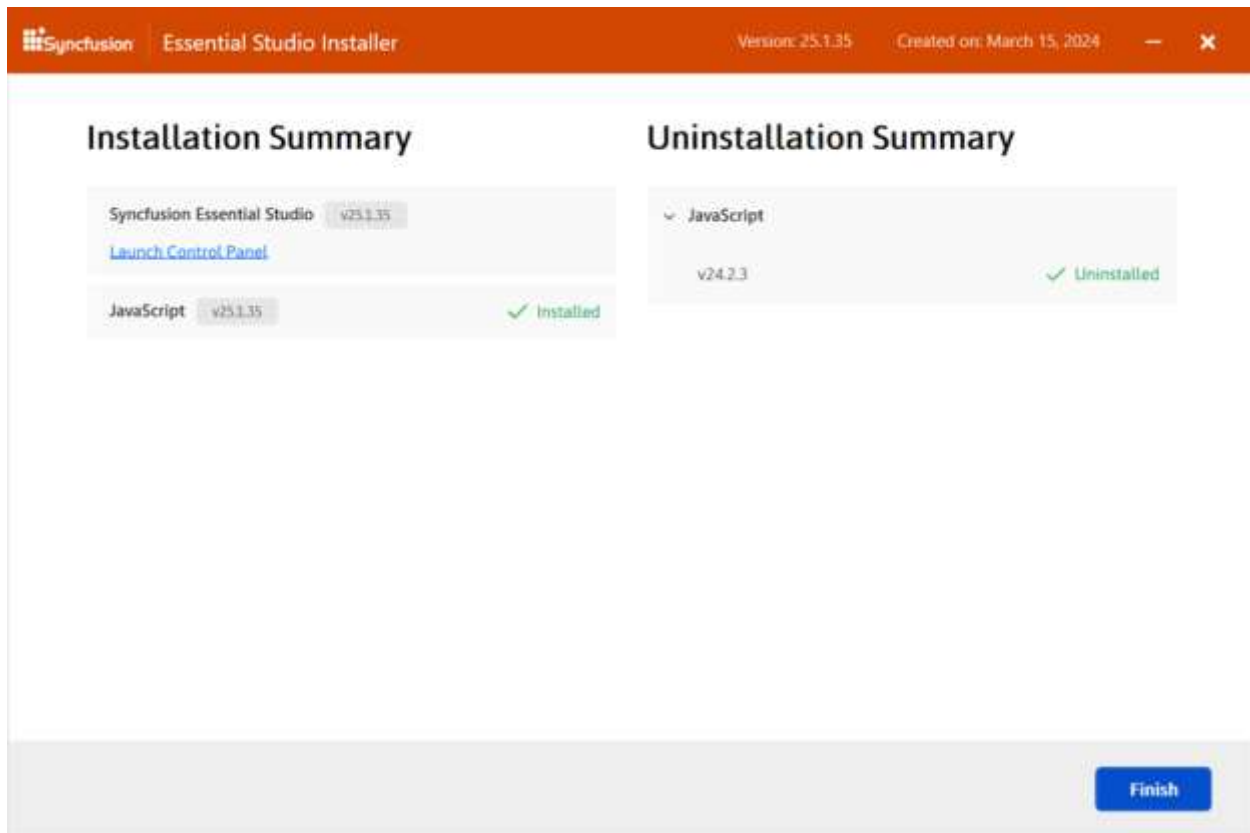
The screenshot shows the 'Syncfusion Essential Studio Installer' window. The title bar is orange and contains the Syncfusion logo, the text 'Essential Studio Installer', the version 'Version: 25.1.35', the creation date 'Created on: March 15, 2024', and standard window controls. The main content area is white and features the heading 'Just one more step!' followed by the instruction 'Please login to begin installation.' Below this are two input fields: 'Email' with a placeholder 'email id' and 'Password' with a placeholder 'password' and a toggle icon. At the bottom of the form are two links: 'Create an Account' and 'Forgot password?'. At the bottom right of the window are two buttons: 'Back' and 'Install'.

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

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



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



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

Installation using offline installer

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

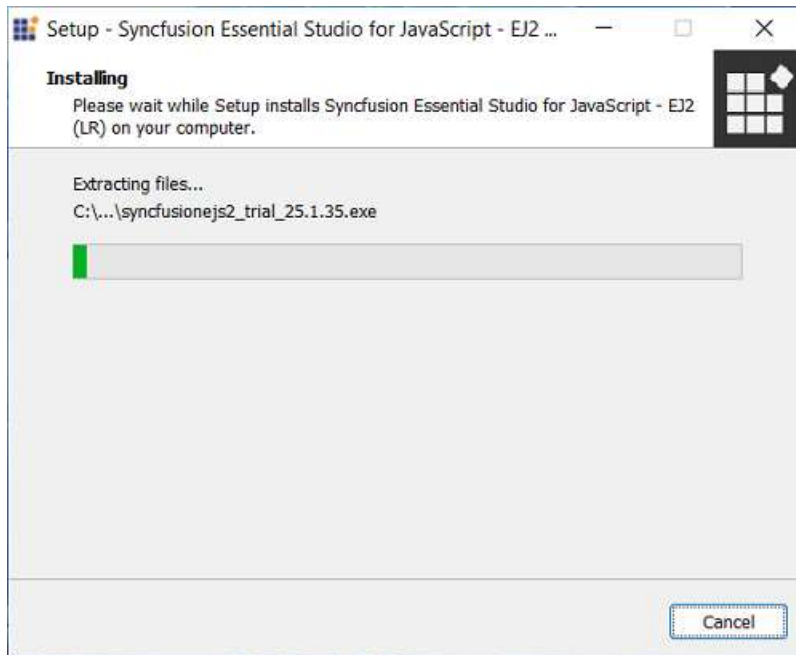
The frameworks listed below are supported in this installer.

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

Installing with UI

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

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

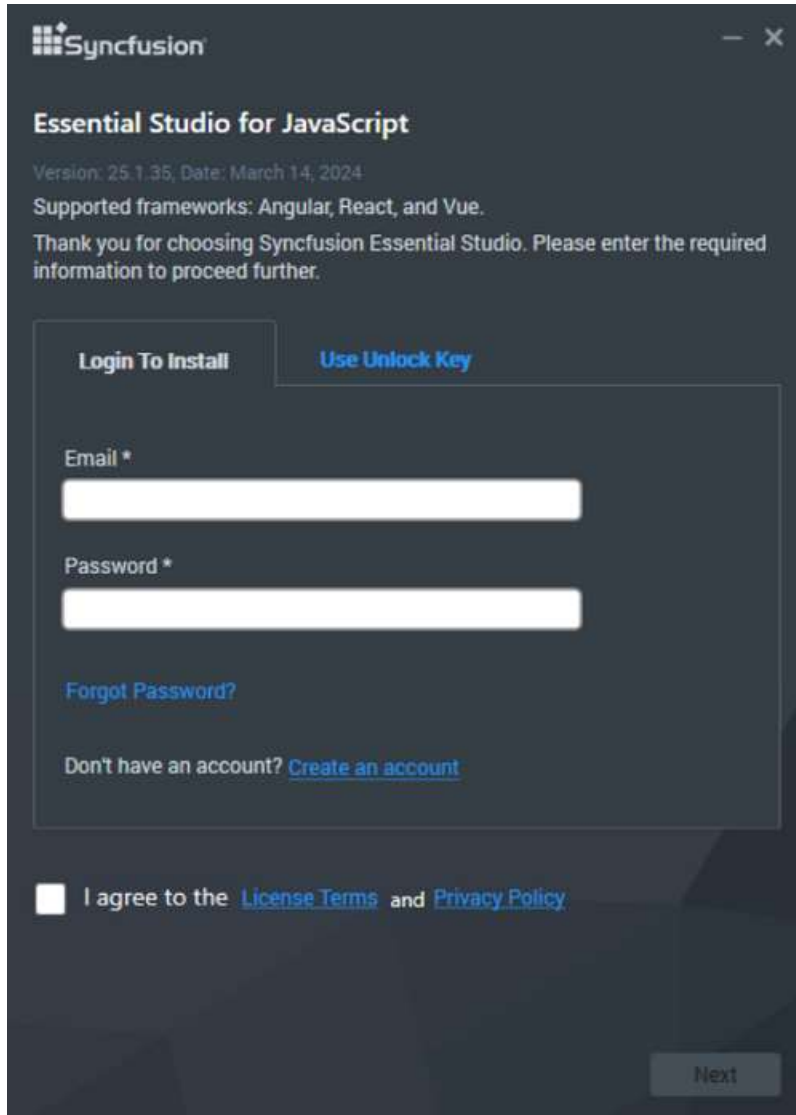


Note: The Installer wizard extracts the syncfusionejs2 (version).exe dialog, which displays the package's unzip operation.

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

Login To Install

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



Syncfusion

Essential Studio for JavaScript

Version: 25.1.35, Date: March 14, 2024

Supported frameworks: Angular, React, and Vue.

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

Login To Install **Use Unlock Key**

Email *

Password *

[Forgot Password?](#)

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

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

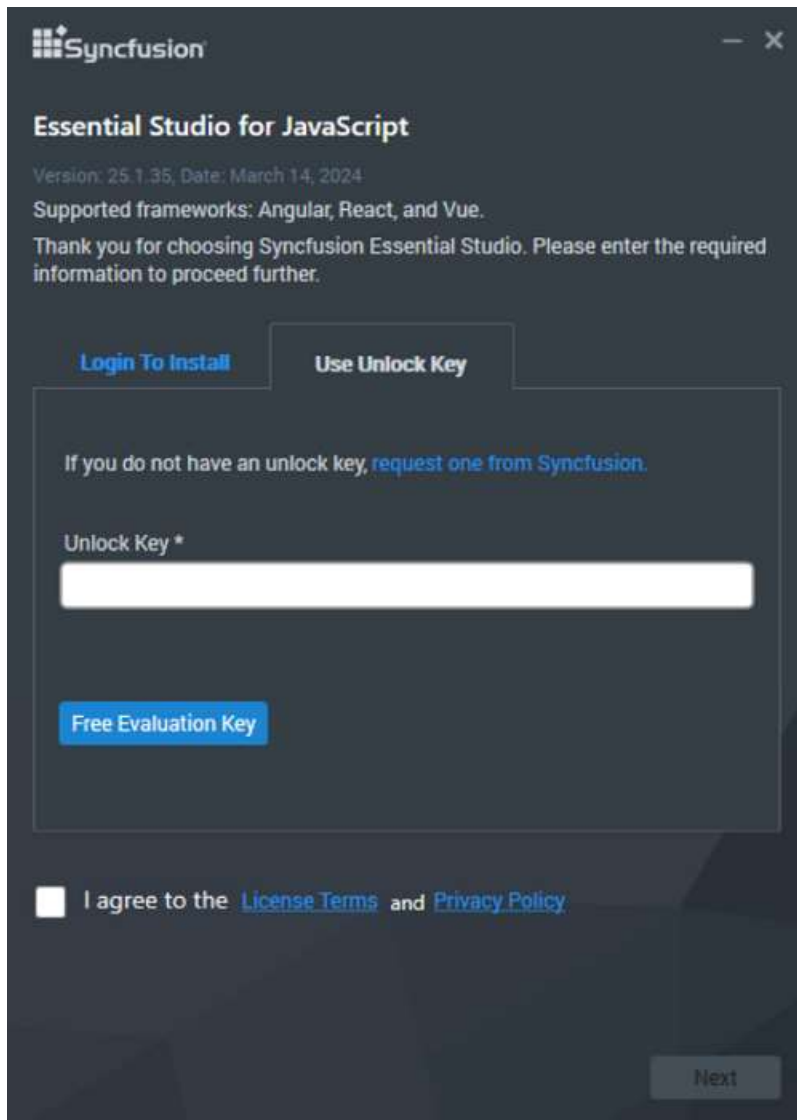
Next

Use Unlock Key

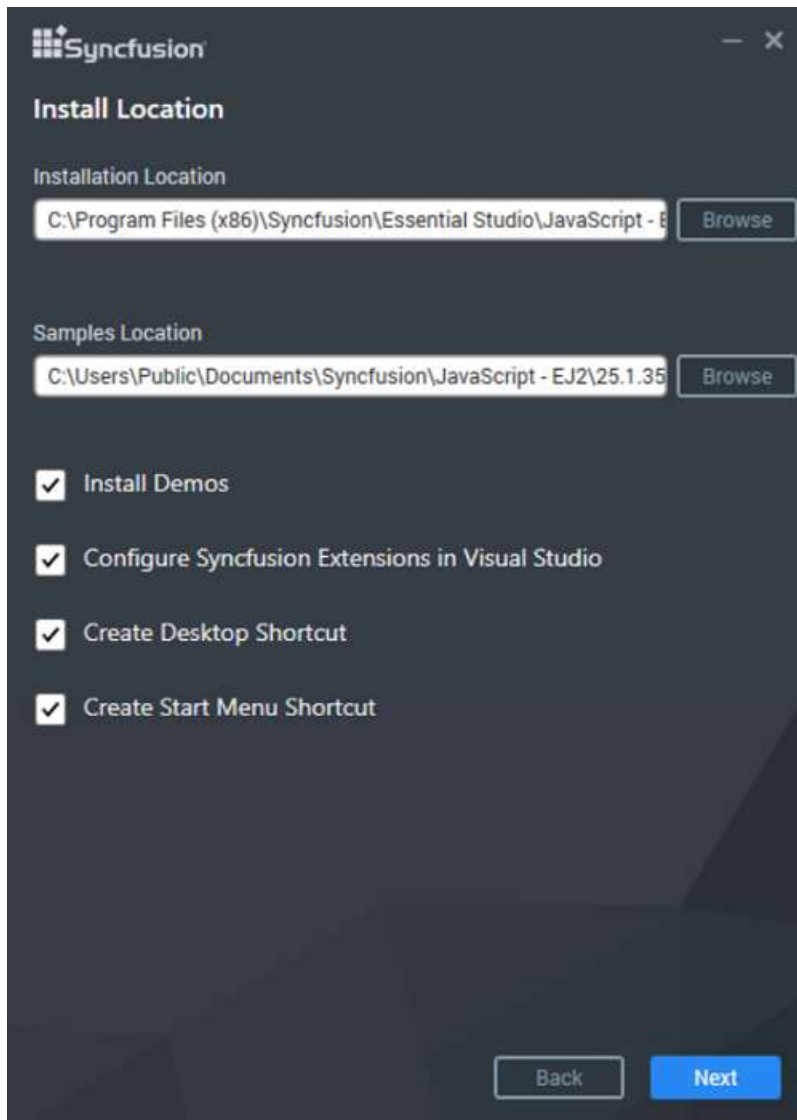
Unlock keys are used to unlock the Syncfusion offline installer, and they are platform and version specific. You should use either Syncfusion licensed or trial Unlock key to unlock Syncfusion JavaScript – EJ2 installer.

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

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

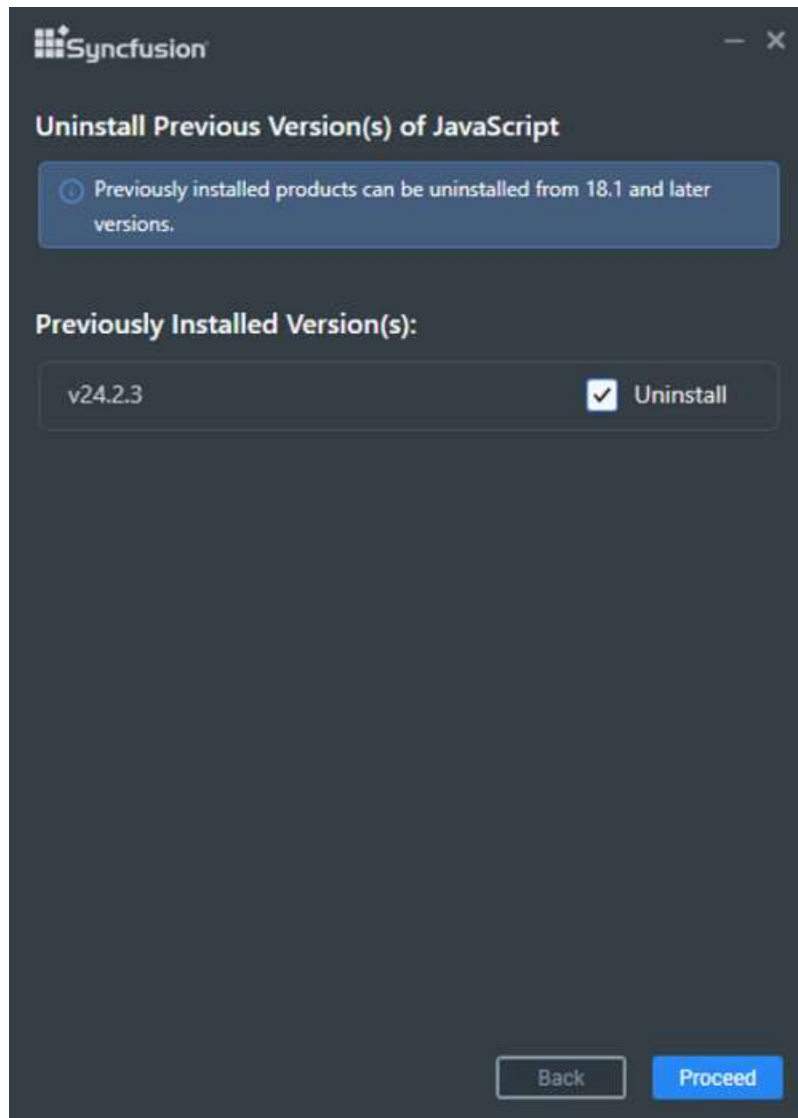


3. After reading the License Terms and Privacy Policy, check the **"I agree to the License Terms and Privacy Policy"** check box. Click the Next button.
4. Change the install and sample locations here. You can also change the Additional settings. Click Next/Install to install with the default settings.



Additional Settings

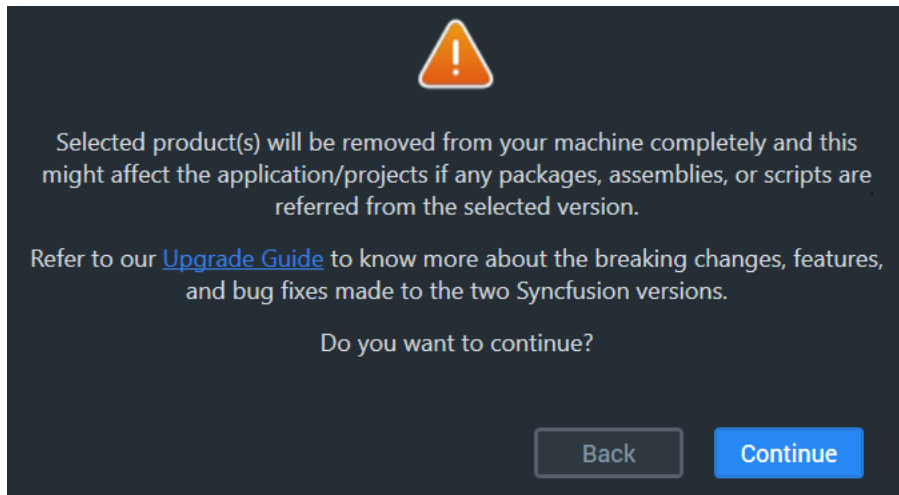
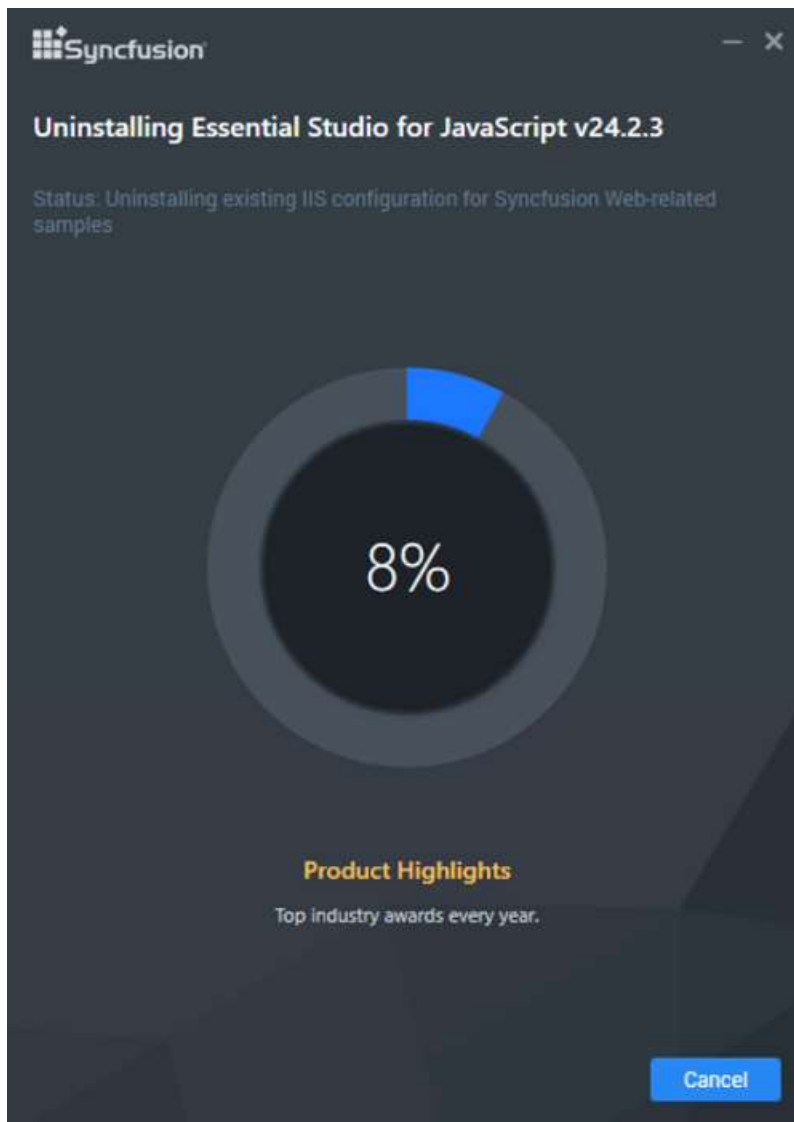
- Select the **Install Demos** check box to install Syncfusion samples, or leave the check box unchecked, if you do not want to install Syncfusion samples
 - Select the **Configure Syncfusion Extensions controls in Visual Studio** checkbox to configure the Syncfusion Extensions in Visual Studio or clear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

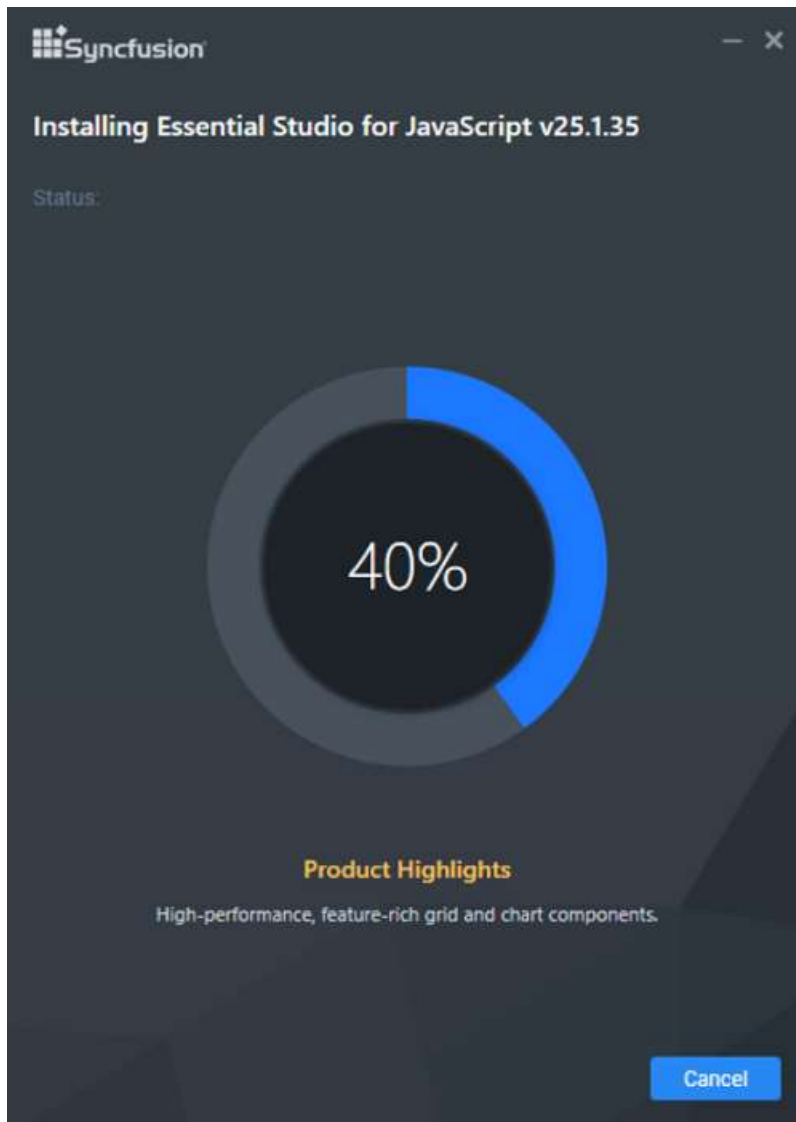


Note: From the 2021 Volume 1 release, Syncfusion has added the option to uninstall previous versions from 18.1 while installing the new version.

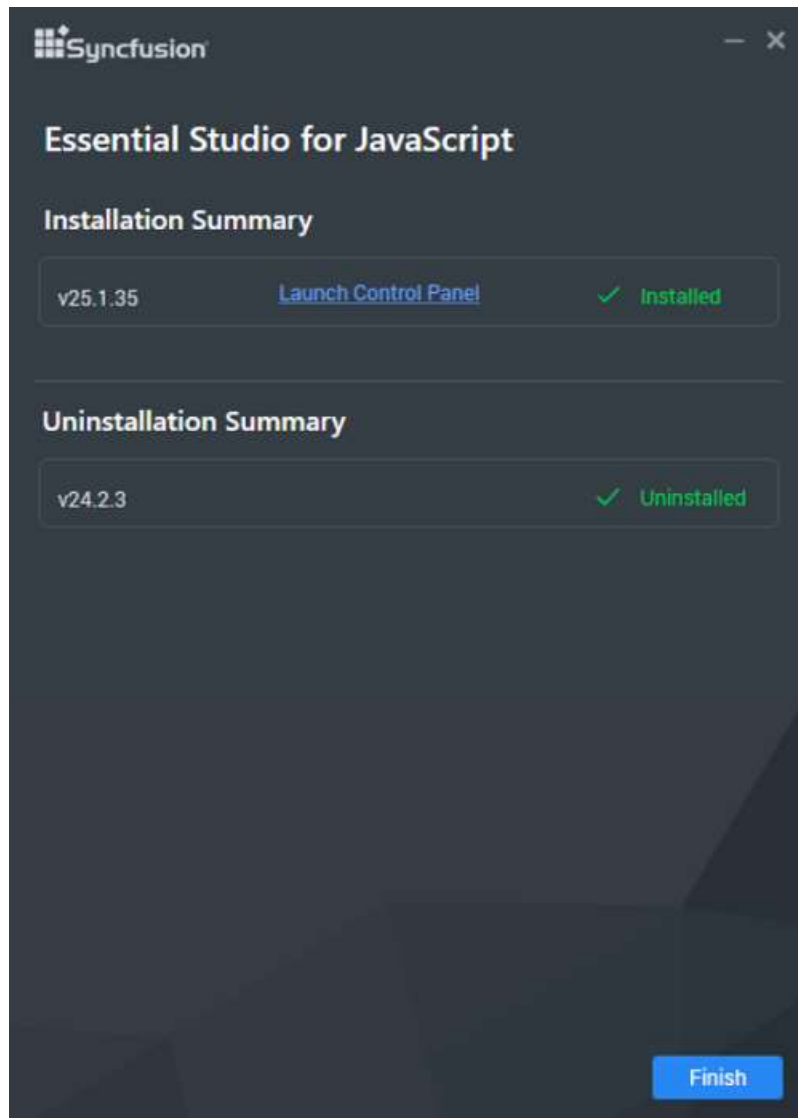
Note: If any version is selected to uninstall, a confirmation screen will appear; if continue is selected, the Progress screen will display the uninstall and install progress, respectively. If none of the versions are chosen to be uninstalled, only the installation progress will be displayed.

Confirmation Alert

**Uninstall Progress:**

Install Progress

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



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

Installing in Silent Mode

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

Command Line Installation

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

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

4. Copy the extracted syncfusionejs2_(version).exe file in local drive.
5. Exit the Wizard.
6. Run Command Prompt in administrator mode and enter the following arguments.

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

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

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

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

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

Command Line Uninstallation

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

1. Run the Syncfusion JavaScript – EJ2 installer by double-clicking it. The Installer Wizard automatically opens and extracts the package.
2. The syncfusionejs2_(version).exe file will be extracted into the Temp directory.
3. Run %temp%. The Temp folder will be opened. The syncfusionejs2_(version).exe file will be located in one of the folders.
4. Copy the extracted syncfusionejs2_(version).exe file in local drive.
5. Exit the Wizard.
6. Run Command Prompt in administrator mode and enter the following arguments.

Arguments: "Copied installer file path\ syncfusionejs2_(version).exe" /uninstall silent

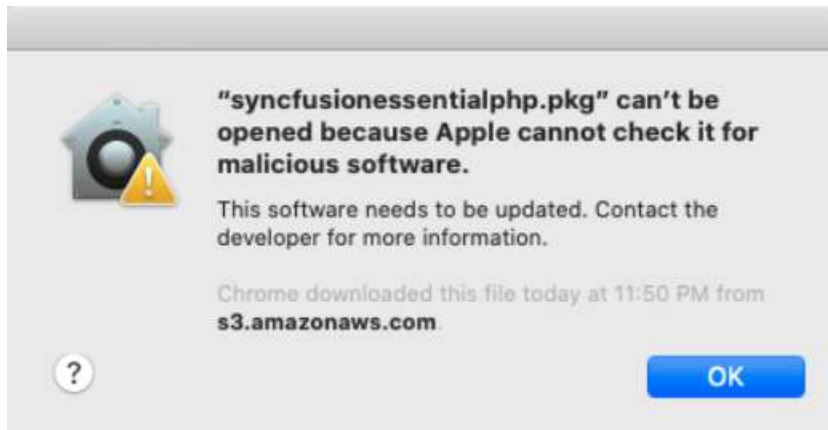
Example: "D:\Temp\syncfusionejs2_x.x.x.x.exe" /uninstall silent

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

Installing Syncfusion JavaScript – EJ2 Mac Installer

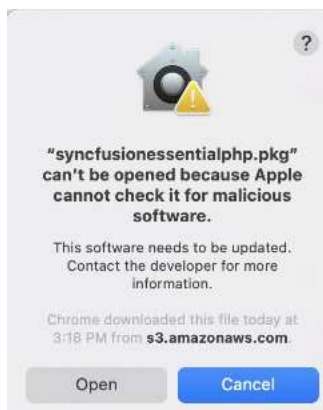
Steps to resolve the warning message in Catalina OS or later

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



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

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

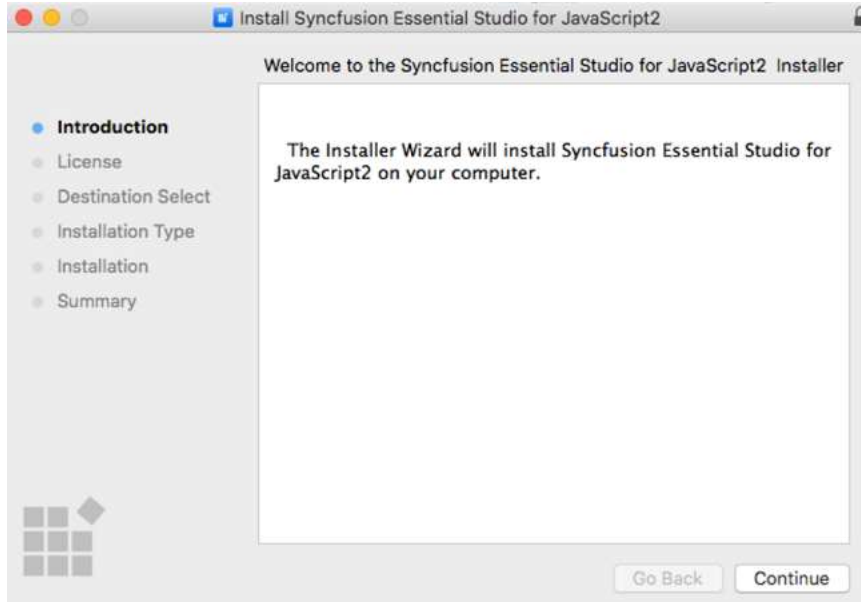


3. When you click "Open" the installer window will be opened.

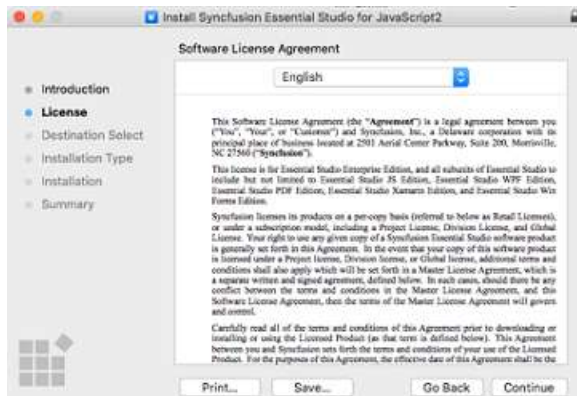
Step-by-Step Installation

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

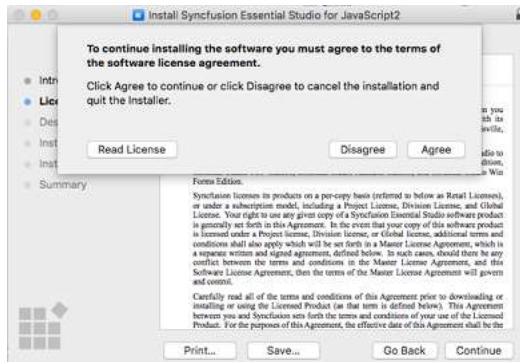
1. Double-click the Syncfusion Essential Studio JavaScript - EJ2 Mac installer(.pkg) file. The installer Wizard opens. Click Continue.



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

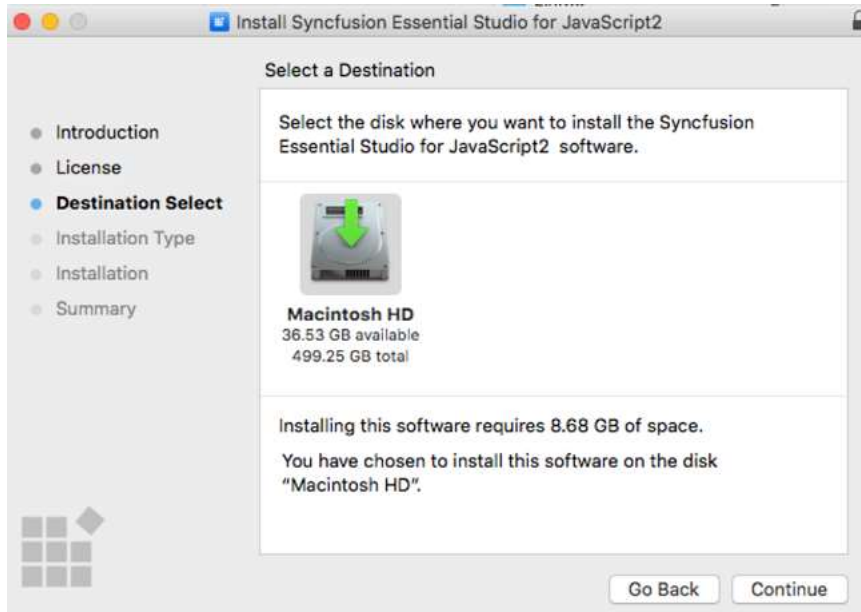


3. The License Agreement's Confirmation window will appear. If you have read the Software License Agreement, click **Agree**.

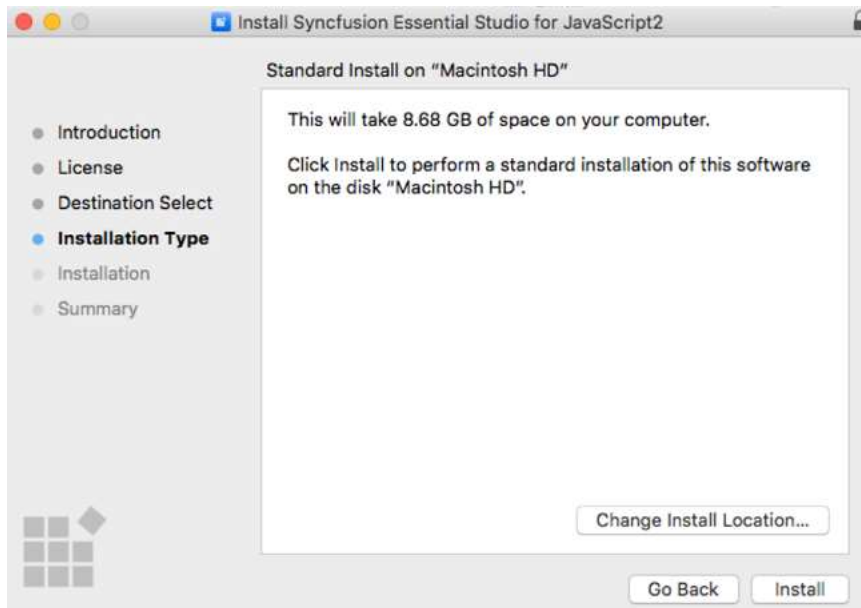


Note: The Unlock key is not required to install the Mac installer. The Syncfusion Essential Studio JavaScript - EJ2 Mac installer can be used for development purposes without registering the Unlock key.

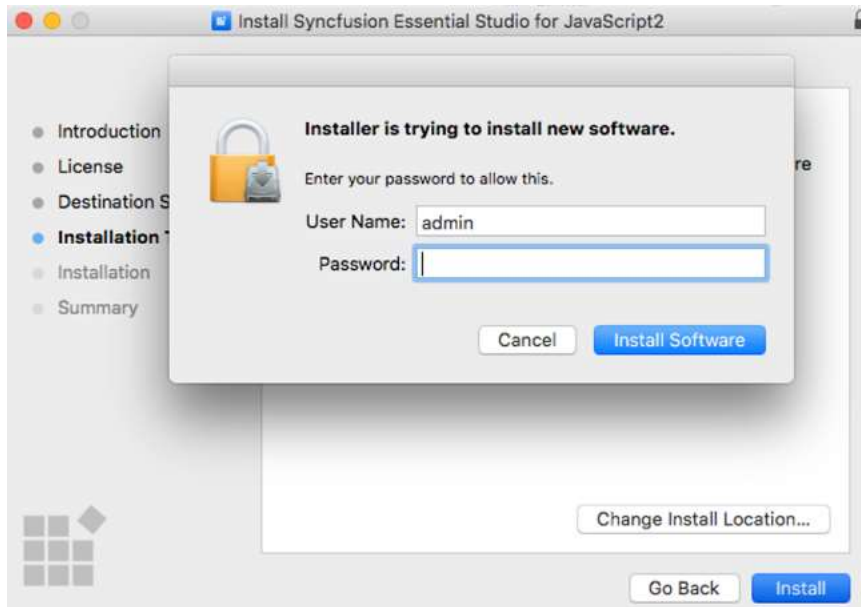
4. The Destination select wizard will appear. You can choose which disc to install the Syncfusion Essential Studio for JavaScript - EJ2 Mac installer on here.



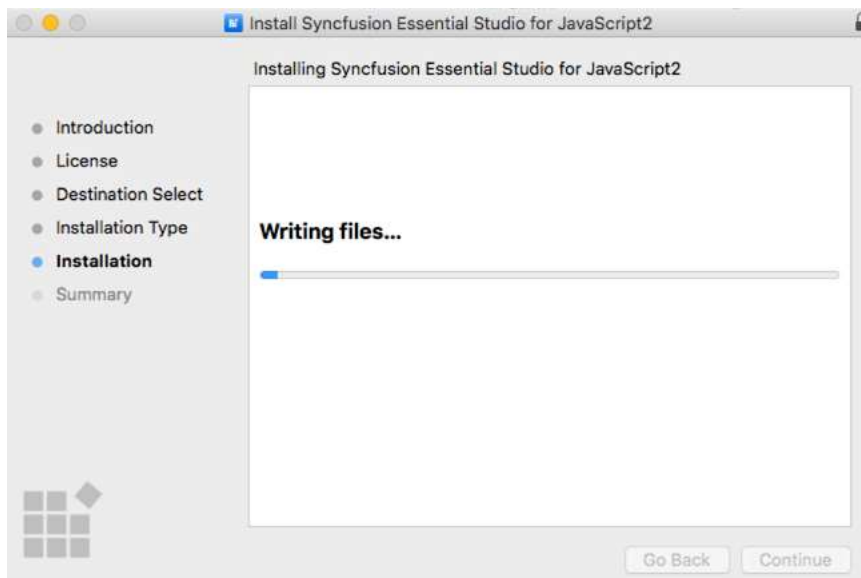
5. The Installation Type wizard will appear. Click Install to begin the standard installation of the Syncfusion Essential Studio JavaScript - EJ2 Mac installer.



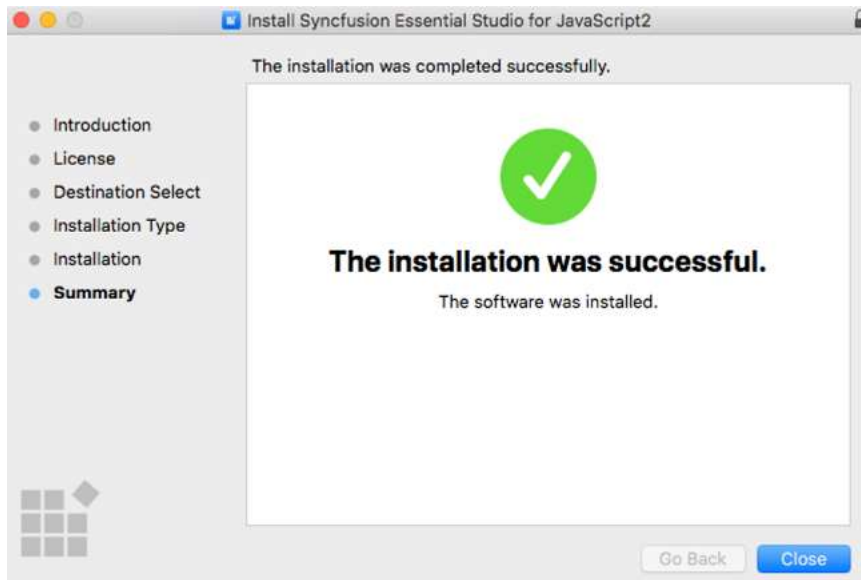
6. The Authentication window will appear. To begin the installation, enter the Mac machine's password and click **Install Software**.



7. The installation process will begin on your machine.

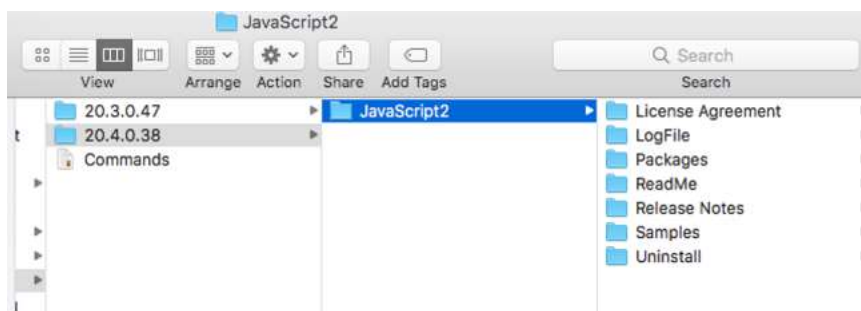


8. Once the installation is complete, the completed screen will be displayed. To exit the installation wizard, click Close.



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

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



License key registration in samples

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

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

Linux Installer

Download Syncfusion JavaScript Linux Installer

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

- Trial Installer - Licensed Installer

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

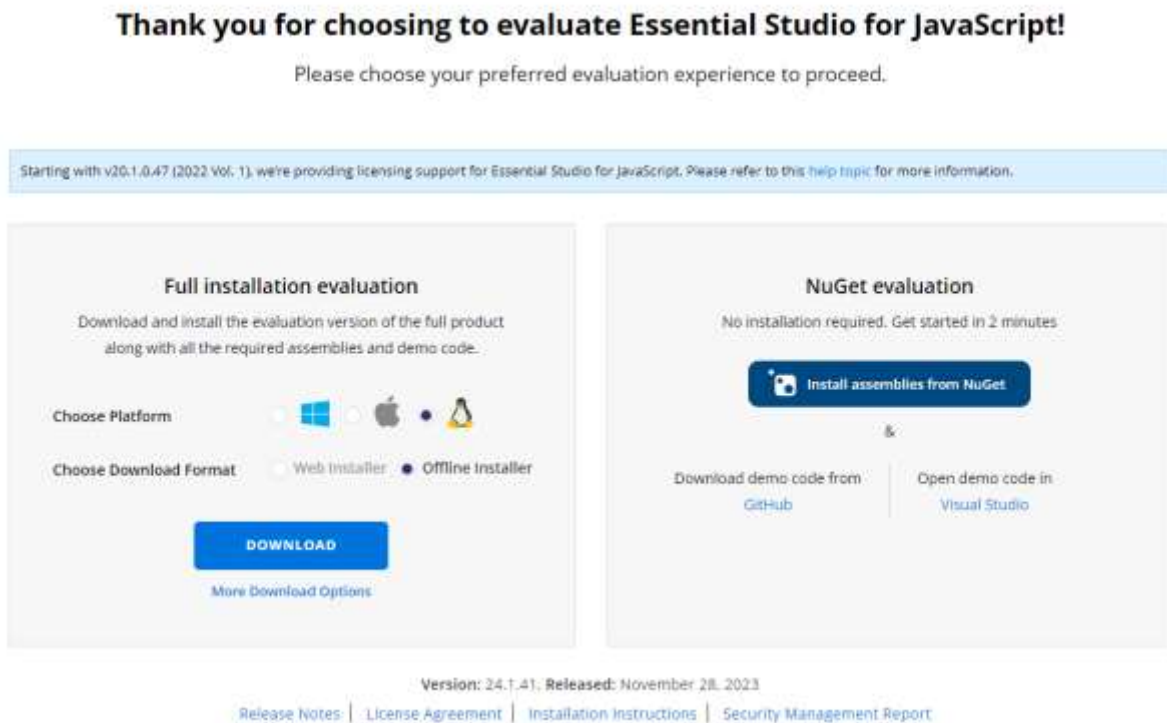
[Download the Trial Version](#)

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

- Download Free Trial Setup
- Start Trials if using components through [NuGet.org](https://www.nuget.org)

Download Free Trial Setup

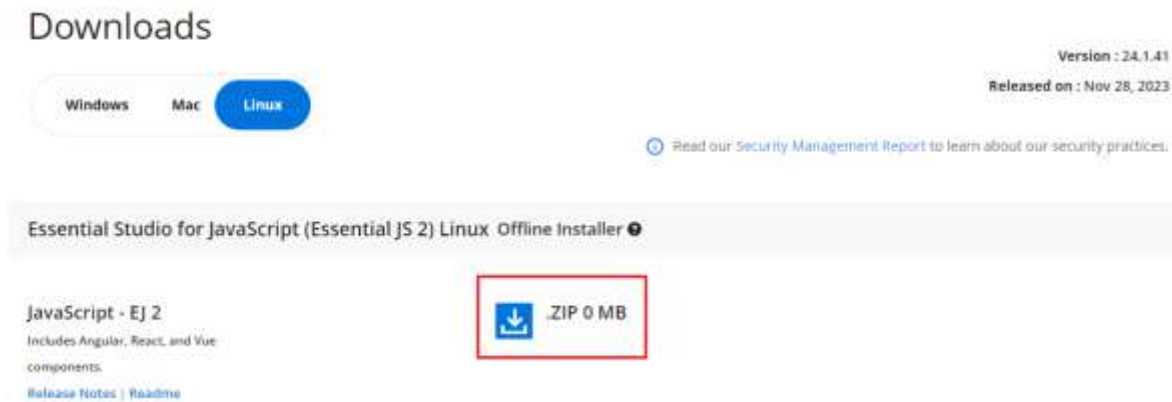
1. You can evaluate our 30-day free trial by visiting the [Download Free Trial](#) page and select the product
2. After completing the required form or logging in with your registered Syncfusion account, you can download the trial installer from the confirmation page. (as shown in below screenshot.)



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



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



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

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

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

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



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

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

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



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

Download the License Version

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



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

Downloads

Windows

Mac

Linux

Version : 24.1.41
Released on : Dec 18, 2023

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

WEB - Essential JS 2

Offline Installer

Blazor

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

.ZIP 652 MB
Checksum Value

ASP.NET Core - EJ 2

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

.ZIP 1092 MB
Checksum Value

JavaScript - EJ 2

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

.ZIP 2066 MB
Checksum Value

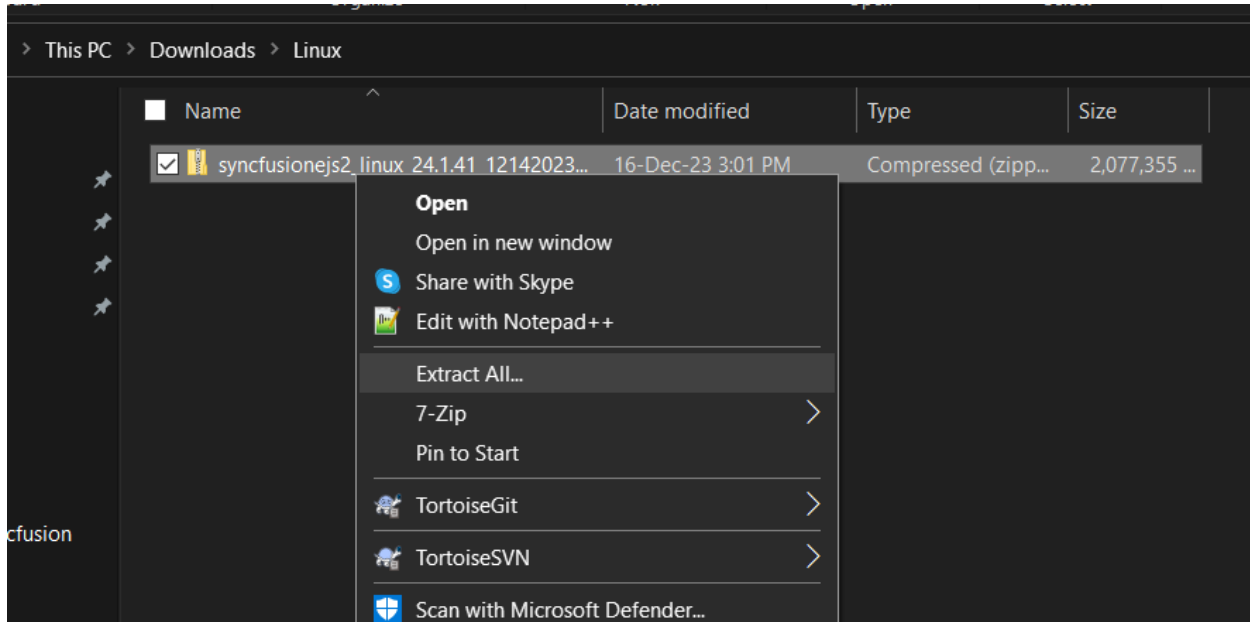
You can also refer to the [JavaScript Linux installer](#) links for step-by-step installation guidelines.

Installing Syncfusion JavaScript Linux installer

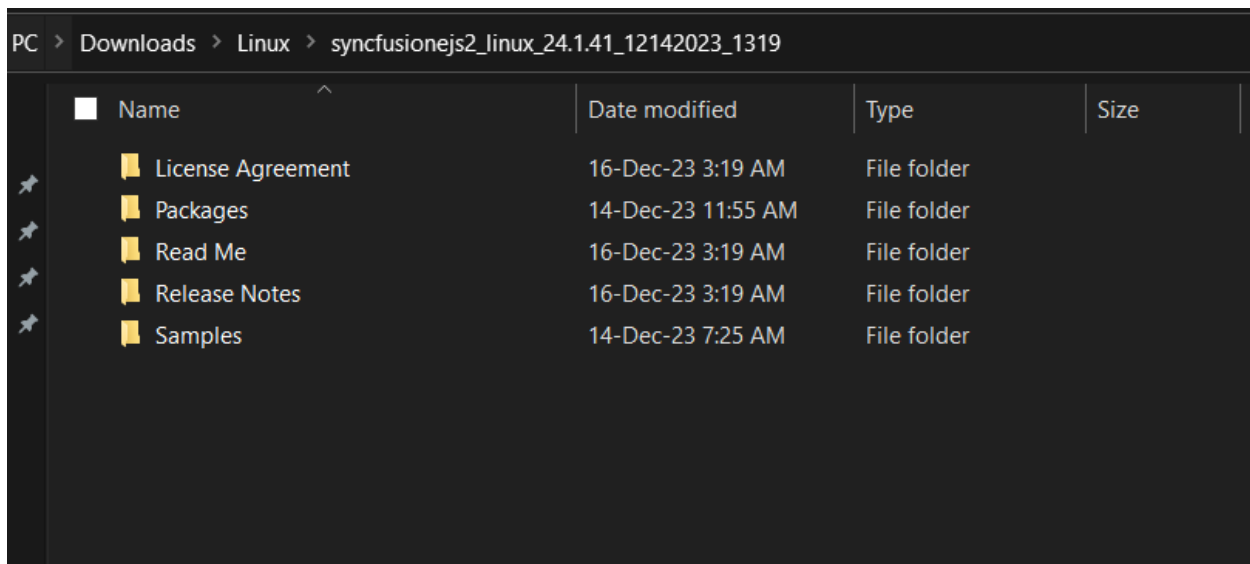
Step-by-Step Installation

The steps below show how to install JavaScript Linux installer.

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



2. The Linux zip file contains the following folders.



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

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

License key registration in samples

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

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

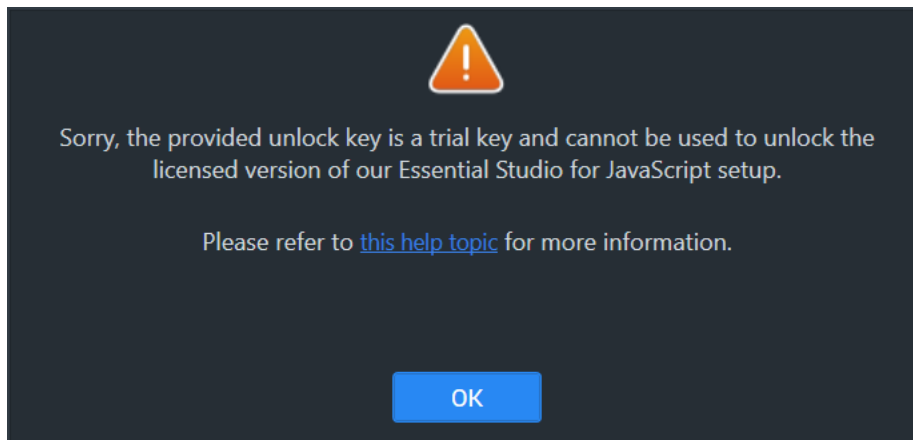
Common Installation Errors

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

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

Unlocking the License Installer using the Trial Key

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



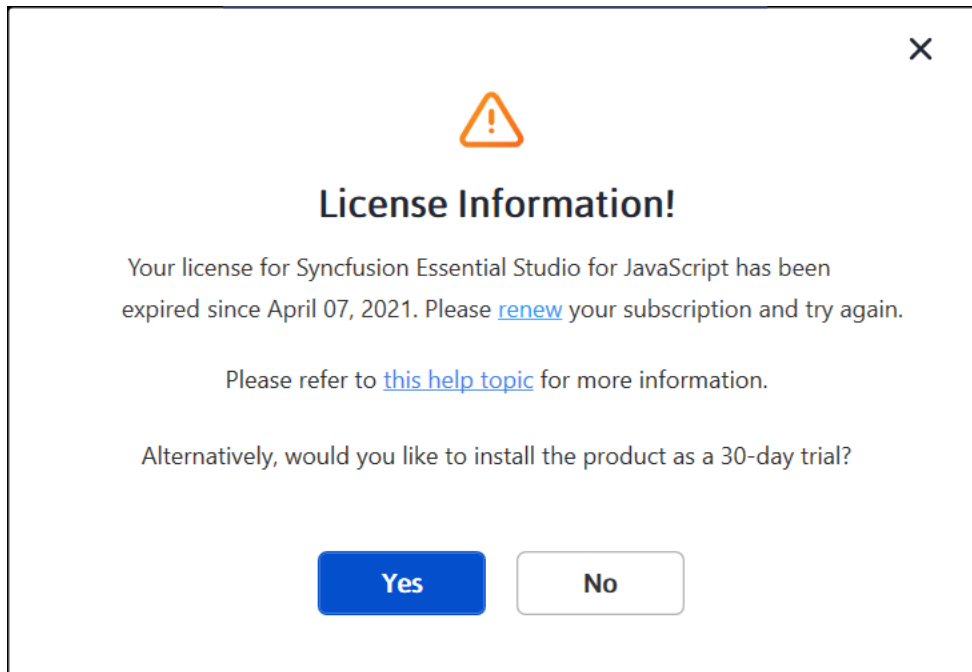
Reason
 You are attempting to use a Trial unlock key to unlock the licensed installer.

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

License has Expired

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

Online Installer



Reason
 This error message will appear if your license has expired.

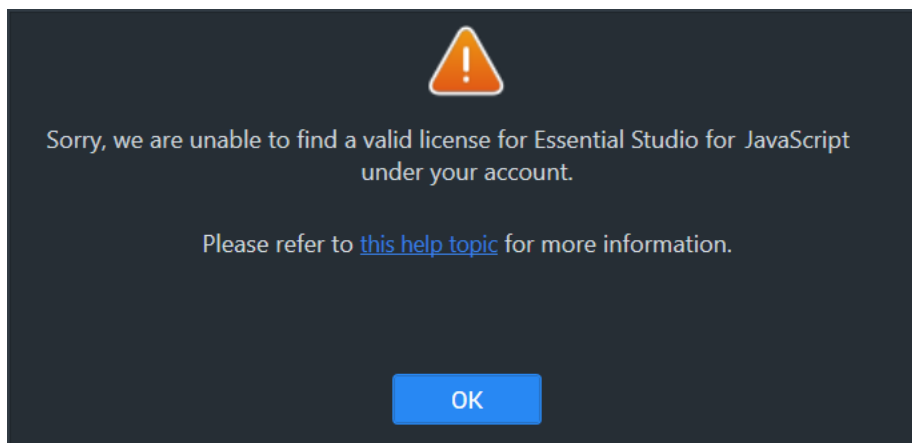
Suggested solution
 You can choose from the options below.

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

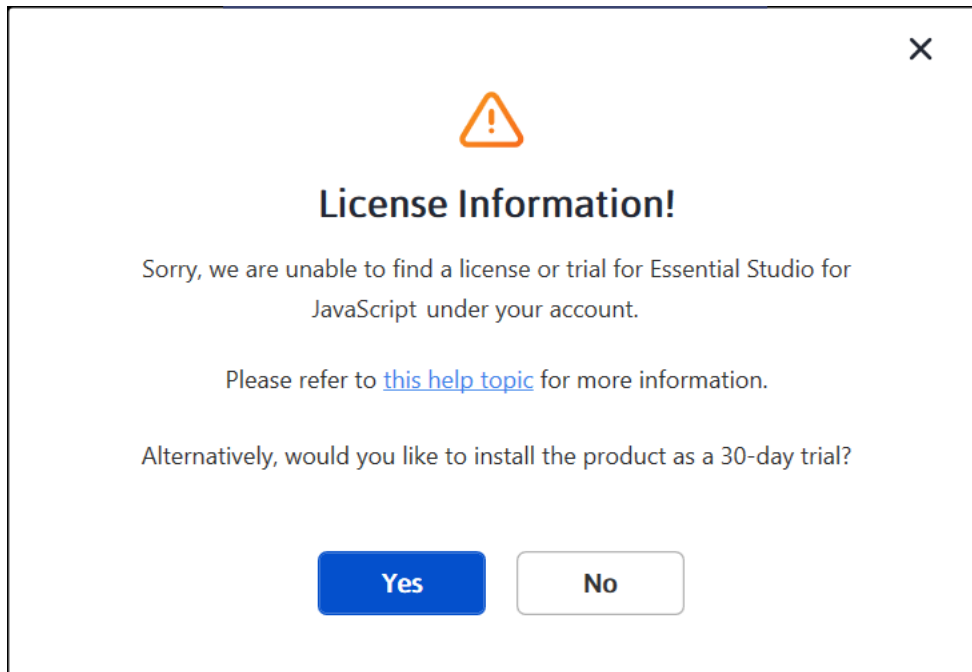
Unable to find a valid license or trial

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

Offline installer



Online installer



Reason
 The following are possible causes of this error:

The following are possible causes of this error:

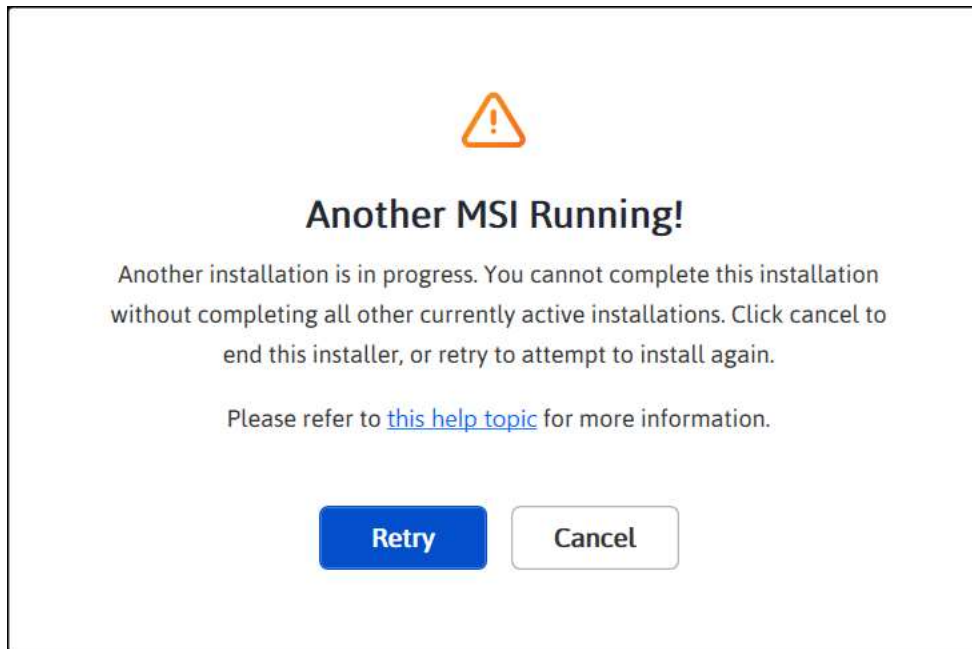
- When your trial period expired
- When you don't have a license or an active trial
- You are not the license holder of your license
- Your account administrator has not yet assigned you a license.

Suggested solution

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

Unable to Install because of Another Installation

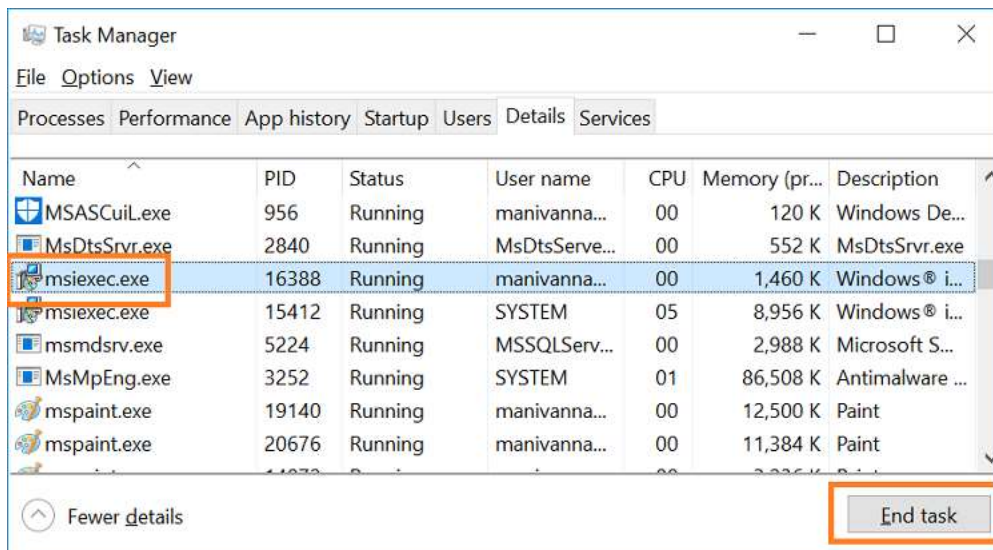
Error Message: Another installation is in progress. You cannot start this installation without completing all other currently active installations. Click cancel to end this installer or retry to attempt after currently active installation completed to install again.



Reason
 You are trying to install when another installation is already running in your machine.

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

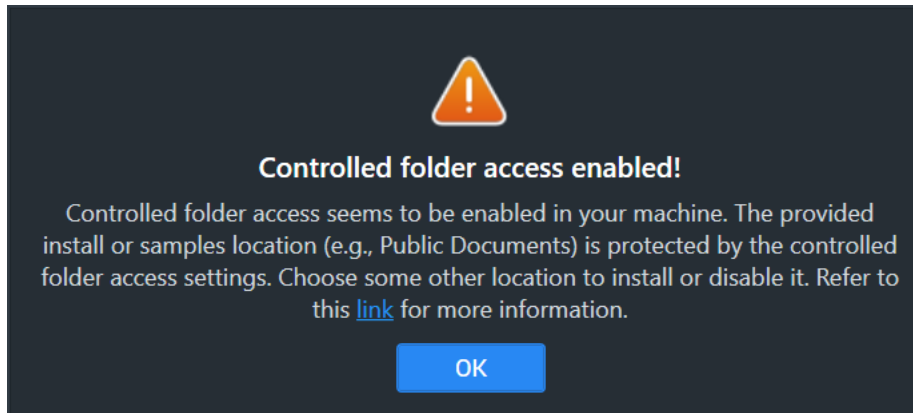
1. Open the Windows Task Manager.
2. Browse the Details tab.
3. Select the msixec.exe and click **End task**.



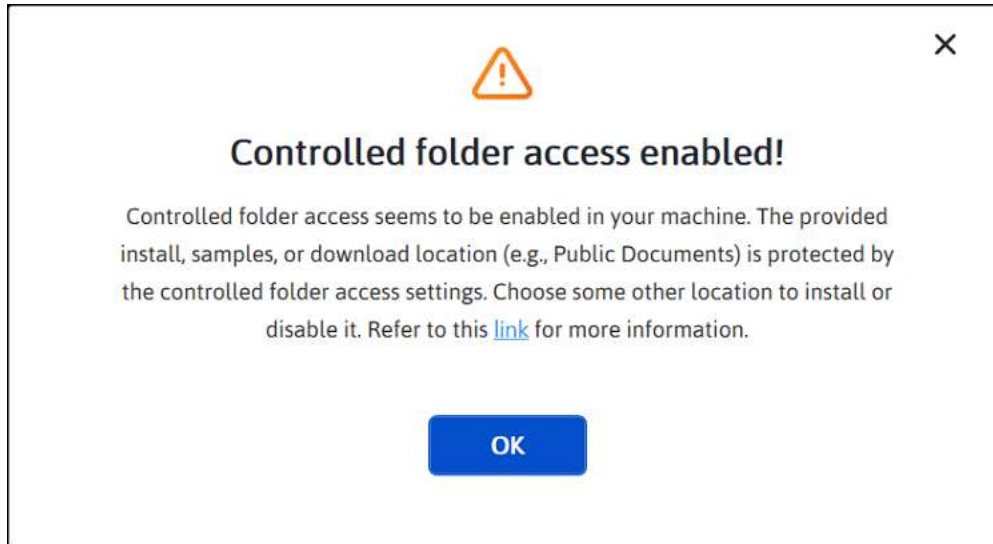
Unable to Install due to Controlled Folder Access

Offline

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

**Online**

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



Reason
 You have enabled controlled folder access settings on your computer.

Suggested solution**Suggestion 1:**

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

Suggestion 2:

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

Licensing

Syncfusion Licensing Overview

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

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

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

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

Privacy Assurance

The Syncfusion license does not store any user-specific information or details related to the user's company. The license key provided contains solely the encrypted values necessary for Syncfusion component validation. This ensures the privacy and security of user data while enabling effective validation of the components.

It's crucial to emphasize that the encrypted license key undergoes validation entirely on the client side, eliminating the need for network connections or HTTP requests. This approach enhances privacy, as Syncfusion neither accesses personal data nor collects information about users or devices. Whether working on a local development machine or within a CI/CD system, the license key validation process remains consistent, offering a secure and efficient experience without any data transmission to Syncfusion servers or third-party entities.

Difference between unlock key and license key

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

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

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

Registering Syncfusion license keys in Build server

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

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

| **NuGet package** | If the Syncfusion assemblies used in Build Server were from the Syncfusion NuGet packages, then no need to install any Syncfusion installer. We can directly use the required Syncfusion NuGet packages at [nuget.org](#).

But, if using NuGet packages from the [nuget.org](#), then we should

register the Syncfusion license key in the application. | Yes | Use any developer license to [generate](#) keys for Build Environments as well. |

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

| **Licensed installer** | If the Syncfusion assemblies used in Build Server were from Licensed Installer, then there is no need to register the license keys.

 You can [download](#) and [install](#) the licensed version of our installer. | No | Not applicable |

The license verification process implemented by Syncfusion has been designed to seamlessly integrate with your app's functionality without any adverse impact on its performance. Although keys are permitted to be included in the application source code/bundle, it is imperative to refrain from actively promoting, publishing, or distributing license keys. Any attempt to disseminate license key information with the intention of circumventing licensing requirements is strongly discouraged.

See Also

- [How to Generate Syncfusion EJ2-Angular License Key?](#)
- [How to Register Syncfusion License Key in EJ2-Angular Application?](#)
- [Licensing FAQ](#)

Generate Syncfusion EJ2-Angular License key

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



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

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

Claim License key

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

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

Active License

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

Claim License Key

Version : 23.1.36

Essential Studio Enterprise Edition v23.1.36

COPY TO CLIPBOARD

SEND EMAIL

For more details about this generated key, [click here](#).

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

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

Active Trial

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

Claim License Key

Version : 23.1.36

Essential Studio Enterprise Edition Trial v23.1.36

COPY TO CLIPBOARD

SEND EMAIL

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

Your license key expires on November 23, 2023.

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

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

Purchase Essential Studio

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

BUY NOW

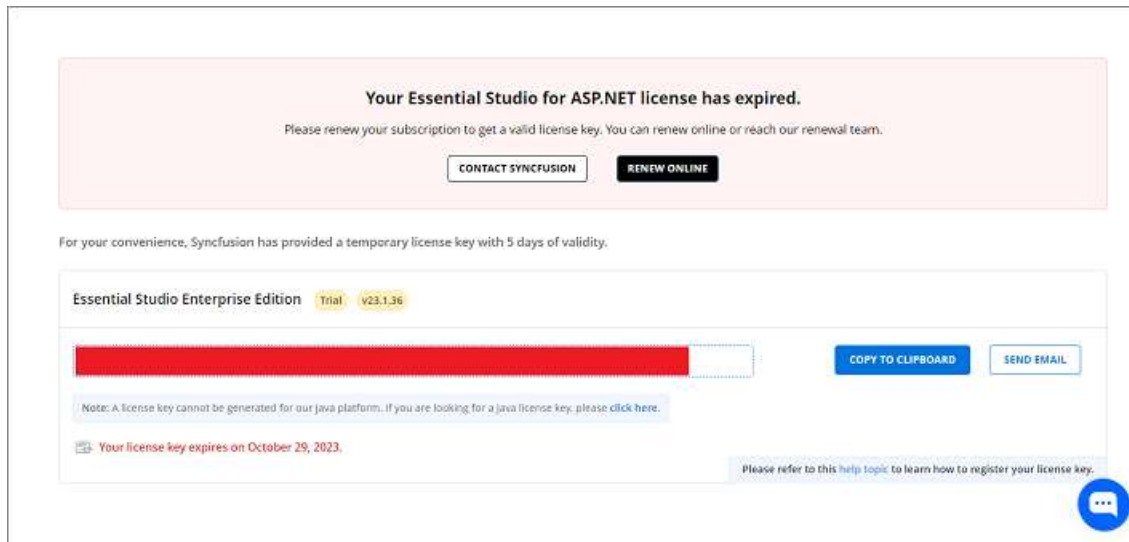
Free Community License

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

CLAIM YOUR FREE LICENSE

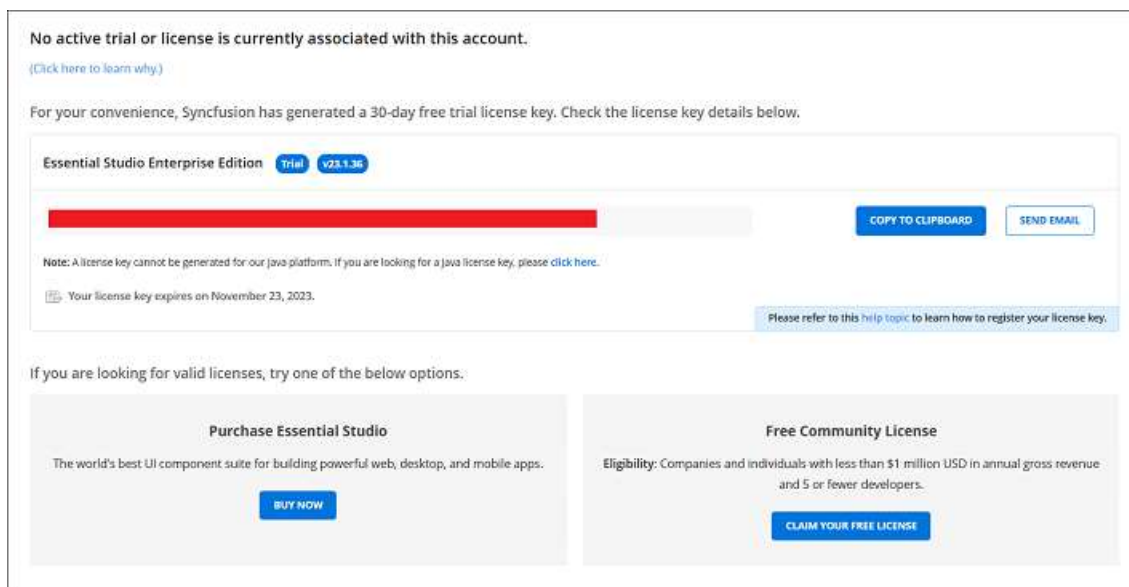
Expired License

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



No Trial or No License or Expired trial

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



See Also

- [How to Register Syncfusion License Key in the Application?](#)
- [Licensing FAQ](#)

Register Syncfusion License key in EJ2-Angular application

Syncfusion license key should be registered, if your project using Syncfusion EJ2-Angular packages reference. The generated license key is a string that needs to be registered after any [Syncfusion Angular reference](#).

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

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

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

Register Syncfusion License key in the project

Register the license key in the `main.ts` file of the Angular project.

```
`typescript
import { enableProdMode } from '@angular/core';
import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';
import { AppModule } from './app/app.module';
import { environment } from './environments/environment';
import { registerLicense } from '@syncfusion/ej2-base';
// Registering Syncfusion license key
registerLicense('Replace your generated license key here');
if (environment.production) {
  enableProdMode();
}
platformBrowserDynamic().bootstrapModule(AppModule)
.catch(err => console.error(err));
`
```

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

Register Syncfusion License key using the npx command

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

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

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

Register the license key with the license file

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

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

```
`sh
```

```
npx syncfusion-license activate
```

```
`
```

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

License message: `
` (INFO) Syncfusion License imported successfully.

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

If you don't want to use the license text file in the application, refer to this [link](#) to use an environment variable and register the Syncfusion license key. Also, check out some common licensing FAQs while registering the license key using the npx command from this [link](#)

Register the license key with the environment variable

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

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

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

Windows

- Open the command prompt and use [setx](#) command to add the new environment variable.

```
`sh
```

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

```
`
```

Mac

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

```
`sh
```

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

```
`
```

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

```
`sh
```

```
nano .bash_profile
```

```
,
```

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

Linux

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

```
`sh
```

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

```
,
```

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

```
`sh
```

```
npx syncfusion-license activate
```

```
,
```

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

License message:
 (INFO) Syncfusion License imported successfully.

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

Register Syncfusion license key in CI services

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

GitHub Actions

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


```
`bash
```

```
steps:
```

- name: Install node modules

```
run: npm install
```

- name: Activate Syncfusion License

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

```
env:
```

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

```
,
```

Azure Pipelines (YAML)

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

The following example shows the syntax for Windows build agents.

```
`bash
```

```
pool:
```

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

```
steps:
```

- script: call npm install

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

- script: call npx syncfusion-license activate

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

```
env:
```

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

```
,
```

The following example shows the syntax for Linux build agents.

```
`bash
```

```
pool:
```

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

```
steps:
```

- script: npm install

displayName: 'Install node modules'

- script: npx syncfusion-license activate

displayName: 'Activate Syncfusion License'

env:

SYNCFUSIONLICENSE: \$(SYNCFUSIONLICENSE)

,

Azure Pipelines (Classic)

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

`bash

Activate the license

npx syncfusion-license activate

,

← Bash ⓘ

Type ⓘ

☐ File Path

☒ Inline

Script *

Activate the license
npx syncfusion-license activate

Advanced ▼

About this task

Add

See Also

- [Licensing FAQ](#)

Syncfusion Licensing Errors

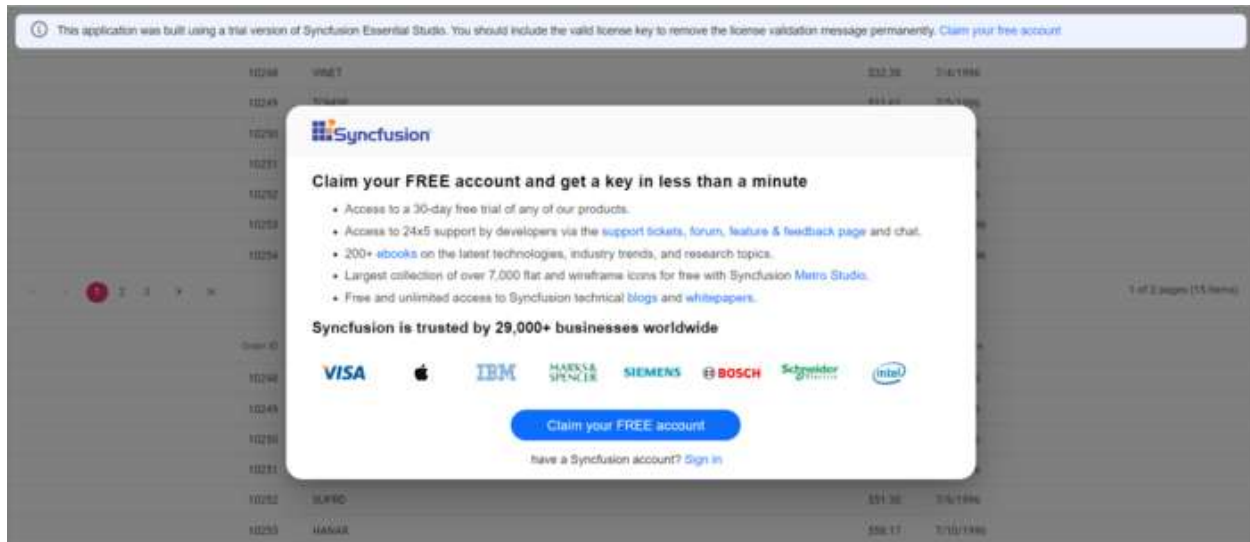
Licensing error popup is displayed with various messages under different circumstances. Here are some ways to resolve different issues.

Licensing errors

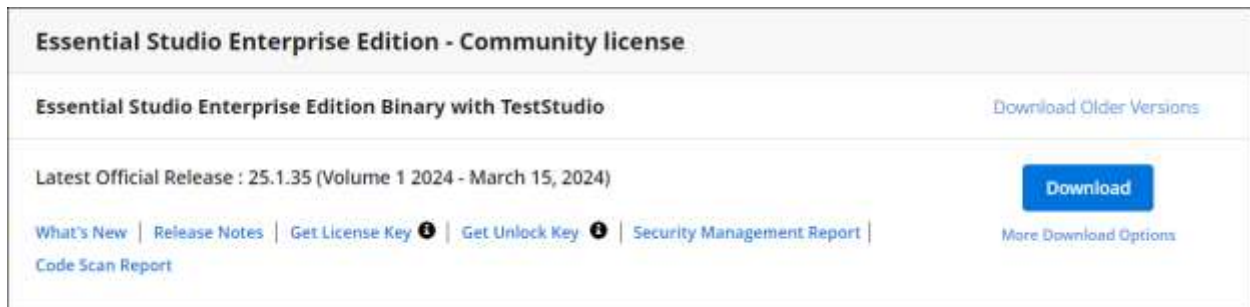
License key not registered\Trial Expired

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

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

**Solution:**

- If you use EJ2-Angular components through syncfusion installer, you can choose from the options listed below
 1. If you **have a valid Syncfusion license**, you can **generate a license key for a specific version and product** from [this page](#).

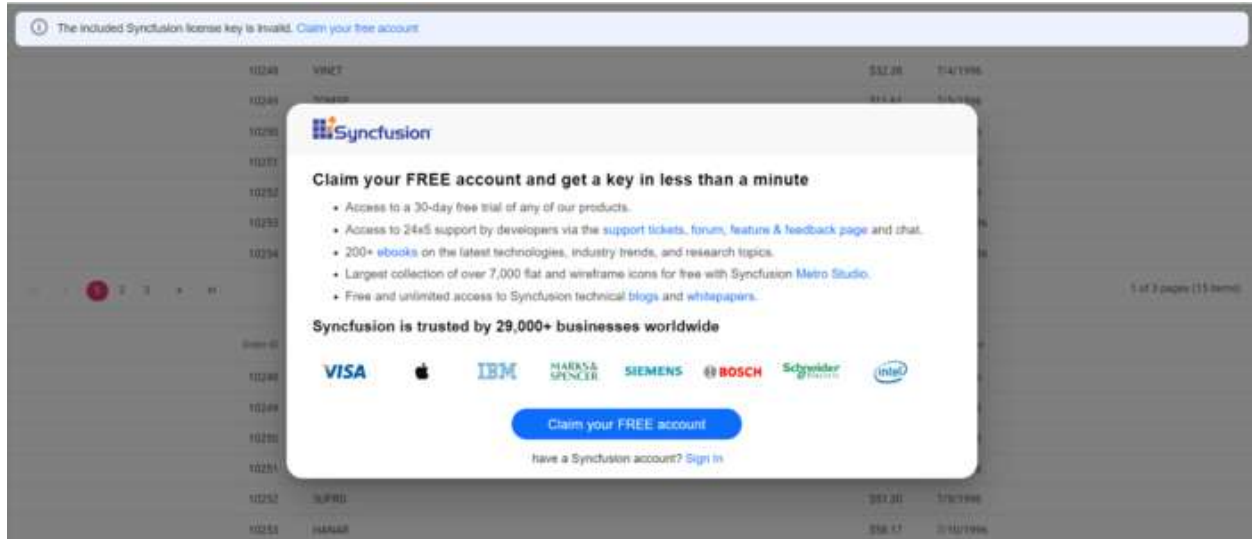


2. If you **have a Syncfusion account and an active trial**, you can **generate the trial license key for a specific version and platform** from [this page](#).
3. If you **have a Syncfusion account but no active trials**, [purchase a license](#) or [start your 30-day free trial](#). Then you can generate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

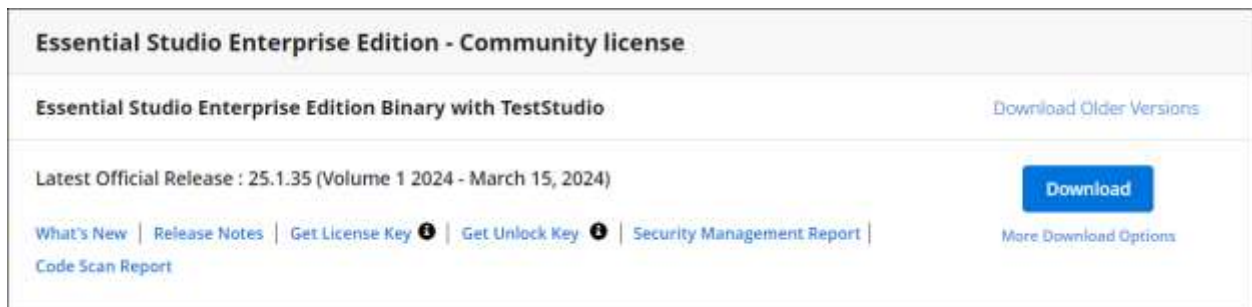
Invalid key

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

Error Message: The included Syncfusion license key is invalid.

**Solution:**

- If you use EJ2-Angular components through syncfusion installer, you can choose from the options listed below
 1. If you have a valid Syncfusion license, you can **generate a license key for a specific version and product** from [this page](#).



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

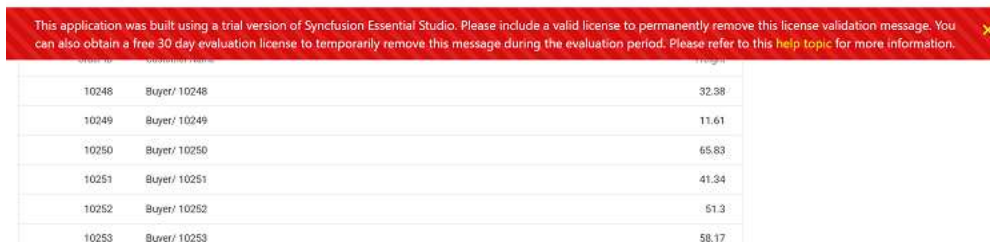
- In your application, register the generated license key. Please refer to this [help topic](#) for information on registering the license key.

Licensing errors from version 16.2.0* to 20.3.0*

License key not registered

The following error message will be shown if a Syncfusion license key has not been registered in your application.

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



Solution:

- If you use EJ2-Angular components through syncfusion installer, you can choose from the options listed below
 1. If you **have a valid Syncfusion license**, you can **generate a license key for a specific version and product** from [this page](#).



2. If you **have a Syncfusion account and an active trial**, you can **generate the trial license key for a specific version and platform** from [this page](#).
3. If you **have a Syncfusion account but no active trials**, [purchase a license](#) or [start your 30-day free trial](#). Then you can generate the trial license key for **a specific version and platform** from [this page](#).
4. If you **do not already have a Syncfusion account**, you can create one [here](#) and [purchase a license](#) or start your 30-day free trial. Then you can **generate the trial license key for a specific version and platform** from [this page](#).
 - In your application, register the generated license key. Please refer to this [help topic](#) for information on registering the license key.

Invalid key

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

Error message:
 The included Syncfusion license is invalid. Please refer to this [help topic](#) for more information.



Order ID	Customer Name	Freight
10248	Buyer/ 10248	32.38
10249	Buyer/ 10249	11.61
10250	Buyer/ 10250	65.83
10251	Buyer/ 10251	41.34
10252	Buyer/ 10252	51.3
10253	Buyer/ 10253	58.17

Solution:

- If you use EJ2-Angular components through syncfusion installer, you can choose from the options listed below
 - If you have a valid Syncfusion license, you can **generate a license key for a specific version and product** from [this page](#).



- If you have a Syncfusion account and an active trial, you can **generate the trial license key for a specific version and product** from [this page](#).
- If you **have a Syncfusion account but no active trials**, [purchase a license](#) or [start your 30-day free trial](#). Then you can **generate the trial license key for a specific version and product** from [this page](#).
- If you **do not already have a Syncfusion account**, you can create one here and [purchase a license](#) or [start your 30-day free trial](#). Then you can **generate the trial license key for a specific version and product** from [this page](#).
 - In your application, register the generated license key. Please refer to this [help topic](#) for information on registering the license key.

Trial expired

The following error message will be shown if the trial key has expired after 30 days.

Error message:
 Your Syncfusion trial license has expired. Please refer to this [help topic](#) for more information.

Your Syncfusion trial license has expired. Please refer to this help topic for more information.		
Order ID	Customer Name	Freight
10248	Buyer/ 10248	32.38
10249	Buyer/ 10249	11.61
10250	Buyer/ 10250	65.83
10251	Buyer/ 10251	41.94
10252	Buyer/ 10252	51.3
10253	Buyer/ 10253	58.17

Solution: Purchase from [here](#) to get a valid Syncfusion license.

Platform Mismatch

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

Error message: The included Syncfusion license is invalid (Platform mismatch). Please refer to this [help topic](#) for more information.

The included Syncfusion license is invalid (Platform mismatch). Please refer to this help topic for more information.

Solution:

- License keys are version and product specific. So, if you use EJ2 Angular components through syncfusion installer, you can choose from the options listed below
 - If you have a valid Syncfusion license, you can **generate a license key for a specific version and product** from [this page](#).

Essential Studio Enterprise Edition - Community license

Essential Studio Enterprise Edition Binary with TestStudio [Download Older Versions](#)

Latest Official Release : 25.1.35 (Volume 1 2024 - March 15, 2024)

[What's New](#) |
 [Release Notes](#) |
 [Get License Key](#) |
 [Get Unlock Key](#) |
 [Security Management Report](#) |
 [Code Scan Report](#)

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

- If you have a Syncfusion account and an active trial, you can **generate the trial license key for a specific version and product** from [this page](#).
- If you **have a Syncfusion account but no active trials**, [purchase a license](#) or [start your 30-day free trial](#). Then you can **generate the trial license key for a specific version and product** from [this page](#).
 - In your application, register the generated license key. Please refer to this [help topic](#) for information on registering the license key.

Version Mismatch

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

Error message: The included Syncfusion license ({Registered Version}) is invalid for version {Required version}. Please refer to this [help topic](#) for more information.

The included Syncfusion license (v19.1.0.44) is invalid for version 20.1.x. Please refer to this help topic for more information.		
Order ID	Customer Name	Freight
10248	Buyer/ 10248	32.38
10249	Buyer/ 10249	11.61
10250	Buyer/ 10250	65.83
10251	Buyer/ 10251	41.34
10252	Buyer/ 10252	51.3

Solution:

- License keys are version and product specific. So, if you use EJ2 Angular components through syncfusion installer, you can choose from the options listed below
 - If you have a valid Syncfusion license, you can **generate a license key for a specific version and product** from [this page](#).

Essential Studio Enterprise Edition - Community license

Essential Studio Enterprise Edition Binary with TestStudio [Download Older Versions](#)

Latest Official Release : 25.1.35 (Volume 1 2024 - March 15, 2024)

[What's New](#) |
 [Release Notes](#) |
 [Get License Key](#) ⓘ |
 [Get Unlock Key](#) ⓘ |
 [Security Management Report](#) |
 [Code Scan Report](#)

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

- If you have a Syncfusion account and an active trial, you can **generate the trial license key for a specific version and product** from [this page](#).
- If you **have a Syncfusion account but no active trials**, [purchase a license](#) or [start your 30-day free trial](#). Then you can **generate the trial license key for a specific version and product** from [this page](#).
 - In your application, register the generated license key. Please refer to this [help topic](#) for information on registering the license key.

Invalid license key structure using the npx command

If you are using `npx syncfusion-license activate` command with an invalid license key structure, the following console error message will appear.

Error message: `
 (Error) License key is not valid. Please refer to this help topic for more information.`

Solution:

- If you use EJ2-Angular components through syncfusion installer, you can choose from the options listed below
 - If you have a valid Syncfusion license, you can **generate a license key for a specific version and product** from [this page](#).



2. If you have a Syncfusion account and an active trial, you can **generate the trial license key for a specific version and product** from [this page](#).
3. If you **have a Syncfusion account but no active trials**, [purchase a license](#) or [start your 30-day free trial](#). Then you can **generate the trial license key for a specific version and product** from [this page](#).
4. If you **do not already have a Syncfusion account**, you can create one here and [purchase a license](#) or [start your 30-day free trial](#). Then you can **generate the trial license key for a specific version and product** from [this page](#).
 - In your application, register the generated license key using the npx command. Please refer to this [help topic](#) for information on registering the license key.

Licensing troubleshoot in Angular

Is an internet connection required for license validation

No, Internet connection is not required for the Syncfusion Essential Studio license validation. The Syncfusion license validation is done offline during application execution. Apps registered with a Syncfusion license key can be deployed on any system that does not have an internet connection.

Upgrade from the trial version after purchasing a license

To upgrade from the trial version, there are two possible solutions:

- Uninstall the trial version and install the fully licensed build from the [License & Downloads](#) section of the Syncfusion website.
- If you are using Syncfusion controls from the [npm](#), replace the currently used trial license key with a paid license key that can be generated from the [License & Downloads](#) section of Syncfusion website. Refer to [this](#) topic for more information regarding registering the license in the application.

The license registration is not required if you reference Syncfusion scripts from the Licensed installer. These licensing changes apply to all evaluators who refer to the Syncfusion scripts from the evaluation installer and those who use the Syncfusion NuGet packages from [nuget.org](#).

Where can I get a License key

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



The Syncfusion license keys are the **version and platform-specific**, refer to the [KB](#) to generate the license key for the required version and platform. Also, refer to this [KB](#) to know which version of the Syncfusion license key should be used in the application.

While using the ASP.NET Core controls with the Javascript(ES5) components, you need to register the license key in both the Javascript(ES5) and the [ASP.NET core](#). Since the license is validated at the client side for Javascript(ES5) components and server-side for the ASP.NET core components.

[Will the registered license key expire](#)

No, the Syncfusion license keys won't expire for a particular version and you can continue to use it. So, you won't face any problems on the live site. If you have used the trial key, it will expire in 30 days and we don't recommend using it in production.

If you upgrade to newer versions of the Syncfusion packages, you have to generate new license keys and use them.

[When to generate new license key while upgrading](#)

You don't have to generate and change license keys for minor version upgrades. If you upgrade from one major version and another major version, you have to generate new license keys and register in the application.

For example,

- If you upgrade to weekly releases or the SP release in the same major version, you don't have to change license such as if you upgrade from the 20.1.47 version to 20.1.* you don't have to change the license keys.
- If you upgrade from one major version to another major version, you have to generate new license keys for the latest version and change in the application, such as if you upgrade from the 20.1. version to 20.2., you have to generate new license keys for the latest version and change in the application.

[License registration for multiple developers on your project](#)

Syncfusion license key is a version based and it's not based on the developer. You don't have to register different keys for each developer. Just register one valid license key when developing and publishing the software.

[Can I use the same key for all the web apps under the project](#)

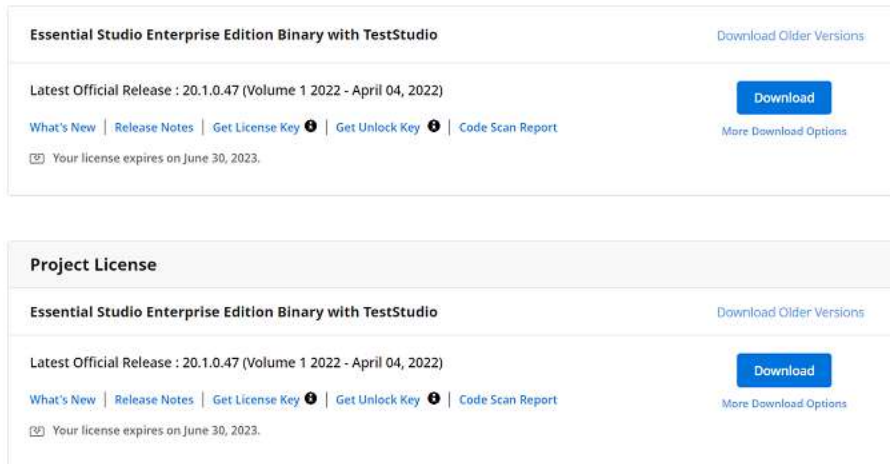
Yes, you can use the same license key for all the web apps.

[Does the license registration access any resources or data](#)

No, the license registration doesn't access any data or resources.

License & Downloads shows the "Essential Studio Enterprise Edition Binary with Test Studio" and the "Project License". Which license to use

Use any licenses shown on the [accounts & downloads](#) page. It shows two licenses because if you are part of your company's enterprise portal Global license and an individual license is also assigned to your account, on your account & downloads page, the individual license and your enterprise portal Global license are shown.



Refer to the [KB](#) article which explains the Licenses offered by Syncfusion.

If I registered the license key in both the application and the license text file
The application registered license key is set priority and used for license validation.

Reactivating license once after updating the package version while using npx

It is essential to reactivate the license key when upgrading the Syncfusion packages while the license has been registered through the `npx` command.

Potential causes of licensing errors in applications.

Below are the possible reasons that could lead to a license error within the application:

- The application may have a license issue due to duplicate Syncfusion packages.
- An invalid license issue may occur because of Syncfusion packages being referred with multiple versions.
- Registering the license key of a different version than the referred Syncfusion package version in the application can also cause licensing errors.
- Inclusion of [non-Angular](#) packages in the dependencies may lead to licensing errors due to the presence of duplicate instances of our packages.

License issue due to duplicate Syncfusion packages in the application

One of the possible cases on experiencing license issues in your application is due to duplicate packages exists after upgrading packages to next or latest version. To remove the duplicate packages follow the below steps.

- Delete the `@Syncfusion` folder from `node_modules` and `package-lock.json` file from app root folder.

- Clear the npm `.cache` by running the command `npm cache clean --force` or you can directly delete the file present in the application.
- It is recommended to update all Syncfusion components in the `package.json` file to the **same major version**. This ensures consistency and compatibility across the project. For instance, if the updated version being utilized is `v20.4.XX`, it is advised to upgrade all components to the **same version**.
- Run `npm install` Command.

Invalid license issue because of Syncfusion packages referred with multiple version

It is essential to ensure that all the components used in a project are compatible and work seamlessly together. One common issue that arises in such scenarios is **version mismatch**. Version mismatch occurs when **different components** have **different major versions**, leading to compatibility issues and difficulties in license registration.

For example, consider a situation where one component in the project has a version of `v20.1.XX`, while another component has a version of `v20.2.XX`. When such components are used together, a **version mismatch** occurs, leading to license errors. To avoid version mismatch and ensure smooth functioning of the project, it is crucial to use the **same major version** for all the Syncfusion components. This will ensure compatibility and prevent any licensing issues that may arise due to version incompatibility.

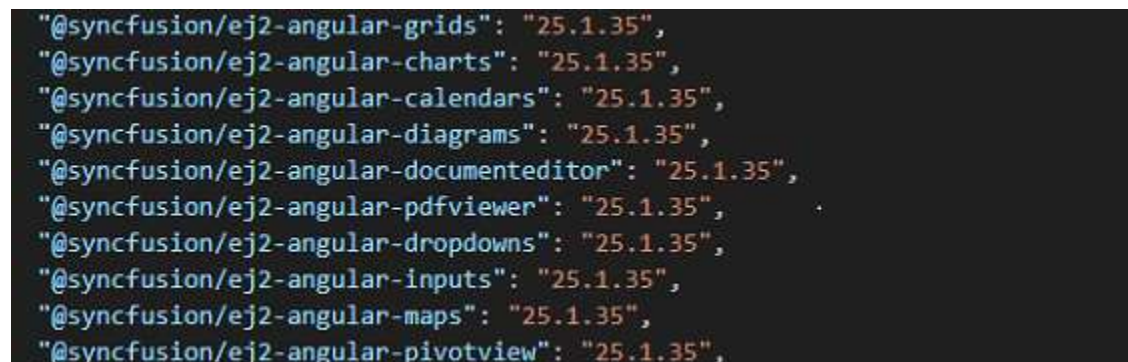
Registering the license key of a different version than the referred Syncfusion package version in the application

When developing an application with Syncfusion packages, it is important to register the appropriate license key that matches the version of the package installed. Failure to do so may result in license errors within the application.

For instance, if you are using a component version labeled as `(v20.4.XX)`, it is essential to register the license key generated **specifically** for that version. By doing so, it ensures the smooth functioning of the controls and provides access to all features and functionality without encountering any license validation errors.

License issue due to including non-Angular packages in the dependencies

When integrating Syncfusion with your Angular project, it's essential to include only our Angular component packages in the dependencies, as shown in the image below.



```
"@syncfusion/ej2-angular-grids": "25.1.35",
"@syncfusion/ej2-angular-charts": "25.1.35",
"@syncfusion/ej2-angular-calendars": "25.1.35",
"@syncfusion/ej2-angular-diagrams": "25.1.35",
"@syncfusion/ej2-angular-documenteditor": "25.1.35",
"@syncfusion/ej2-angular-pdfviewer": "25.1.35",
"@syncfusion/ej2-angular-dropdowns": "25.1.35",
"@syncfusion/ej2-angular-inputs": "25.1.35",
"@syncfusion/ej2-angular-maps": "25.1.35",
"@syncfusion/ej2-angular-pivotview": "25.1.35",
```

Avoid including our TypeScript packages separately.

```
"@syncfusion/ej2-base": "25.1.35",
"@syncfusion/ej2-icons": "25.1.35",
"@syncfusion/ej2-svg-base": "25.1.35",
"@syncfusion/ej2-pdf-export": "25.1.35",
"@syncfusion/ej2-excel-export": "25.1.35",
"@syncfusion/ej2-grids": "25.1.35",
"@syncfusion/ej2-charts": "25.1.35",
"@syncfusion/ej2-calendars": "25.1.35",
"@syncfusion/ej2-diagrams": "25.1.35",
"@syncfusion/ej2-documenteditor": "25.1.35",
"@syncfusion/ej2-pdfviewer": "25.1.35",
"@syncfusion/ej2-angular-grids": "25.1.35",
"@syncfusion/ej2-angular-charts": "25.1.35",
"@syncfusion/ej2-angular-calendars": "25.1.35",
"@syncfusion/ej2-angular-diagrams": "25.1.35",
"@syncfusion/ej2-angular-documenteditor": "25.1.35",
"@syncfusion/ej2-angular-pdfviewer": "25.1.35",
```

These are sub-dependencies of our Angular component packages and are automatically installed along with them. Including them separately may sometimes result in duplicate instances of packages, which can lead to issues with license validation. Therefore, to ensure proper license validation and avoid conflicts, stick to including our Angular component packages alone in your project dependencies.

Getting Started

Getting Started with Angular CLI

The Angular CLI is a command-line interface tool that allows developers to create, manage, and build Angular applications. This makes it easy to set up a new Angular project and get started with development.

In this guide, we will show you how to create an Angular project and install the Syncfusion Angular UI Components (Essential JS 2) to get started with development.

Prerequisites

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

Setting up an Angular project

Using [Angular CLI](#), you can easily setup Angular projects. This section provides guidance for installing a specific version of Angular CLI and creating an Angular 17 application based on your requirements.

Installing a Specific Version

To install a specific version of Angular CLI, use the following command,

```
`bash
```

```
npm install -g @angular/cli@16.0.1
```

```
`
```

Installing the Latest Angular CLI

To create an Angular 17 application, use the Angular CLI with the following command


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

Note: For Angular 17, it default for generates a standalone application. Detailed instructions on creating Syncfusion Angular standalone components using the latest version, please refer to the [Standalone guide](#).

Create a new application

Once the Angular CLI is installed, execute the following command to create a new project,

```
`bash
ng new syncfusion-angular-app
`
```

This command will prompt you for a few settings for the new project, such as whether to add Angular routing and which stylesheet format to use.

```
D:\Sample>ng new syncfusion-angular-app
? Would you like to add Angular routing? No
? Which stylesheet format would you like to use? (Use arrow keys)
> CSS
  SCSS [ https://sass-lang.com/documentation/syntax#scss ]
  Sass [ https://sass-lang.com/documentation/syntax#the-indented-syntax ]
  Less [ http://lesscss.org ]
```

By default, it will create a CSS-based application. You can specify that you want to use SCSS by running the following command instead:

```
`bash
ng new syncfusion-angular-app --style=scss
`
```

Next, navigate to the created project folder:

```
`bash
cd syncfusion-angular-app
`
```

Installing Syncfusion Angular packages

Syncfusion packages are distributed in npm under the `@syncfusion` scope. You can obtain all of the available Angular Syncfusion packages from [npm](#).

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

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

Syncfusion's latest Angular packages are Ivy-distributed and compatible with Angular 12 and above. To install the package use the following command,

```
`bash
ng add @syncfusion/ej2-angular-grids
`
```

If you are not using fully ivy compiler application, use the `ngcc` tagged packages of the Syncfusion Angular components.

Note: The `ngcc` packages are still compatible with Angular CLI versions 15 and below. However, they may generate warnings suggesting the use of IVY compiled packages. Starting from Angular 16, support for the `ngcc` package has been completely removed. If you have further questions regarding `ngcc` compatibility, please refer to the following [FAQ](#).

```
`bash
npm add @syncfusion/ej2-angular-grids@20.2.38-ngcc
`
```

The above command does the following configuration to your Angular app,

- Adds `@syncfusion/ej2-angular-grids` package and its peer dependencies to your `package.json` file.
- Imports the `GridModule` in your application module `app.module.ts`.
- Registers the Syncfusion UI default theme (material) in the `angular.json` file.

This makes it easy to add the Syncfusion Angular Grids module to your project and start using it in your application.

For more information about version compatibility, see [version compatibility](#).

Adding Syncfusion Angular components

To use Syncfusion Angular components in your application, you will need to add them to your template and specify their properties in your component class.

In `src/app/app.component.ts`, you can use column directives with the `<ejs-grid>` selector and specify the `e-column` elements inside the `<ejs-grid>` element in the template for your component. The `e-column` element allows you to define the properties of a column in the Grid, such as its field name, header text, and data type.

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

@Component({
  selector: 'app-root',
  template: `
    <h1>
    Syncfusion Angular UI Grid!
    </h1>
    <ejs-grid [dataSource]='data'>
```



```

<e-columns>
<e-column field='OrderID' headerText='Order ID' textAlign='Right' width=90></e-column>
<e-column field='CustomerID' headerText='Customer ID' width=120></e-column>
<e-column field='Freight' headerText='Freight' textAlign='Right' format='C2' width=90></e-column>
<e-column field='OrderDate' headerText='Order Date' textAlign='Right' format='yMd' width=120></e-
column>
</e-columns>
</ejs-grid>
`

})
export class AppComponent {
  public data: Object[] = [
    {
      OrderID: 10248, CustomerID: 'VINET', EmployeeID: 5, OrderDate: new Date(8364186e5),
      ShipName: 'Vins et alcools Chevalier', ShipCity: 'Reims', ShipAddress: '59 rue de l Abbaye',
      ShipRegion: 'CJ', ShipPostalCode: '51100', ShipCountry: 'France', Freight: 32.38, Verified: !0
    },
    {
      OrderID: 10249, CustomerID: 'TOMSP', EmployeeID: 6, OrderDate: new Date(836505e6),
      ShipName: 'Toms Spezialitäten', ShipCity: 'Münster', ShipAddress: 'Luisenstr. 48',
      ShipRegion: 'CJ', ShipPostalCode: '44087', ShipCountry: 'Germany', Freight: 11.61, Verified: !1
    },
    {
      OrderID: 10250, CustomerID: 'HANAR', EmployeeID: 4, OrderDate: new Date(8367642e5),
      ShipName: 'Hanari Carnes', ShipCity: 'Rio de Janeiro', ShipAddress: 'Rua do Paço, 67',
      ShipRegion: 'RJ', ShipPostalCode: '05454-876', ShipCountry: 'Brazil', Freight: 65.83, Verified: !0
    }
  ];
}
`

```

This will add a Grid to your application with the specified columns and data.

Run the application

Run the `ng serve` command in the console, it will serve your application and you can open the browser window.

Syncfusion Angular UI Grid!

Order ID	Customer ID	Freight	Order Date
10248	VINET	\$32.38	7/4/1996
10249	TOMSP	\$11.61	7/5/1996
10250	HANAR	\$65.83	7/8/1996

Refer the below sample for more information.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { GridAllModule } from '@syncfusion/ej2-angular-grids'
import { enableRipple } from '@syncfusion/ej2-base';
import { Component } from '@angular/core';
// enable ripple effects
enableRipple(true);
@Component({
  imports: [

    ButtonModule,
    GridAllModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `
    <h1>
      Syncfusion Angular UI Grid!
    </h1>
    <ejs-grid [dataSource]='data'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID' textAlign='Right'
width=90></e-column>
        <e-column field='CustomerID' headerText='Customer ID' width=120></e-
column>
        <e-column field='Freight' headerText='Freight' textAlign='Right'
format='C2' width=90></e-column>
        <e-column field='OrderDate' headerText='Order Date' textAlign='Right'
format='yMd' width=120></e-column>
      </e-columns>
    </ejs-grid>
  `
})
export class AppComponent {
  public data: Object[] = [
    {
```

```

    OrderID: 10248, CustomerID: 'VINET', EmployeeID: 5, OrderDate: new
    Date(8364186e5),
    ShipName: 'Vins et alcools Chevalier', ShipCity: 'Reims', ShipAddress:
    '59 rue de l Abbaye',
    ShipRegion: 'CJ', ShipPostalCode: '51100', ShipCountry: 'France',
    Freight: 32.38, Verified: !0
  },
  {
    OrderID: 10249, CustomerID: 'TOMSP', EmployeeID: 6, OrderDate: new
    Date(836505e6),
    ShipName: 'Toms Spezialitäten', ShipCity: 'Münster', ShipAddress:
    'Luisenstr. 48',
    ShipRegion: 'CJ', ShipPostalCode: '44087', ShipCountry: 'Germany',
    Freight: 11.61, Verified: !1
  },
  {
    OrderID: 10250, CustomerID: 'HANAR', EmployeeID: 4, OrderDate: new
    Date(8367642e5),
    ShipName: 'Hanari Carnes', ShipCity: 'Rio de Janeiro', ShipAddress:
    'Rua do Paço, 67',
    ShipRegion: 'RJ', ShipPostalCode: '05454-876', ShipCountry: 'Brazil',
    Freight: 65.83, Verified: !0
  }
];
}

```

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 see a license banner when running your application, it means that you need to obtain a license key and register it within the application in order to use Syncfusion components. You can find more information on how to obtain and register a license key on our [Licensing overviewLink to the Video](#) page.

You can also refer below video to get start Syncfusion Angular Grid component.

Syncfusion components-based styles

By default, the **Material** theme is registered in the `styles.css` file when you run the `ng add` command. However, Syncfusion Angular components offer a range of built-in [themes](#) that you can easily add to your project by importing the relevant theme.

The default **Material** theme includes styles for all Syncfusion Angular components. If you only want to use the styles for specific Syncfusion components, you can import only the required dependencies. For example, to use the styles for the Grid component alone, you can import the required dependencies as shown in the following snippet,

```

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

```

```
@import '../node_modules/@syncfusion/ej2-calendars/styles/material.css';
@import '../node_modules/@syncfusion/ej2-dropdowns/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-navigations/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-angular-grids/styles/material.css';
`
```

For information on using SCSS styles, see [here](#).

Adding feature Modules to Syncfusion Angular components

Syncfusion Angular components offer module-based services and are easy to import and extend the functionalities of the components. If you want to enable features such as paging, filtering, and sorting in a Grid that has been rendered in your application, you will need to include the corresponding service modules in the `providers` array of your modules.

```
`typescript
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { GridModule } from '@syncfusion/ej2-angular-grids';
// Imports Grid services from Syncfusion Angular Grids
import { PageService, FilterService, SortService } from '@syncfusion/ej2-angular-grids';
import { AppComponent } from './app.component';
/

  • Module

*/
@NgModule({
  imports: [
    BrowserModule,
    GridModule
  ],
  declarations: [AppComponent],
  bootstrap: [AppComponent],
  // Add feature Service in the provider
  providers: [PageService,
```

```
SortService,
FilterService]
})
export class AppModule { }
`
```

The component has several options set on the `ejs-grid` element, including `allowPaging`, `allowSorting`, and `allowFiltering`, which enable paging, sorting, and filtering features in the Grid, respectively. The `pageSettings` property is also set to specify the page size for the Grid. you can use the following code snippet in the `app.component.ts` file,

```
`typescript
import { Component, OnInit } from '@angular/core';
import { data } from './datasource';
import { PageSettingsModel } from '@syncfusion/ej2-angular-grids';
@Component({
  selector: 'app-root',
  template: `<ejs-grid
[dataSource]='data'
[allowPaging]="true"
[allowSorting]="true"
[allowFiltering]="true"
[pageSettings]="pageSettings">
<e-columns>
<e-column field='OrderID' headerText='Order ID' textAlign='Right' width=90></e-column>
<e-column field='CustomerID' headerText='Customer ID' width=120></e-column>
<e-column field='Freight' headerText='Freight' textAlign='Right' format='C2' width=90></e-column>
<e-column field='OrderDate' headerText='Order Date' textAlign='Right' format='yMd' width=120></e-
column>
</e-columns>
</ejs-grid>`
})
export class AppComponent implements OnInit {
  public data: object[];
  public pageSettings: PageSettingsModel;
  ngOnInit(): void {
```

```

this.data = data;

// The pageSettings property is also set to specify the page size for the Grid
this.pageSettings = { pageSize: 6 };
}
}
`

```

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule } from '@syncfusion/ej2-angular-grids'
import { PageService, SortService, FilterService, GroupService } from
 '@syncfusion/ej2-angular-grids'
import { Component, OnInit } from '@angular/core';
import { data } from './datasource';
import { PageSettingsModel } from '@syncfusion/ej2-angular-grids';
@Component({
  imports: [

    GridModule
  ],
  providers: [PageService,
    SortService,
    FilterService,
    GroupService]
  standalone: true,
  selector: 'app-root',
  template: `<ejs-grid [dataSource]='data' [allowPaging]="true"
[allowSorting]="true"
    [allowFiltering]="true" [pageSettings]="pageSettings">
    <e-columns>
      <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=90></e-column>
      <e-column field='CustomerID' headerText='Customer ID'
width=120></e-column>
      <e-column field='Freight' headerText='Freight'
textAlign='Right' format='C2' width=90></e-column>
      <e-column field='OrderDate' headerText='Order Date'
textAlign='Right' format='yMd' width=120></e-column>
    </e-columns>
  </ejs-grid>`
})
export class AppComponent implements OnInit {
  public data?: object[];
  public pageSettings?: PageSettingsModel;
  ngOnInit(): void {
    this.data = data;
    this.pageSettings = { pageSize: 6 };
  }
}

```

MAIN.TS

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

By using the Syncfusion Angular Grid component, you can easily add a robust data Grid to your Angular application that supports paging and sorting of data. For more details refer [Syncfusion Angular Grid](#).

Syncfusion Angular components showcase samples

Syncfusion has a collection of sample applications that demonstrate the use of Syncfusion Angular UI components.

- [Expense Tracker](#)
- [Diagram Builder](#)
- [Web mail](#)
- [Appointment Planner](#)
- [Stock Chart](#)

See also

- [Upgradation Guide](#)
- [Upgradation Guide](#)

Getting Started with Angular Standalone component

Standalone components provide a simplified way to build Angular applications. Standalone components, directives, and pipes aim to streamline the authoring experience by reducing the need for NgModules. Existing applications can optionally and incrementally adopt the new standalone style without any breaking changes.

Create a new application

Once the Latest Angular CLI is installed, you can use it to create a new Angular project by running the following command:

```
`bash
```

```
ng new syncfusion-angular-app
```

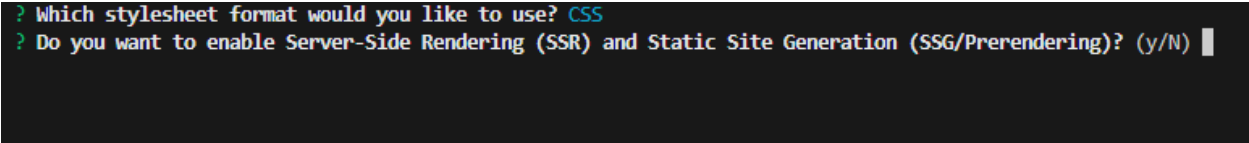
```
`
```

This command will prompt you for a few settings for the new project, such as which stylesheet format to use.

```
D:\Sample>ng new syncfusion-angular-app
? Would you like to add Angular routing? No
? Which stylesheet format would you like to use? (Use arrow keys)
> CSS
  SCSS [ https://sass-lang.com/documentation/syntax#scss ]
  Sass [ https://sass-lang.com/documentation/syntax#the-indented-syntax ]
  Less [ http://lesscss.org ]
```

By default, it will create a CSS-based application. You can specify that you want to use SCSS by running the following command instead:

```
`bash
ng new syncfusion-angular-app --style=scss
`
```



```
? Which stylesheet format would you like to use? CSS
? Do you want to enable Server-Side Rendering (SSR) and Static Site Generation (SSG/Prerendering)? (y/N)
```

Angular brought server-side rendering (SSR) and static-site generation (SSG or prerendering) closer to developers with a prompt in `ng new`. Also you can enable SSR in new projects with below command.

```
`bash
ng new syncfusion-angular-app --ssr
`
```

Next, navigate to the created project folder:

```
`bash
cd syncfusion-angular-app
`
```

Installing Syncfusion Angular packages

Syncfusion packages are distributed in npm under the `@syncfusion` scope. You can obtain all of the available Angular Syncfusion packages from [npm](#).

Syncfusion's latest Angular packages are Ivy-distributed and compatible with Angular 12 and above. To install the package use the following command,

```
`bash
ng add @syncfusion/ej2-angular-grids@latest
`
```

The above command does the following configuration to your Angular app,

- Adds `@syncfusion/ej2-angular-grids` package and its peer dependencies to your `package.json` file.
- Imports the `GridModule` in your application default standalone component `app.component.ts`.
- Registers the Syncfusion UI default theme (material) in the `angular.json` file.

This makes it easy to add the Syncfusion Angular Grids module to your project and start using it in your application.

Adding Syncfusion Angular components

To use Syncfusion Angular components in your application, you will need to add them to your template and specify their properties in your component class.

In `src/app/app.component.ts`, you can use column directives with the `<ejs-grid>` selector and specify the `e-column` elements inside the `<ejs-grid>` element in the template for your component. The `e-column` element allows you to define the properties of a column in the Grid, such as its field name, header text, and data type.

```
`typescript
import { Component } from '@angular/core';
import { GridModule, PagerModule } from '@syncfusion/ej2-angular-grids';
import { CommonModule } from '@angular/common';
import { RouterOutlet } from '@angular/router';

@Component({
  selector: 'app-root',
  standalone: true,
  imports: [GridModule, PagerModule, CommonModule, RouterOutlet],
  template: `
    <h1>
    Syncfusion Angular UI Grid!
    </h1>
    <ejs-grid [dataSource]='data'>
    <e-columns>
    <e-column field='OrderID' headerText='Order ID' textAlign='Right' width=90></e-column>
    <e-column field='CustomerID' headerText='Customer ID' width=120></e-column>
    <e-column field='Freight' headerText='Freight' textAlign='Right' format='C2' width=90></e-column>
    <e-column field='OrderDate' headerText='Order Date' textAlign='Right' format='yMd' width=120></e-
    column>
    </e-columns>
    </ejs-grid>
    `
})
export class AppComponent {
  public data: Object[] = [
    {
      OrderID: 10248, CustomerID: 'VINET', EmployeeID: 5, OrderDate: new Date(8364186e5),
      ShipName: 'Vins et alcools Chevalier', ShipCity: 'Reims', ShipAddress: '59 rue de l Abbaye',
      ShipRegion: 'CJ', ShipPostalCode: '51100', ShipCountry: 'France', Freight: 32.38, Verified: !0
    }
  ]
}
```

```

},
{
  OrderID: 10249, CustomerID: 'TOMSP', EmployeeID: 6, OrderDate: new Date(836505e6),
  ShipName: 'Toms Spezialitäten', ShipCity: 'Münster', ShipAddress: 'Luisenstr. 48',
  ShipRegion: 'CJ', ShipPostalCode: '44087', ShipCountry: 'Germany', Freight: 11.61, Verified: !1
},
{
  OrderID: 10250, CustomerID: 'HANAR', EmployeeID: 4, OrderDate: new Date(8367642e5),
  ShipName: 'Hanari Carnes', ShipCity: 'Rio de Janeiro', ShipAddress: 'Rua do Paço, 67',
  ShipRegion: 'RJ', ShipPostalCode: '05454-876', ShipCountry: 'Brazil', Freight: 65.83, Verified: !0
}
];
}
`

```

This will add a Grid to your application with the specified columns and data.

[Adding CSS reference](#)

The following CSS files are available in `../node_modules/@syncfusion` package folder.

This can be referenced in `[src/styles.css]` using following code.

```

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

```

[Run the application](#)

Run the `ng serve` command in the console, it will serve your application and you can open the browser window.

Syncfusion Angular UI Grid!

Order ID	Customer ID	Freight	Order Date
10248	VINET	\$32.38	7/4/1996
10249	TOMSP	\$11.61	7/5/1996
10250	HANAR	\$65.83	7/8/1996

Note: If you see a license banner when running your application, it means that you need to obtain a license key and register it within the application in order to use Syncfusion components. You can find more information on how to obtain and register a license key on our [Licensing overview](#) page.

Getting Started ASP.NET Core with Angular using Project Template

This document provides step-by-step instructions on how to create a simple ASP.NET Core application with the Angular Framework using the dotnet CLI and integrate with Syncfusion Angular UI components.

Prerequisites

Before getting started with Syncfusion Angular Components in an ASP.NET Core with Angular project, check whether the following are installed on the developer machine.

- [System requirements for Syncfusion Angular UI components](#)
- [.NET 7.0 SDK](#)

Create an application

Create an ASP.NET Core application with Angular using the dotnet CLI.

1. Open the command prompt at your preferred location, and run the following command to create an ASP.NET Core application with Angular.

CMD

```
dotnet new angular -o Syncfusion-ASP.NET-Core-Angular
```

2. Navigate to the application folder using the following command.

CMD

```
cd Syncfusion-ASP.NET-Core-Angular
```

3. Set an environment variable called `ASPNETCORE_ENVIRONMENT` with a value of `Development` on Windows, run the following command.

CMD

```
SET ASPNETCORE_ENVIRONMENT=Development
```

For Linux or macOS, run the `export ASPNETCORE_ENVIRONMENT=Development` in the terminal.

4. Now, run the build command to ensure the application builds correctly. During the initial run, the build process restores npm dependencies, which may take several minutes. Subsequent builds will be significantly faster.

CMD

```
dotnet build
```

5. The ASP.NET Core with angular project template is created successfully.

Installing Syncfusion Grid Package

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

Navigate to the ClientApp folder and install the Syncfusion Angular DataGrid package using the following commands.

CMD

```
cd ClientApp
npm install @syncfusion/ej2-angular-grids --save
```

Adding Grid Module

After installing the package, users can configure the component modules in your application from the Syncfusion installed package. Open the `~/src/app.module.ts` file in the `ClientApp` folder using Visual Studio Code or your preferred code editor, and refer to the following code snippet to import the Grid module.

APP.MODULE.TS

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { FormsModule } from '@angular/forms';
import { HttpClientModule } from '@angular/common/http';
import { RouterModule } from '@angular/router';
import { AppComponent } from './app.component';
import { NavMenuComponent } from './nav-menu/nav-menu.component';
import { HomeComponent } from './home/home.component';
import { CounterComponent } from './counter/counter.component';
import { FetchDataComponent } from './fetch-data/fetch-data.component';
// Imported Syncfusion grid module from grids package
import { GridModule } from '@syncfusion/ej2-angular-grids';
@NgModule({
  declarations: [
    AppComponent,
    NavMenuComponent,
    HomeComponent,
    CounterComponent,
    FetchDataComponent
  ],
  imports: [
    BrowserModule.withServerTransition({ appId: 'ng-cli-universal' }),
```

```

HttpClientModule,
FormsModule,
//Registering EJ2 grid module
GridModule,
RouterModule.forRoot([
  { path: '', component: HomeComponent, pathMatch: 'full' },
  { path: 'counter', component: CounterComponent },
  { path: 'fetch-data', component: FetchDataComponent },
])
],
providers: [],
bootstrap: [AppComponent]
})
export class AppModule { }

```

Adding Syncfusion Component

Add the following grid component code snippet in the `~/src/home/home.component.ts` file as follows.

HOME.COMPONENT.TS

```

import { Component } from '@angular/core';
@Component({
  selector: 'app-home',
  template: `
<h1>
Syncfusion Angular UI Grid!
</h1>
<ejs-grid [dataSource]='data'>
<e-columns>
<e-column field='OrderID' headerText='Order ID' textAlign='Right'
width=90></e-column>
<e-column field='CustomerID' headerText='Customer ID' width=120></e-column>
<e-column field='Freight' headerText='Freight' textAlign='Right' format='C2'
width=90></e-column>
<e-column field='OrderDate' headerText='Order Date' textAlign='Right'
format='yMd' width=120></e-column>
</e-columns>
</ejs-grid>
`
})
export class HomeComponent {
  public data: Object[] = [
    {
      OrderID: 10248, CustomerID: 'VINET', EmployeeID: 5, OrderDate: new
      Date(8364186e5),
      ShipName: 'Vins et alcools Chevalier', ShipCity: 'Reims', ShipAddress: '59
      rue de l Abbaye',
      ShipRegion: 'CJ', ShipPostalCode: '51100', ShipCountry: 'France', Freight:
      32.38, Verified: !0
    },
    {
      OrderID: 10249, CustomerID: 'TOMSP', EmployeeID: 6, OrderDate: new
      Date(836505e6),
      ShipName: 'Toms Spezialitäten', ShipCity: 'Münster', ShipAddress:
      'Luisenstr. 48',

```

```

ShipRegion: 'CJ', ShipPostalCode: '44087', ShipCountry: 'Germany', Freight:
11.61, Verified: !1
},
{
OrderID: 10250, CustomerID: 'HANAR', EmployeeID: 4, OrderDate: new
Date(8367642e5),
ShipName: 'Hanari Carnes', ShipCity: 'Rio de Janeiro', ShipAddress: 'Rua do
Paço, 67',
ShipRegion: 'RJ', ShipPostalCode: '05454-876', ShipCountry: 'Brazil',
Freight: 65.83, Verified: !0
}
];
}

```

Adding CSS reference

Syncfusion Angular components offer a variety of built-in [themes](#) that you can easily add to your project by importing the relevant css file from the `~/node_modules/@syncfusion` package. Add the following Grid component styles as shown in the `~/src/styles.css` file.

STYLES.CSS

```

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

```

Run the application

Navigate to the root directory, and execute the following commands to run the application.

CMD

```

cd ..
dotnet run

```

Syncfusion Angular UI Grid!

Order ID	Customer ID	Freight	Order Date
10248	VINET	\$32.38	7/4/1996
10249	TOMSP	\$11.61	7/5/1996
10250	HANAR	\$65.83	7/8/1996

Note: [View the Angular Sample with ASP.NET Core on GitHub](#)

See also

- [How to create an ASP.NET Core with Angular project using the visual studio](#)
- [Getting started with angular CLI](#)

Getting Started with Angular CLI as Front end in ASP.NET MVC

This document explains how to create an ASP.NET MVC framework with an Angular CLI project as the front end and integrate Syncfusion Angular UI components.

Prerequisites

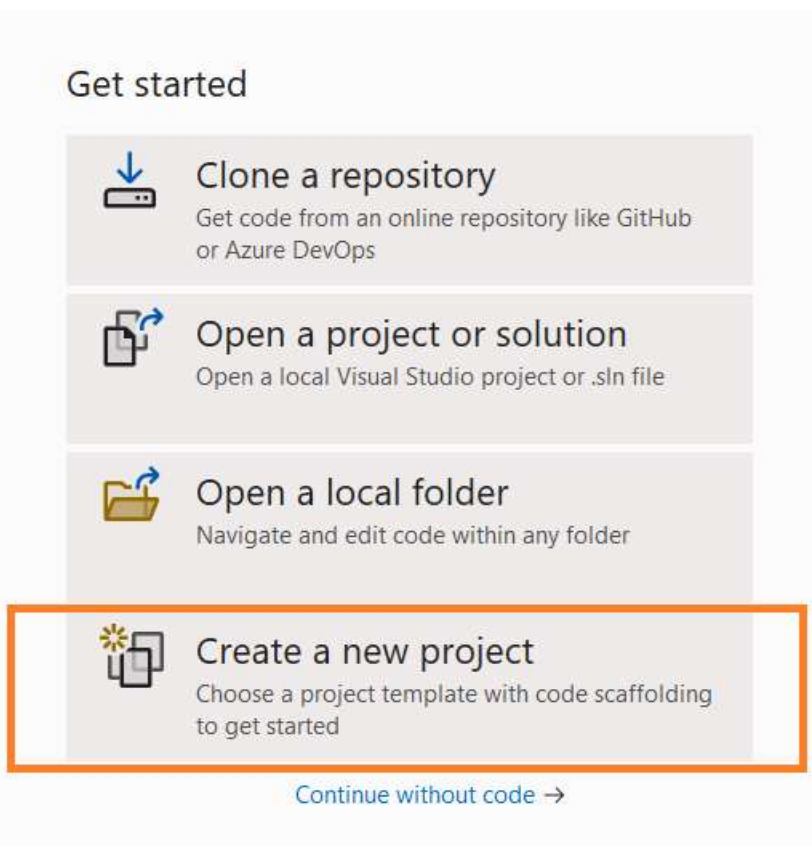
Before getting started with Syncfusion Angular Components in an ASP.NET MVC with Angular project, check whether the following are installed on the developer machine.

- [System requirements for Syncfusion Angular UI components](#)
- [Visual Studio 2022](#)

Create an ASP.NET MVC Web application

Create a new ASP.NET MVC Web application using the Microsoft template.

1. Open the Visual Studio and select the **create a new project** option.



2. Search the MVC template in the search box and select the **ASP.NET Web Application(.NET Framework)** template.

Create a new project

Recent project templates

A list of your recently accessed templates will be displayed here.


MVC


Clear all


All languages


All platforms

All project types

**ASP.NET Core Web API**
A project template for creating an ASP.NET Core application with an example Controller for a RESTful HTTP service. This template can also be used for ASP.NET Core **MVC** Views and Controllers.
C# Linux macOS Windows Cloud Service Web WebAPI

**ASP.NET Core Web App (Model-View-Controller)**
A project template for creating an ASP.NET Core application with example ASP.NET Core **MVC** Views and Controllers. This template can also be used for RESTful HTTP services.
C# Linux macOS Windows Cloud Service Web

**ASP.NET Web Application (.NET Framework)**
Project templates for creating ASP.NET applications. You can create ASP.NET Web Forms, **MVC**, or Web API applications and add many other features in ASP.NET.
C# Windows Cloud Web

**ASP.NET Web Application (.NET Framework)**
Project templates for creating ASP.NET applications. You can create ASP.NET Web Forms, **MVC**, or Web API applications and add many other features in ASP.NET.
Visual Basic Windows Cloud Web

Next

3. Enter the project name as `SyncfusionAngularASPNETMVC` and click the next button.

□ ×

Configure your new project

ASP.NET Web Application (.NET Framework) C# Windows Cloud Web

Project name

SyncfusionAngularASPNETMVC

Location

D:\ASP.NET core angular demo\Syncfusion core demo

Solution name ⓘ

SyncfusionAngularASPNETMVC

☒ Place solution and project in the same directory

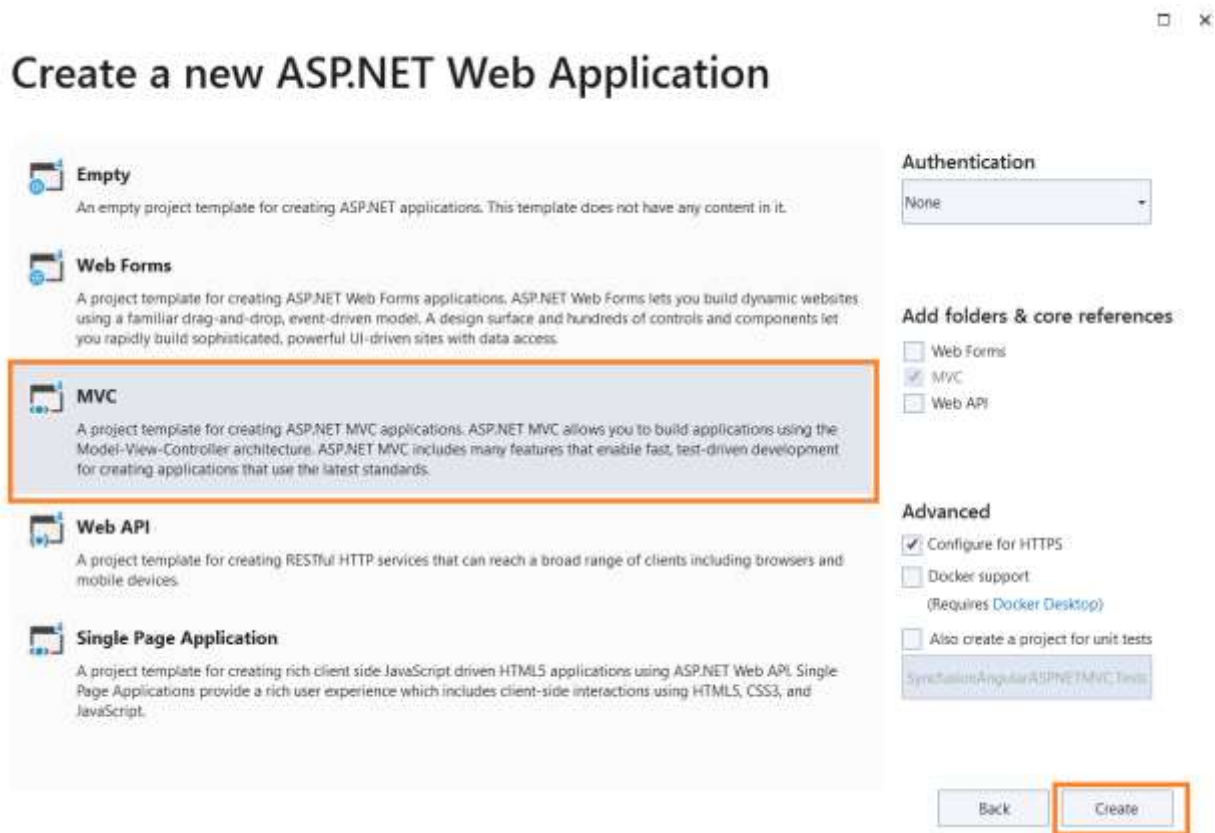
Framework

.NET Framework 4.7.2

Project will be created in "D:\ASP.NET core angular demo\Syncfusion core demo\SyncfusionAngularASPNETMVC\."

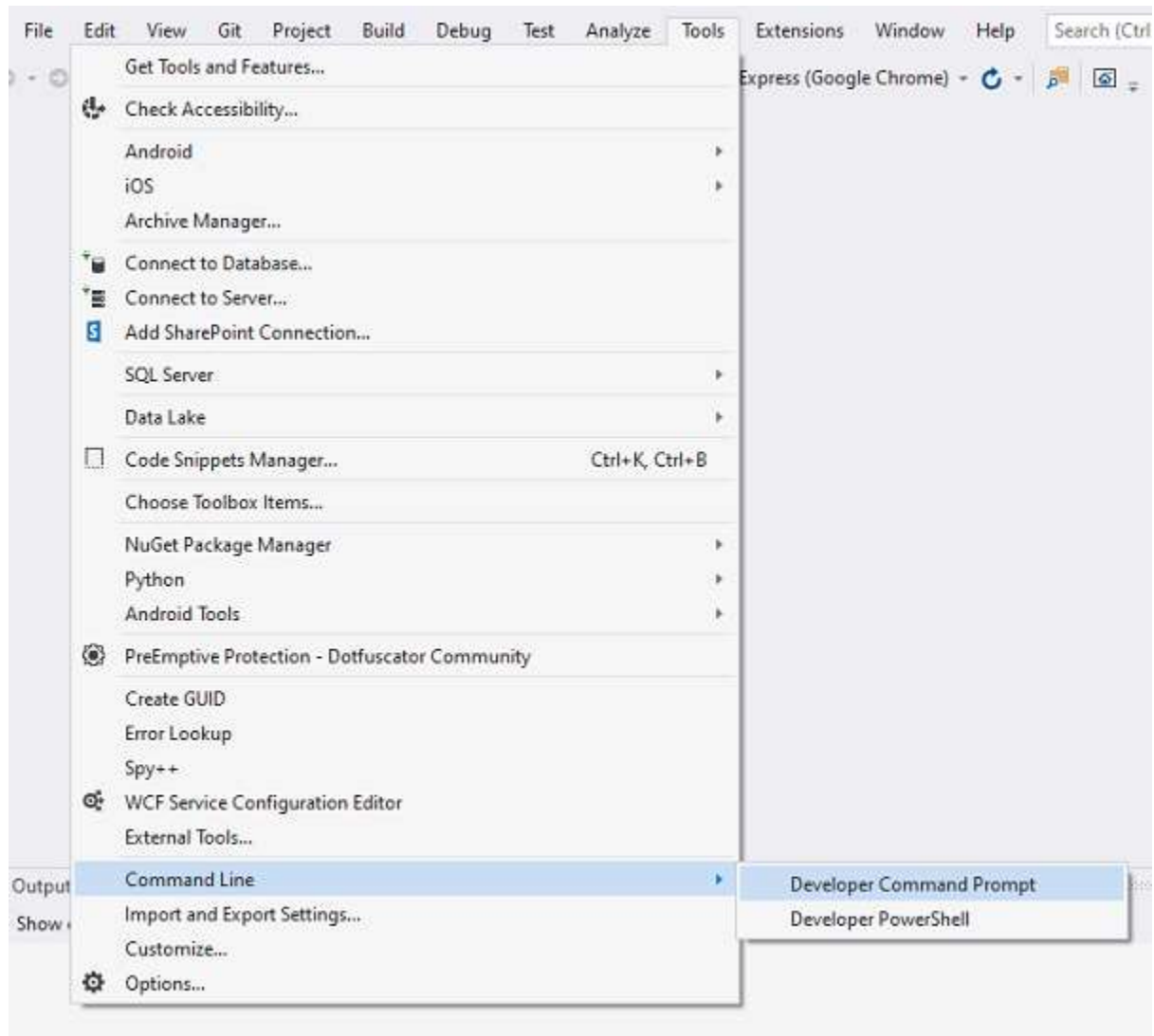
Back Create

4. Then, select **MVC** as a project template and click **Create**. Now the application is created.



Create Angular CLI application

1. Open the **Developer Command Prompt** from Visual Studio, as follows.



2. Create an Angular CLI application by executing the `ng new ClientApp` command, as shown in the following image.

```
C:\Windows\system32\cmd.exe

*****
** Visual Studio 2019 Developer Command Prompt v16.8.5
** Copyright (c) 2020 Microsoft Corporation
*****

C:\Users\<username>\source\repos\SyncfusionAngularASPNETMVC>ng new ClientApp_
```

3. Navigate to the ClientApp directory using the `cd ClientApp` command.

```
C:\Windows\system32\cmd.exe
The file will have its original line endings in your working directory
Successfully initialized git.

C:\Users\<redacted>\source\repos\SyncfusionAngularASPNETMVC>cd ClientApp

C:\Users\<redacted>\source\repos\SyncfusionAngularASPNETMVC\ClientApp>
```

4. Install and add the Syncfusion Angular components as described in the [Getting Started with Angular CLI](#) documentation. 5. Then, change the `outputPath` value to `../Scripts/ClientApp` for the production build in the `angular.json` file.

~/ANGULAR.JSON

```
"projects": {
  "ClientApp": {
    "projectType": "application",
    "schematics": {},
    "root": "",
    "sourceRoot": "src",
    "prefix": "app",
    "architect": {
      "build": {
        "builder": "@angular-devkit/build-angular:browser",
        "options": {
          "outputPath": "../Scripts/ClientApp",
        }
      }
    }
  }
}
```

Configuring ASP.NET MVC application

For building Angular application using MS Build

To automate the installation and building of the Angular application whenever the MVC application is built, add MS Build configuration to the end of the `SyncfusionAngularASPNETMVC.csproj` file, as shown in the following code sample.

SYNCFUSIONANGULARASPNETMVC.CSPROJ

```
<PropertyGroup>
  <TypeScriptCompileBlocked>true</TypeScriptCompileBlocked>
  <TypeScriptToolsVersion>Latest</TypeScriptToolsVersion>
  <IsPackable>false</IsPackable>
  <SpaRoot>ClientApp\</SpaRoot>
  <DefaultItemExcludes>$(DefaultItemExcludes); $(SpaRoot)node_modules\**</DefaultItemExcludes>
  <BuildServerSideRenderer>false</BuildServerSideRenderer>
</PropertyGroup>
<ItemGroup>
```

```

<!-- Don't publish the SPA source files, but do show them in the project
files list -->
<Content Remove="$(SpaRoot)*)" />
<None Remove="$(SpaRoot)*)" />
<None Include="$(SpaRoot)*)" Exclude="$(SpaRoot)node_modules\*" />
</ItemGroup>
<Target Name="BeforeBuild" AfterTargets="ComputeFilesToPublish">
<!-- As part of publishing, ensure the JS resources are freshly built in
production mode -->
<Exec WorkingDirectory="$(SpaRoot)" Command="npm install" />
<Exec WorkingDirectory="$(SpaRoot)" Command="npm run build --prod -- --base-
href /" />
<Exec WorkingDirectory="$(SpaRoot)" Command="npm run build:ssr --prod"
Condition=" '$(BuildServerSideRenderer)' == 'true' " />
<!-- Include the newly-built files in the publish output -->
<ItemGroup>
<DistFiles Include="$(SpaRoot)dist\*; $(SpaRoot)dist-server\*" />
<DistFiles Include="$(SpaRoot)node_modules\*"
Condition=" '$(BuildServerSideRenderer)' == 'true' " />
<ResolvedFileToPublish Include="@ (DistFiles->'%(FullPath)') "
Exclude="@ (ResolvedFileToPublish) ">
<RelativePath>%(DistFiles.Identity)</RelativePath>
<CopyToPublishDirectory>PreserveNewest</CopyToPublishDirectory>
<ExcludeFromSingleFile>true</ExcludeFromSingleFile>
</ResolvedFileToPublish>
</ItemGroup>
</Target>

```

Configure MVC Bundle with Angular production scripts

To configure the MVC bundle with Angular production script and style files, modify the `App_Start\BundleConfig.cs` file as shown in the following code snippet.

BUNDLECONFIG.CS

```

using System.Web;
using System.Web.Optimization;
namespace SyncfusionAngularASPNETMVC
{
    public class BundleConfig
    {
        // For more information on bundling, visit
        // https://go.microsoft.com/fwlink/?LinkId=301862
        public static void RegisterBundles(BundleCollection bundles)
        {
            bundles.Add(new Bundle("~/bundles/clientapp").Include(
                "~/Scripts/ClientApp/runtime.*",
                "~/Scripts/ClientApp/polyfills.*",
                "~/Scripts/ClientApp/main.*"));
            bundles.Add(new StyleBundle("~/Content/clientapp").Include(
                "~/Scripts/ClientApp/styles.*"));
        }
    }
}

```

Include Angular production scripts in MVC

To include the Angular production scripts and style files, include the following highlighted code in the `~/Views/Shared/_Layout.cshtml` file.

~/ LAYOUT.CSHTML

```

<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>@ViewBag.Title - My ASP.NET Application</title>
<base href="/" />
@Styles.Render("~/Content/css")
@Styles.Render("~/Content/clientapp")
@Scripts.Render("~/bundles/modernizr")
</head>
<body>
@RenderBody()
...
@Scripts.Render("~/bundles/jquery")
@Scripts.Render("~/bundles/bootstrap")
@Scripts.Render("~/bundles/clientapp")
@RenderSection("scripts", required: false)
</body>
</html>

```

Add the `<app-root>` tag in the `~/Views/Home/index.cshtml` file.

~/VIEWS/HOME/INDEX.CSHTML

```

@{
    ViewBag.Title = "Home Page";
}
<app-root></app-root>

```

Run the Application

Run the application from Visual Studio to render the component.

Order ID	Customer ID	Freight	Order Date
10248	VINET	\$32.38	7/4/1996
10249	TOMSP	\$11.61	7/5/1996
10250	HANAR	\$65.83	7/8/1996

[View the Angular Sample with ASP.NET MVC on GitHub.](#)

Getting started with Ionic and Angular

This document helps you create a simple Angular application with the `Ionic Framework` and including `Syncfusion Angular UI components` can be a great way to add functionality and a polished look to your app.

Prerequisites

Before getting started with Syncfusion Angular Components in an Ionic project with Angular, check whether the following are installed on the developer's machine.

- [System requirements for Syncfusion Angular UI components](#)
- ionic CLI - `^6.x.x` or later

Create an Application

To create a new project using the command prompt, use the following command:

```
`bash
npm i -g @ionic/cli
`
```

Here, we are using Node.js version 16 and Ionic version 7.0.0 to support Angular 16.

Once your development environment is set up, users can start by creating a new project using the Ionic CLI. To do this, run the following command in your command prompt,

```
`bash
ionic start syncfusion-angular-ionic blank --type=angular
`
```

This command will create a new Ionic template application in a folder called "syncfusion-angular-ionic" and will also install the default npm packages needed for the application.

Refer to this [getting started](#) document to install ionic framework.

Installing Syncfusion Grid package

Add the Syncfusion angular packages to the application which needs to be run. For example, we have add the Syncfusion Angular Grid package to the application.

```
`bash
npm i @syncfusion/ej2-angular-grids --save
`
```

Adding Grid Module

After installing the package, the component modules are available to configure your application from Syncfusion installed package.

Refer to the following code snippet to import the Grid module in `~/src/app/home/home.module.ts` from the `@syncfusion/ej2-angular-grids`.

```
`typescript
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
import { IonicModule } from '@ionic/angular';
import { FormsModule } from '@angular/forms';
import { HomePage } from './home.page';
import { GridModule } from '@syncfusion/ej2-angular-grids';
import { HomePageRoutingModule } from './home-routing.module';

@NgModule({
  imports: [
```

```

CommonModule,
FormsModule,
IonicModule,
HomePageRoutingModule,
GridModule
],
declarations: [HomePage]
})
export class HomePageModule {}
`

```

Adding Syncfusion component

After importing the package, add the following grid component code snippet in the `~/src/app/home/home.page.ts` file.

```

`typescript
import { Component } from '@angular/core';
@Component({
  selector: 'app-home',
  template: `
<h1>
Syncfusion Angular UI Grid!
</h1>
<ejs-grid [dataSource]='data'>
<e-columns>
<e-column field='OrderID' headerText='Order ID' textAlign='Right' width=90></e-column>
<e-column field='CustomerID' headerText='Customer ID' width=120></e-column>
<e-column field='Freight' headerText='Freight' textAlign='Right' format='C2' width=90></e-column>
<e-column field='OrderDate' headerText='Order Date' textAlign='Right' format='yMd' width=120></e-
column>
</e-columns>
</ejs-grid>
`,
  styleUrls: ['home.page.scss'],
})
export class HomePage {

```



```

public data: Object[] = [
{
  OrderID: 10248, CustomerID: 'VINET', EmployeeID: 5, OrderDate: new Date(8364186e5),
  ShipName: 'Vins et alcools Chevalier', ShipCity: 'Reims', ShipAddress: '59 rue de l Abbaye',
  ShipRegion: 'CJ', ShipPostalCode: '51100', ShipCountry: 'France', Freight: 32.38, Verified: !0
},
{
  OrderID: 10249, CustomerID: 'TOMSP', EmployeeID: 6, OrderDate: new Date(836505e6),
  ShipName: 'Toms Spezialitäten', ShipCity: 'Münster', ShipAddress: 'Luisenstr. 48',
  ShipRegion: 'CJ', ShipPostalCode: '44087', ShipCountry: 'Germany', Freight: 11.61, Verified: !1
},
{
  OrderID: 10250, CustomerID: 'HANAR', EmployeeID: 4, OrderDate: new Date(8367642e5),
  ShipName: 'Hanari Carnes', ShipCity: 'Rio de Janeiro', ShipAddress: 'Rua do Paço, 67',
  ShipRegion: 'RJ', ShipPostalCode: '05454-876', ShipCountry: 'Brazil', Freight: 65.83, Verified: !0
}
];
}
,

```

Adding CSS Reference

Add the following Grid component styles as specified in the `~/src/global.scss` file.

~/SRC/STYLES.CSS

```

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

```

Running the Application

Finally, run the following command to start the application. The Syncfusion Angular Grid component will be rendered in the ionic framework.

```

`bash
ionic serve
,

```

For your convenience, we have prepared an [Angular sample with ionic framework](#).

Getting started with Angular and Electron

This document helps you to create a simple Angular application with **Electron Framework** and **Syncfusion Angular UI components**.

Prerequisites

Before getting started with the Angular project, make sure you have the following installed on your machine,

- Angular version 6 or later
- TypeScript version 2.6 or later
- Electron CLI version 6.0.10 or later

If you do not have the **Electron CLI** installed, refer to the [Electron package](#) for instructions on how to install it.

Setup Angular environment

You can use follow the [Setting up the local environment and workspace](#).

Install electron framework using the following command.

```
`bash
npm install -g electron
`
```

Note: Here, we are using electron version 6.0.10 to support Angular 6.

Note: Refer to this [getting started](#) to install electron framework.

Installing Syncfusion Menu package

Syncfusion packages are distributed on npm under the **@syncfusion** scope. You can find all of the Angular Syncfusion packages [here](#).

To install the Menu package, run the following command.

```
`bash
npm install @syncfusion/ej2-angular-navigations@ngcc --save
(or)
npm i @syncfusion/ej2-angular-navigations@ngcc --save
`
```

Adding the Menu module

After installing the package, the component modules will be available for you to configure your application. The Syncfusion Angular package provides two different types of ng-Modules.

Import Menu module into Angular application (app.module.ts) from the package **@syncfusion/ej2-angular-navigations**.

```
`typescript
```

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
// Imported Syncfusion menu module from navigations package.
import { MenuModule } from '@syncfusion/ej2-angular-navigations';
import { AppComponent } from './app.component';
@NgModule({
  imports: [BrowserModule, MenuModule], // Registering EJ2 Menu Module.
  declarations: [AppComponent],
  bootstrap: [AppComponent]
})
export class AppModule { }
,
```

Adding Syncfusion Menu component

To add the Syncfusion Menu component to your application, modify the template in the `app.component.ts` file by adding the `ejs-menu` element and binding it to the `menuItems` variable.

```
`typescript
import { Component } from '@angular/core';
import { enableRipple } from '@syncfusion/ej2-base';
import { MenuItemModel } from '@syncfusion/ej2-angular-navigations';
enableRipple(true);
@Component({
  selector: 'app-root',
  template: `<!-- To Render Menu. -->
<ejs-menu [items]='menuItems'></ejs-menu>`
})
export class AppComponent {
  public menuItems: MenuItemModel[] = [
    {
      text: 'File',
      items: [
        { text: 'Open', url: 'https://www.google.com/search?q=washing+machine' },
        { text: 'Save' },
        { text: 'Exit' }
      ]
    }
  ]
}
```

```

},
{
  text: 'Edit',
  items: [
    { text: 'Cut' },
    { text: 'Copy' },
    { text: 'Paste' }
  ]
},
{
  text: 'View',
  items: [
    { text: 'Toolbar' },
    { text: 'Sidebar' }
  ]
},
{
  text: 'Tools',
  items: [
    { text: 'Spelling & Grammar' },
    { text: 'Customize' },
    { text: 'Options' }
  ]
},
{ text: 'Go' },
{ text: 'Help' }
];
}
,

```

Adding CSS reference

Add Menu component's styles as given below in `style.css`.

`typescript

```
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
```

```
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";  
`
```

Create main.js file

Create a `main.js` file in the root folder of the project, and add the following code in `main.js` file

```
`typescript  
const { app, BrowserWindow } = require('electron');  
let win;  
function createWindow () {  
  // Create the browser window.  
  win = new BrowserWindow({ width: 800, height: 600 });  
  // Load the index.html of the app.  
  win.loadFile('./dist/my-app/index.html');  
  // Open the DevTools.  
  win.webContents.openDevTools();  
  // Emitted when the window is closed.  
  win.on('closed', () => {  
    win = null  
  })  
};  
// This method will be called when Electron finish initialization and is ready to create browser windows.  
// Some APIs can only be used after this event occurs.  
app.on('ready', createWindow);  
// Quit when all windows are closed.  
app.on('window-all-closed', () => {  
  // On macOS, it is common for applications and their menu bar to stay active until the user quits  
  // explicitly with Cmd + Q.  
  if (process.platform !== 'darwin') {  
    app.quit()  
  }  
});  
app.on('activate', () => {  
  // On macOS, it is common to re-create a Window in an app when the dock icon is clicked and there are  
  // no other windows open.  
  if (win === null) {
```

```
createWindow()
}
});
`
```

Update index.html

In the `/src/index.html` file, change `<base href="/">` as `<base href=".">`, so that the Electron can able to find the Angular files.

Update package.json

```
`typescript
"main": "main.js",
"scripts": {
  "ng": "ng",
  "start": "ng serve",
  "build": "ng build",
  "test": "ng test",
  "lint": "ng lint",
  "e2e": "ng e2e",
  "electron-build": "ng build --prod",
  "electron": "electron ."
},
`
```

Then, include the above code in the `package.json` file.

Update tsconfig.json

In the `tsconfig.json` file, change the target as demonstrated in the following code sample.

```
`typescript
"target": "es5"
`
```

Running the application

Finally, run the following command line to start the application. The Syncfusion Essential JS 2 menu component will be rendered in the Electron framework.

```
`bash
npm run electron-build
npm run electron
`
```

Note: For your convenience, we have prepared an [Angular sample with electron framework](#).

See also

- [Electron Browser Window](#)
- [Angular 10 Electron Tutorial](#)
- [Build Angular Desktop Apps With Electron Tutorial](#)
- [Getting started with angular CLI](#)

Appearance

Angular Theme in Syncfusion Components

The Syncfusion Angular UI can allow you to apply styles for your application. The following list of themes are included in the Syncfusion Angular components library

|Theme |Style Sheet Name|

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

|Material 3 | material3.css |

|Material 3 Dark | material3-dark.css |

|Bootstrap 5 | bootstrap5.css |

|Bootstrap 5 Dark | bootstrap5-dark.css |

| Fluent | fluent.css |

| Fluent Dark | fluent-dark.css |

|Google's Material | material.css |

|Google's Material-Dark | material-dark.css |

|Tailwind CSS | tailwind.css |

|TailwindDark CSS | tailwind-dark.css |

|Microsoft Office Fabric | fabric.css |

|Microsoft Office Fabric Dark | fabric-dark.css |

|High Contrast | highcontrast.css |

Installation

There are four ways to include Syncfusion Angular UI themes in application.

Install via NPM

Themes are shipped as individual and combined CSS files. Combined CSS file can be referred from the npm package `@syncfusion/ej2` and individual CSS files are available within same component repository's `style` folder. In ej2 npm packages, we have shipped both CSS and SCSS files for all components.

To use the combined CSS files, install the npm package using the following command

```
`bash
```

```
npm install @syncfusion/ej2
```

```
,
```

CDN Direct Referral

Instead of using a local resource on your server, you can use a cloud CDN to reference the theme style sheets.

Syncfusion Angular Themes are available in the CDN. Make sure that the version in the URLs matches the version of the Syncfusion Angular Package you are using.

`

```
<head>
```

```
<link href="https://cdn.syncfusion.com/ej2/<version>/<theme_name>.css" rel="stylesheet"/>
```

```
</head>
```

`

Theme Name	CDN Reference
Material 3	https://cdn.syncfusion.com/ej2/22.1.34/material3.css
Material 3 Dark	https://cdn.syncfusion.com/ej2/22.1.34/material3-dark.css
Fluent	https://cdn.syncfusion.com/ej2/22.1.34/fluent.css
Fluent Dark	https://cdn.syncfusion.com/ej2/22.1.34/fluent-dark.css
Bootstrap 5	https://cdn.syncfusion.com/ej2/22.1.34/bootstrap5.css
Bootstrap 5 Dark	https://cdn.syncfusion.com/ej2/22.1.34/bootstrap5-dark.css
Bootstrap 4	https://cdn.syncfusion.com/ej2/22.1.34/bootstrap4.css
Bootstrap 3	https://cdn.syncfusion.com/ej2/22.1.34/bootstrap.css
Bootstrap 3 Dark	https://cdn.syncfusion.com/ej2/22.1.34/bootstrap-dark.css
Google's Material	https://cdn.syncfusion.com/ej2/22.1.34/material.css
Google's Material Dark	https://cdn.syncfusion.com/ej2/22.1.34/material-dark.css
Tailwind CSS	https://cdn.syncfusion.com/ej2/22.1.34/tailwind.css
Tailwind CSS Dark	https://cdn.syncfusion.com/ej2/22.1.34/tailwind-dark.css
Microsoft Office Fabric	https://cdn.syncfusion.com/ej2/22.1.34/fabric.css
Microsoft Office Fabric Dark	https://cdn.syncfusion.com/ej2/22.1.34/fabric-dark.css
High Contrast	https://cdn.syncfusion.com/ej2/22.1.34/highcontrast.css

Theme Name	CDN Reference
Material 3	https://cdn.syncfusion.com/ej2/22.1.34/material3.css
Material 3 Dark	https://cdn.syncfusion.com/ej2/22.1.34/material3-dark.css
Fluent	https://cdn.syncfusion.com/ej2/22.1.34/fluent.css
Fluent Dark	https://cdn.syncfusion.com/ej2/22.1.34/fluent-dark.css
Bootstrap 5	https://cdn.syncfusion.com/ej2/22.1.34/bootstrap5.css
Bootstrap 5 Dark	https://cdn.syncfusion.com/ej2/22.1.34/bootstrap5-dark.css
Bootstrap 4	https://cdn.syncfusion.com/ej2/22.1.34/bootstrap4.css
Bootstrap 3	https://cdn.syncfusion.com/ej2/22.1.34/bootstrap.css
Bootstrap 3 Dark	https://cdn.syncfusion.com/ej2/22.1.34/bootstrap-dark.css
Google's Material	https://cdn.syncfusion.com/ej2/22.1.34/material.css
Google's Material Dark	https://cdn.syncfusion.com/ej2/22.1.34/material-dark.css
Tailwind CSS	https://cdn.syncfusion.com/ej2/22.1.34/tailwind.css
Tailwind CSS Dark	https://cdn.syncfusion.com/ej2/22.1.34/tailwind-dark.css
Microsoft Office Fabric	https://cdn.syncfusion.com/ej2/22.1.34/fabric.css
Microsoft Office Fabric Dark	https://cdn.syncfusion.com/ej2/22.1.34/fabric-dark.css
High Contrast	https://cdn.syncfusion.com/ej2/22.1.34/highcontrast.css

*Using Precompiled CSS and SCSS File**Precompiled css*

In terms Precompiled means minified and optimized CSS.

The Syncfusion Angular UI theme includes a precompiled CSS file that contains the styles for all Syncfusion components.

```
`css
```

```
@import "../nodemodules/@syncfusion/ej2/<themenam>.css";
```


`

Referring all components SCSS

`scss

```
@import "ej2/<theme_name>.scss";
```

`

We can get the individual theme from [individual package](#) or from ej2 package.

Referring individual Component from individual package

`scss

```
@import "ej2-buttons/button/<theme_name>.scss";
```

`

Advantages of individual components theme

- Reducing the page load time of application
- Reducing bundling size
- Avoid unused CSS

Compiling Themes from SCSS Source File

The Syncfusion Angular UI theme includes a compilation of SCSS file. Refer [Compiling SCSS file](#) section for more information.

Common Variables

The following list of common variables is used in the Syncfusion Angular library themes for all UI components. You can change these variables to customize the corresponding theme.

Syncfusion Material 3 theme

Name	Value (Default Theme)	Value (Dark Theme)
--color-sf-black	rgb(0,0,0)	rgb(0,0,0)
--color-sf-white	rgb(255,255,255)	rgb(255,255,255)
--color-sf-primary	rgb(103, 80, 164)	rgb(208, 188, 255)
--color-sf-primary-container	rgb(234, 221, 255)	rgb(79, 55, 139)
--color-sf-on-primary	rgb(255, 255, 255)	rgb(55, 30, 115)
--color-sf-on-primary-container	rgb(33, 0, 94)	rgb(234, 221, 255)
--color-sf-surface	rgb(255, 255, 255)	rgb(28, 27, 31)
--color-sf-surface-variant	rgb(231, 224, 236)	rgb(73, 69, 79)

Name	Value (Default Theme)	Value (Dark Theme)
--color-sf-on-surface	rgb(28, 27, 31)	rgb(230, 225, 229)
--color-sf-on-surface-variant	rgb(73, 69, 78)	rgb(202, 196, 208)
--color-sf-secondary	rgb(98, 91, 113)	rgb(204, 194, 220)
--color-sf-secondary-container	rgb(232, 222, 248)	rgb(74, 68, 88)
--color-sf-on-secondary	rgb(255, 255, 255)	rgb(51, 45, 65)
--color-sf-on-secondary-container	rgb(30, 25, 43)	rgb(232, 222, 248)
--color-sf-tertiary	rgb(125, 82, 96)	rgb(239, 184, 200)
--color-sf-tertiary-container	rgb(255, 216, 228)	rgb(99, 59, 72)
--color-sf-on-tertiary	rgb(255, 255, 255)	rgb(73, 37, 50)
--color-sf-on-tertiary-containe	rgb(55, 11, 30)	rgb(255, 216, 228)
--color-sf-background	rgb(255, 255, 255)	rgb(28, 27, 31)
--color-sf-on-background	rgb(28, 27, 31)	rgb(230, 225, 229)
--color-sf-outline	rgb(121, 116, 126)	rgb(147, 143, 153)
--color-sf-outline-variant	rgb(196, 199, 197)	rgb(68, 71, 70)
--color-sf-shadow	rgb(0, 0, 0)	rgb(0, 0, 0)
--color-sf-surface-tint-color	rgb(103, 80, 164)	rgb(208, 188, 255)
--color-sf-inverse-surface	rgb(49, 48, 51)	rgb(230, 225, 229)
--color-sf-inverse-on-surface	rgb(244, 239, 244)	rgb(49, 48, 51)
--color-sf-inverse-primary	rgb(208, 188, 255)	rgb(103, 80, 164)
--color-sf-scrim	rgb(0, 0, 0)	rgb(0, 0, 0)
--color-sf-error	rgb(179, 38, 30)	rgb(242, 184, 181)
--color-sf-error-container	rgb(249, 222, 220)	rgb(140, 29, 24)

Name	Value (Default Theme)	Value (Dark Theme)
--color-sf-on-error	rgb(255, 250, 250)	rgb(96, 20, 16)
--color-sf-on-error-container	rgb(65, 14, 11)	rgb(249, 222, 220)
--color-sf-success	rgb(32, 81, 7)	rgb(83, 202, 23)
--color-sf-success-container	rgb(209, 255, 186)	rgb(22, 62, 2)
--color-sf-on-success	rgb(244, 255, 239)	rgb(13, 39, 0)
--color-sf-on-success-container	rgb(13, 39, 0)	rgb(183, 250, 150)
--color-sf-info	rgb(1, 87, 155)	rgb(71, 172, 251)
--color-sf-info-container	rgb(233, 245, 255)	rgb(0, 67, 120)
--color-sf-on-info	rgb(250, 253, 255)	rgb(0, 51, 91)
--color-sf-on-info-container	rgb(0, 51, 91)	rgb(173, 219, 255)
--color-sf-warning	rgb(145, 76, 0)	rgb(245, 180, 130)
--color-sf-warning-container	rgb(254, 236, 222)	rgb(123, 65, 0)
--color-sf-on-warning	rgb(255, 255, 255)	rgb(99, 52, 0)
--color-sf-on-warning-container	rgb(47, 21, 0)	rgb(255, 220, 193)

Bootstrap 5 Theme

Name	Value (Default Theme)	Value (Dark Theme)
\$black	#000	#000
\$white	#fff	#fff
\$gray-100	#f8f9fa	#f8f9fa
\$gray-200	#e9ecef	#e9ecef
\$gray-300	#dee2e6	#dee2e6
\$gray-400	#ced4da	#ced4da

Name	Value (Default Theme)	Value (Dark Theme)
\$gray-500	#adb5bd	#adb5bd
\$gray-600	#6c757d	#6c757d
\$gray-700	#495057	#495057
\$gray-800	#343a40	#343a40
\$gray-900	#212529	#212529
\$blue	#0d6efd	#0d6efd
\$indigo	#6610f2	#6610f2
\$purple	#6f42c1	#6f42c1
\$pink	#d63384	#d63384
\$red	#dc3545	#dc3545
\$orange	#fd7e14	#fd7e14
\$yellow	#ffc107	#ffc107
\$green	#198754	#198754
\$teal	#20c997	#20c997
\$cyan	#0dcaf0	#0dcaf0

Fluent Theme

Name	Value (Default Theme)	Value (Dark Theme)
\$black	#000	#000
\$white	#fff	#fff
\$gray220	#11100f	#11100f
\$gray210	#161514	#161514
\$gray200	#1b1a19	#1b1a19

Name	Value (Default Theme)	Value (Dark Theme)
\$gray190	#201f1e	#201f1e
\$gray180	#252423	#252423
\$gray170	#292827	#292827
\$gray160	#323130	#323130
\$gray150	#3b3a39	#3b3a39
\$gray140	#484644	#484644
\$gray130	#605e5c	#605e5c
\$gray120	#797775	#797775
\$gray110	#8a8886	#8a8886
\$gray100	#979593	#979593
\$gray90	#a19f9d	#a19f9d
\$gray80	#b3b0ad	#b3b0ad
\$gray70	#bebbb8	#bebbb8
\$gray60	#c8c6c4	#c8c6c4
\$gray50	#d2d0ce	#d2d0ce
\$gray40	#e1dfdd	#e1dfdd
\$gray30	#edebe9	#edebe9
\$gray20	#f3f2f1	#f3f2f1
\$gray10	#faf9f8	#faf9f8
\$cyanblue10	#0078d4	#0078d4
\$red10	#d13438	#d13438
\$orange20	#ca5010	#ca5010

Name	Value (Default Theme)	Value (Dark Theme)
\$green20	#0b6a0b	#0b6a0b
\$cyan20	#038387	#038387

Material Theme

Name	Value (Default Theme)	Value (Dark Theme)
\$accent	#e3165b	#ff80ab
\$accent-font	#fff	#000
\$primary	#3f51b5	#3f51b5
\$primary-50	#e8eaf6	#e8eaf6
\$primary-100	#c5cae9	#c5cae9
\$primary-200	#9fa8da	#9fa8da
\$primary-300	#7986cb	#7986cb
\$primary-font	#fff	#fff
\$primary-50-font	#000	#000
\$primary-100-font	#000	#000
\$primary-200-font	#000	#000
\$primary-300-font	#fff	#fff
\$grey-white	#fff	#fff
\$grey-black	#000	#000
\$grey-50	#fafafa	#fafafa
\$grey-100	#f5f5f5	#f5f5f5
\$grey-200	#eee	#eee
\$grey-300	#e0e0e0	#e0e0e0

Name	Value (Default Theme)	Value (Dark Theme)
\$grey-400	#bdbdbd	#bdbdbd
\$grey-500	#9e9e9e	#9e9e9e
\$grey-600	#757575	#757575
\$grey-700	#616161	#616161
\$grey-800	#424242	#424242
\$grey-900	#212121	#212121
\$grey-dark	#303030	#303030
\$grey-light-font	#000	#000
\$grey-dark-font	#fff	#fff
\$base-font	#000	#000
\$error-font	#f44336	#ff6652
\$success-bg		#4caf50
\$error-bg		#ff6652
\$warning-bg		#ff9800
\$info-bg		#03a9f4
\$message-font		#fff
\$success-font		#4caf50
\$warning-font		#ff9800
\$info-font		#03a9f4

Tailwind CSS Theme

Name	Value (Default Theme)	Value (Dark Theme)
\$black	#000	#000

Name	Value (Default Theme)	Value (Dark Theme)
\$white	#fff	#fff
\$transparent	transparent	transparent
\$cool-gray-50	#f9fafb	#f9fafb
\$cool-gray-100	#f3f4f6	#f3f4f6
\$cool-gray-200	#e5e7eb	#e5e7eb
\$cool-gray-300	#d1d5db	#d1d5db
\$cool-gray-400	#9ca3af	#9ca3af
\$cool-gray-500	#6b7280	#6b7280
\$cool-gray-600	#4b5563	#4b5563
\$cool-gray-700	#374151	#374151
\$cool-gray-800	#1f2937	#1f2937
\$cool-gray-900	#111827	#111827
\$red-100	#fee2e2	#fee2e2
\$red-400	#f87171	#f87171
\$red-500	#ef4444	#ef4444
\$red-600	#dc2626	#dc2626
\$red-800	#991b1b	#991b1b
\$green-100	#dcfce7	#dcfce7
\$green-500	#22c55e	#22c55e
\$green-600	#16a34a	#16a34a
\$green-700	#15803d	#15803d
\$orange-100	#ffedd5	#ffedd5

Name	Value (Default Theme)	Value (Dark Theme)
\$orange-500	#f97316	#f97316
\$orange-600	#ea580c	#ea580c
\$orange-700	#c2410c	#c2410c
\$orange-800	#9a3412	#9a3412
\$cyan-300	#67e8f9	#67e8f9
\$cyan-400	#22d3ee	#22d3ee
\$cyan-500	#06b6d4	#06b6d4
\$cyan-600	#0891b2	#0891b2
\$cyan-800	#155e75	#155e75
\$indigo-50	#eef2ff	
\$indigo-100	#e0e7ff	
\$indigo-200	#c7d2fe	
\$indigo-300	#a5b4fc	
\$indigo-400	#818cf8	
\$indigo-500	#6366f1	
\$indigo-600	#4f46e5	
\$indigo-700	#4338ca	
\$indigo-800	#3730a3	
\$indigo-900	#312e81	
\$green-400		#4ade80
\$light-blue-50		#f0f9ff
\$light-blue-100		#e0f2fe

Name	Value (Default Theme)	Value (Dark Theme)
\$light-blue-400		#38bdf8
\$light-blue-500		#0ea5e9
\$light-blue-600		#0284c7
\$light-blue-700		#0369a1
\$light-blue-800		#075985

Microsoft Office Fabric Theme

Name	Value (Default Theme)	Value (Dark Theme)
\$theme-primary	#0078d6	#0074cc
\$theme-dark-alt	darken(\$theme-primary, 3%)	darken(\$theme-primary, 3%)
\$theme-dark	darken(\$theme-primary, 10%)	darken(\$theme-primary, 6%)
\$theme-darker	darken(\$theme-primary, 18%)	darken(\$theme-primary, 10%)
\$theme-secondary	lighten(\$theme-primary, 3%)	lighten(\$theme-primary, 3%)
\$theme-tertiary	lighten(\$theme-primary, 21%)	lighten(\$theme-primary, 21%)
\$theme-light	lighten(\$theme-primary, 44%)	lighten(\$theme-primary, 44%)
\$theme-lighter	lighten(\$theme-primary, 49%)	lighten(\$theme-primary, 49%)
\$theme-lighter-alt	lighten(\$theme-primary, 55%)	lighten(\$theme-primary, 55%)
\$neutral-white	#fff	#201f1f
\$neutral-lighter-alt	#f8f8f8	#282727
\$neutral-lighter	#f4f4f4	#333232
\$neutral-light	#eaeaea	#414040
\$neutral-quintenaryalt	#dadada	#4a4848
\$neutral-quintenary	#d0d0d0	#514f4f

Name	Value (Default Theme)	Value (Dark Theme)
\$neutral-tertiary-alt	#c8c8c8	#6f6c6c
\$neutral-tertiary	#a6a6a6	#9a9a9a
\$neutral-secondary-alt	#767676	#c8c8c8
\$neutral-secondary	#666	#dadada
\$neutral-primary	#333	#fff
\$neutral-dark	#212121	#f4f4f4
\$neutral-black	#000	#f8f8f8
\$alert-bg	#deecf9	#bf7500
\$error-bg	#fde7e9	#cd2a19
\$success-bg	#dff6dd	#37844d
\$theme-dark-font	#fff	#fff
\$theme-primary-font	#fff	#fff
\$theme-light-font	#333	#000
\$neutral-light-font	#333	#dadada
\$neutral-light-fontalt	#000	#fff
\$grey-dark-font	#fff	#000
\$base-font	#333	#dadada
\$message-font	#333	#fff
\$alert-font	#d83b01	#ff9d48
\$error-font	#a80000	#ff5f5f
\$success-font	#107c10	#8eff8d
\$info-bg		#1e79cb

Name	Value (Default Theme)	Value (Dark Theme)
\$info-font		#62cfff

High Contrast Theme

Name	Value
\$selection-bg	#ffd939
\$selection-font	#000
\$selection-border	#ffd939
\$hover-bg	#685708
\$hover-font	#fff
\$hover-border	#fff
\$border-default	#969696
\$border-alt	#757575
\$border-fg	#fff
\$border-fg-alt	#ffd939
\$bg-base-0	#000
\$bg-base-5	#0d0d0d
\$bg-base-10	#1a1a1a
\$bg-base-15	#262626
\$bg-base-20	#333
\$bg-base-75	#bfbfbf
\$bg-base-100	#fff
\$header-font	#ffd939
\$header-font-alt	#fff

Name	Value
\$content-font	#fff
\$content-font-alt	#969696
\$link	#8a8aff
\$invert-font	#000
\$success-bg	#166600
\$error-bg	#b30900
\$message-font	#fff
\$alert-bg	#944000
\$info-bg	#0056b3
\$success-alt	#2bc700
\$error-alt	#ff6161
\$alert-alt	#ff7d1a
\$info-alt	#66b0ff
\$disable	#757575

Suggested link

- [Syncfusion icons and Customization](#)
- [Theme studio for Syncfusion Component](#)
- [Angular with SCSS compilation](#)

Size Mode for Syncfusion Angular Components

An application that is designed to be accessed through a web browser on various devices, including desktop computers and mobile devices, may have a distinct layout or user interface on a mobile device compared to a desktop computer to better suit the smaller screen size.

Syncfusion Angular components support both touch (bigger) and normal size modes. Touch mode creates a responsive design for mobile devices by adding the `e-bigger` class, which enhances interactions, visibility, and the overall experience.

Size mode for application

The user can enable touch mode (bigger) for the entire application by adding the `e-bigger` class to the `body` element in the `index.html` file as follows:

```
`html
<body className="e-bigger">
...
</body>
`
```

Size mode for a component

The user can enable touch mode (bigger) for a component by adding the `e-bigger` class to the `div` element that contains the component. Another way of enabling touch mode is by adding the `e-bigger` class using the available `cssClass` property of the component.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component, } from '@angular/core';
@Component({
  imports: [
    BrowserModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<div class="e-bigger">
<button ej2-button>Button</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));
```

Change size mode for application at runtime

The user can change the size mode of the application between touch and normal (mouse) mode at runtime by adding and removing the `e-bigger` class. The following steps explain how to change the size mode of an application at runtime:

APP.COMPONENT.TS

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

```
import { BrowserModule } from '@angular/platform-browser'
import { CalendarModule } from '@syncfusion/ej2-angular-calendars'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component, ViewChild } from '@angular/core';
@Component({
  imports: [

    CalendarModule,
    ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `
    <div>
      <div>
        <button ej2-button content="Touch Mode"
(click)="onTouchClick($event)"></button>
        <button ej2-button content="Mouse Mode"
(click)="onMouseClick($event)"></button>
      </div>
      <div>
        <ejs-calendar></ejs-calendar>
      </div>
    </div>
  `
})
export class AppComponent {
  onTouchClick(e: any): void {
    document.body.classList.add('e-bigger');
  }
  onMouseClick(e: any): void {
    document.body.classList.remove('e-bigger');
  }
}
```

MAIN.TS

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

Change size mode for a component at runtime

The user can change the size mode of a component between touch and normal (mouse) mode at runtime by setting the **e-bigger** CSS class.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { CalendarModule } from '@syncfusion/ej2-angular-calendars'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component, ViewChild } from '@angular/core';
@NgModule({
  imports: [
```

```

        CalendarModule,
        ButtonModule
    ],
    standalone: true,
    selector: 'app-root',
    template: `
<div>
  <div>
    <button ejs-button content="Touch Mode"
(click)="onTouchClick($event)"></button>
    <button ejs-button content="Mouse Mode"
(click)="onMouseClicked($event)"></button>
  </div>
  <div class="control">
    <ejs-calendar></ejs-calendar>
  </div>
</div>
`
  })
}
export class AppComponent {
  onTouchClick(e: any): void {
    let controls = document.querySelectorAll('.control');
    for (let index: number = 0; index < controls.length; index++) {
      controls[index].classList.add('e-bigger');
    }
  }
  onMouseClick(e: any): void {
    let controls = document.querySelectorAll('.control');
    for (let index: number = 0; index < controls.length; index++) {
      controls[index].classList.remove('e-bigger');
    }
  }
}
}

```

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

- [Sidebar responsiveness](#)
- [DataGrid responsiveness](#)
- [TreeGrid responsiveness](#)
- [Dashboard Layout responsiveness](#)
- [Kanban responsiveness](#)
- [Toolbar responsiveness](#)
- [Tab responsiveness](#)

Icons in Angular Appearance component

Syncfusion's icon library is a collection of pre-designed icons that can be used to enhance the user interface of an application. This pre-designed icons are set of `base64` formatted font icons. Utilizing this icon library can make it simpler to create a cohesive, visually pleasing design for an application.

Referring icons in Angular application

Using the below approaches, the icons can be referenced in the Angular application.

- [npm package](#) - Use the npm package to access icons.
- [CDN reference](#) - Use the static web asset to access icons.

The npm package

All Syncfusion theme icons are shipped in the [ej2-icons](#) package, which is published on the [npmjs.com](#) public registry. This package contains both CSS and SCSS theme files for all themes.

Icons can be used from the npm package `ej2-icons`. To use the icons, install the npm package using the following command:

```
`bash
npm install @syncfusion/ej2-icons
`
```

Refer to the following syntax to use icons in a Angular application:

```
`css
@import "../nodemodules/@syncfusion/ej2-icons/<themename>.css";
`
```

Example:

```
`css
@import "../node_modules/@syncfusion/ej2-icons/material.css";
`
```

CDN reference

All Syncfusion theme icons are available on the CDN. Instead of using a local resource on the server, use a cloud CDN to refer to the icons.

Make sure that the version of the icons in the URL matches the version of the Syncfusion Angular package. This will prevent compatibility issues and ensure that the correct version of the icons is loaded.

To use the icons from the CDN, refer to the icons by URLs in the application. This can be done by linking the icons in the HTML file by adding a link tag to the head section.

```
`html
// Bootstrap5
<head>
<link href="https://cdn.syncfusion.com/ej2/ej2-icons/styles/bootstrap5.css" rel="stylesheet"/>
</head>
```

```
//Material
<head>
<link href="https://cdn.syncfusion.com/ej2/ej2-icons/styles/material.css" rel="stylesheet"/>
</head>
```

Steps to use icons library

Let's create a Angular application using the following command:

For an introduction and configuration of the common specifications, see [getting started with the Syncfusion Angular application](#).

Using icons directly in HTML element

The built-in Syncfusion icons can be rendered directly in the HTML element by defining the `e-icons` class, which contains the font-family and common properties of font icons, and defining the available icon's class with the `e-` prefix.

The following steps explain the direct rendering of the Syncfusion icon in the HTML element.

1. Add the class name `e-icons` to the HTML element that needs to render the icon.
2. Add the icon class with corresponding icon content from the [available icons](#). For example, the below code snippet represents the paste icon class.

```
`css
.e-paste:before{
content:'\e355';
}
```

3. Add `e-icons` and `e-paste` classes to the HTML element.

```
`html
<span class="e-icons e-paste"></span>
```

4. Add the CDN link reference of icons library in the `~index.html` file.

```
`html
<link href="https://cdn.syncfusion.com/ej2/ej2-icons/styles/bootstrap5.css" rel="stylesheet" />
```

The below code snippet represents the complete example of icon usage.

APP.COMPONENT.TS

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

```
import { BrowserModule } from '@angular/platform-browser'
import { Component, ViewChild } from '@angular/core';
@Component({
  standalone: true,
  selector: 'app-root',
  template: `
    <div class="icon-bar">
      <span class="e-icons e-cut"></span>
      <span class="e-icons e-copy"></span>
      <span class="e-icons e-paste"></span>
    </div>
  `
})
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));
```

Icon size

The `ej2-icons` package offers options to display icons in different size modes. A user can use different icon sizes in their application based on touch or mouse mode. If the user is using touch mode, add `e-large` class to the element to make the icon easily interact, or add the `e-small` or `e-medium` class in mouse mode.

The pre-defined icon size is present in the available classes listed below.

- `e-small` - Sets the icon size as 8px.
- `e-medium` - Sets the icon size to 16px.
- `e-large` - Sets the icon size to 24px.

Example:

```
`html
<span class="e-icons e-small e-search"></span>
<span class="e-icons e-medium e-search"></span>
<span class="e-icons e-large e-search"></span>
`
```

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Component, ViewChild } from '@angular/core';
import { Animation } from '@syncfusion/ej2-base';
@Component({
```

```

standalone: true,
selector: 'app-root',
template: `
  <div>
    <p><b>Smaller Icons</b></p>
    <div class="small-icon-bar">
      <span class="e-icons e-small e-cut"></span>
      <span class="e-icons e-small e-copy"></span>
      <span class="e-icons e-small e-paste"></span>
    </div>
    <p><b>Medium Icons</b></p>
    <div class="medium-icon-bar">
      <span class="e-icons e-medium e-cut"></span>
      <span class="e-icons e-medium e-copy"></span>
      <span class="e-icons e-medium e-paste"></span>
    </div>
    <p><b>Larger Icons</b></p>
    <div class="large-icon-bar">
      <span class="e-icons e-large e-cut"></span>
      <span class="e-icons e-large e-copy"></span>
      <span class="e-icons e-large e-paste"></span>
    </div>
  </div>
`
})
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));

```

Icon appearance customization

The Syncfusion Angular icons can be customized with custom color and size by overriding the `e-icons` class. Customizing the icons in the library can be useful for making the icons more visually appealing and fitting to the design of the application. For example, a user can change the color of an icon to match the color scheme of their application, or increase the size of an icon to make it more visible on smaller screens. It may also be useful for creating a consistent look and feel across different parts of the application. Overall, customizing the icons in the library can improve the overall user experience of the application.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Component, ViewChild } from '@angular/core';
import { Animation } from '@syncfusion/ej2-base';
@Component({
  standalone: true,
  selector: 'app-root',
  template: `

```

```

    <div class="icon-bar">
      <span class="e-icons e-cut"></span>
      <span class="e-icons e-copy"></span>
      <span class="e-icons e-paste"></span>
    </div>
  ,
  })
  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));

```

Available Icons

The complete package of Essential JS 2 icons is listed below. The corresponding icon content can be referred in the content section.

<!-- markdownlint-disable MD033 -->

Bootstrap 5

```

<iframe class="doc-sample-frame"
src="https://ej2.syncfusion.com/products/icons/bootstrap5/demo.html"
style="height:1000px;width:100%;"></iframe>

```

Bootstrap 4

```

<iframe class="doc-sample-frame"
src="https://ej2.syncfusion.com/products/icons/bootstrap4/demo.html"
style="height:1000px;width:100%;"></iframe>

```

Bootstrap

```

<iframe class="doc-sample-frame"
src="https://ej2.syncfusion.com/products/icons/bootstrap/demo.html"
style="height:1000px;width:100%;"></iframe>

```

Material

```

<iframe class="doc-sample-frame"
src="https://ej2.syncfusion.com/products/icons/material/demo.html"
style="height:1000px;width:100%;"></iframe>

```

Tailwind CSS

```

<iframe class="doc-sample-frame"
src="https://ej2.syncfusion.com/products/icons/tailwind/demo.html"
style="height:1000px;width:100%;"></iframe>

```

Office Fabric

```

<iframe class="doc-sample-frame" src="https://ej2.syncfusion.com/products/icons/fabric/demo.html"
style="height:1000px;width:100%;"></iframe>

```

High Contrast

```
<iframe class="doc-sample-frame"
src="https://ej2.syncfusion.com/products/icons/highcontrast/demo.html"
style="height:1000px;width:100%;"></iframe>
```

Fluent

```
<iframe class="doc-sample-frame" src="https://ej2.syncfusion.com/products/icons/fluent/demo.html"
style="height:1000px;width:100%;"></iframe>
```

Theme studio in Angular Appearance component

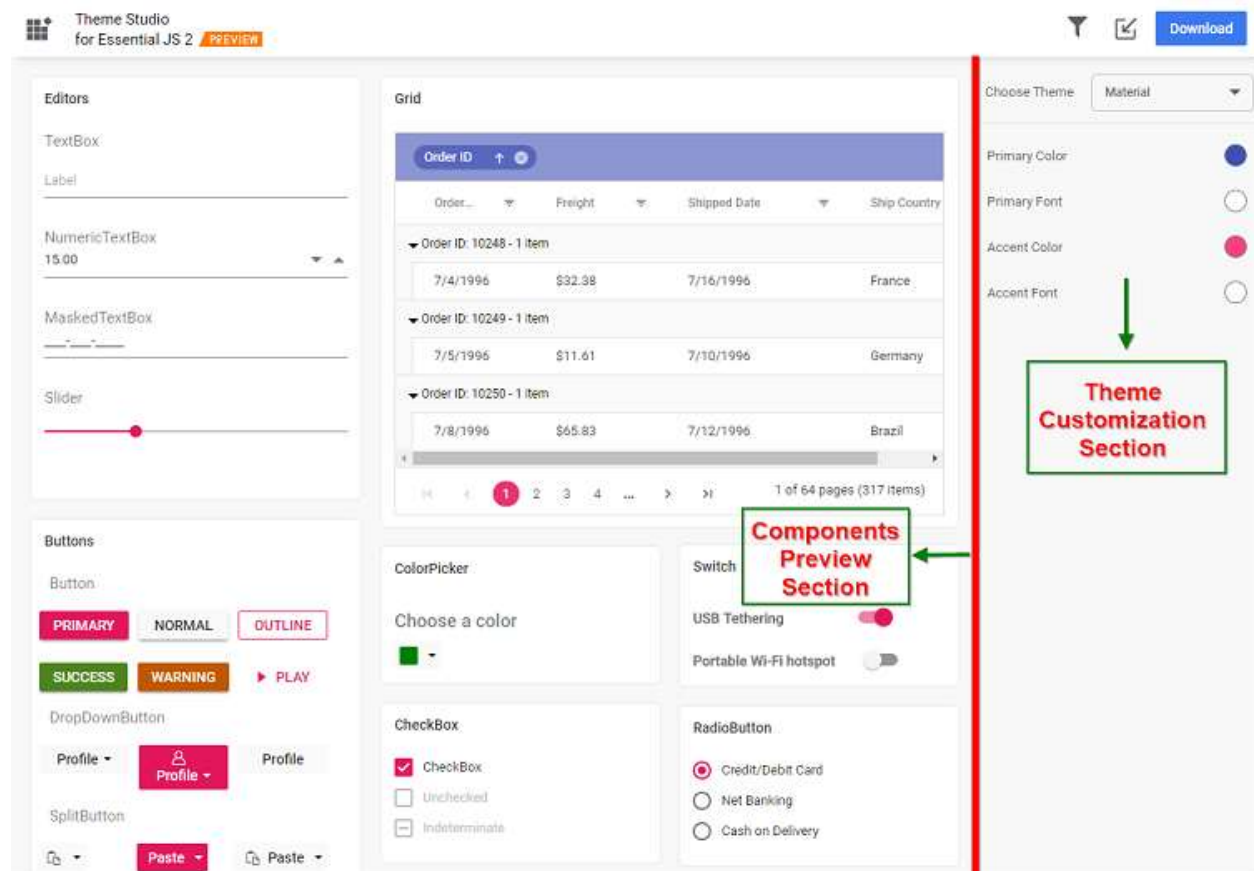
Theme Studio for Syncfusion Angular UI Components can be used to customize a new theme from an existing theme. It doesn't support with Data visualization components like Chart, Diagram, Gauge, Range Navigator, Maps.

Customizing Theme Color from Theme Studio

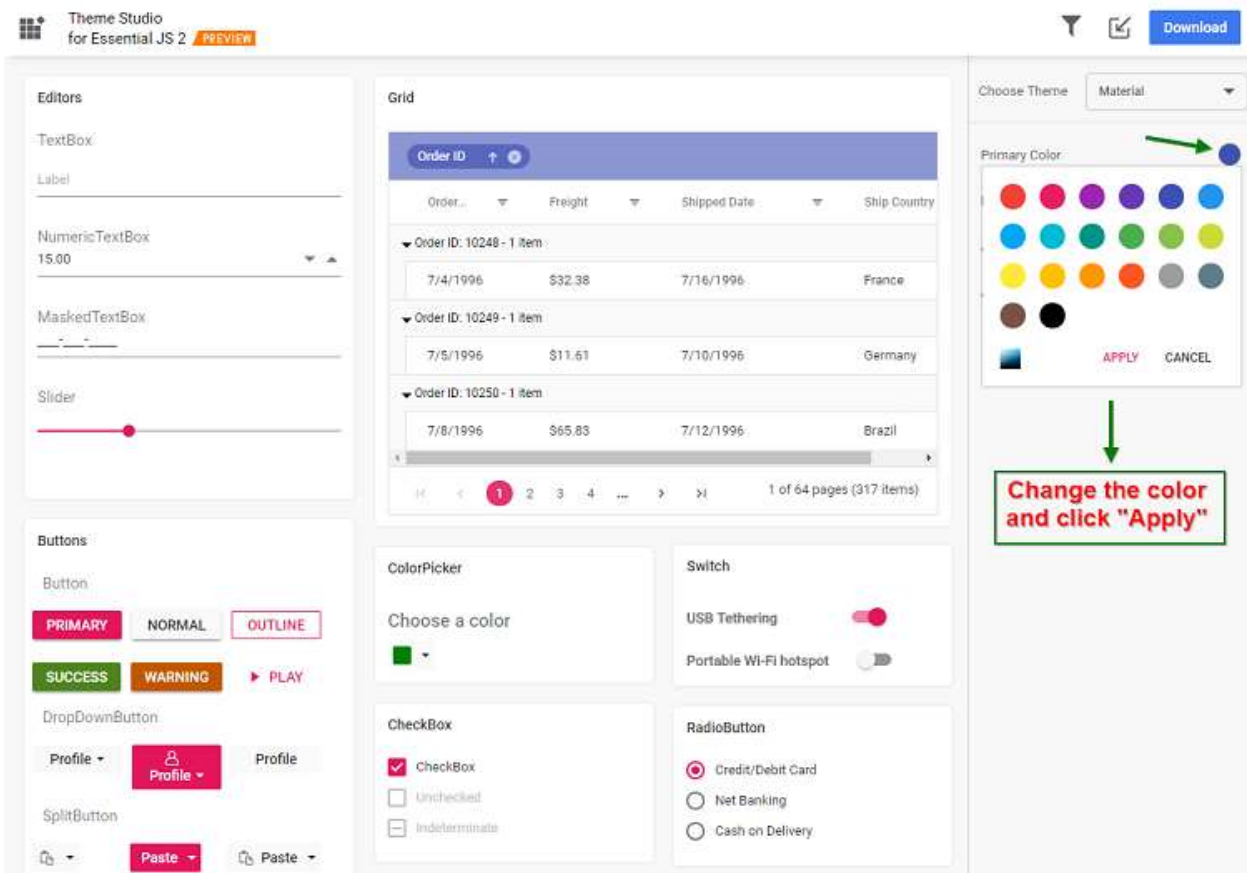
The Syncfusion Angular UI Components themes are developed under the SCSS environment. Each theme has a unique common variable list. When you change the common variable color code value, it will reflect in all the Syncfusion Angular UI Components. All the Syncfusion Angular UI Component styles are derived from these [theme-based common variables](#). This common variable list is handled inside the theme studio application for customizing theme-based colors.

Step 1: Navigate to the theme studio application at <https://ej2.syncfusion.com/themestudio/>.

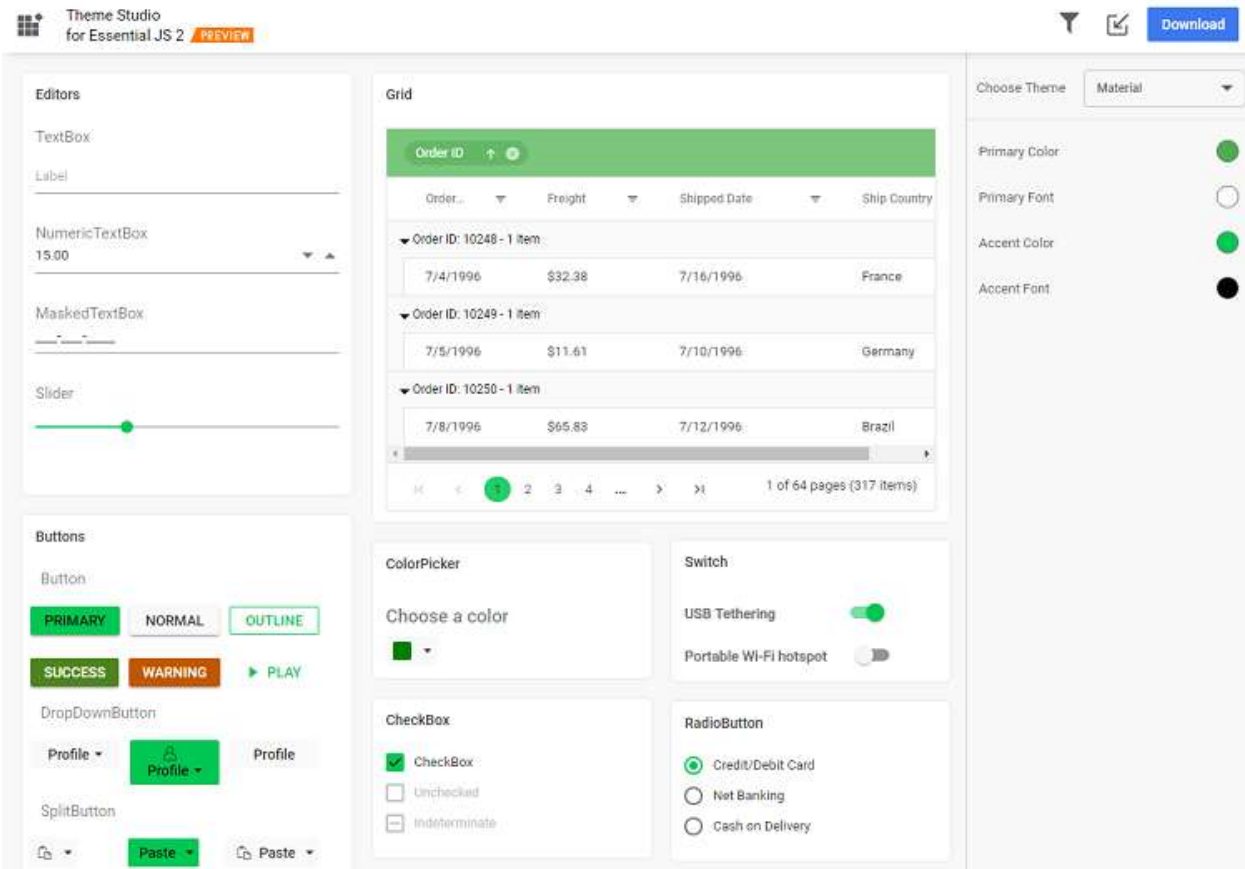
Step 2: The theme studio application page can be divided into two sections: the components preview section on the left, and the theme customization section on the right.



Step 3: Click the color pickers in the theme customization section to select your desired colors.



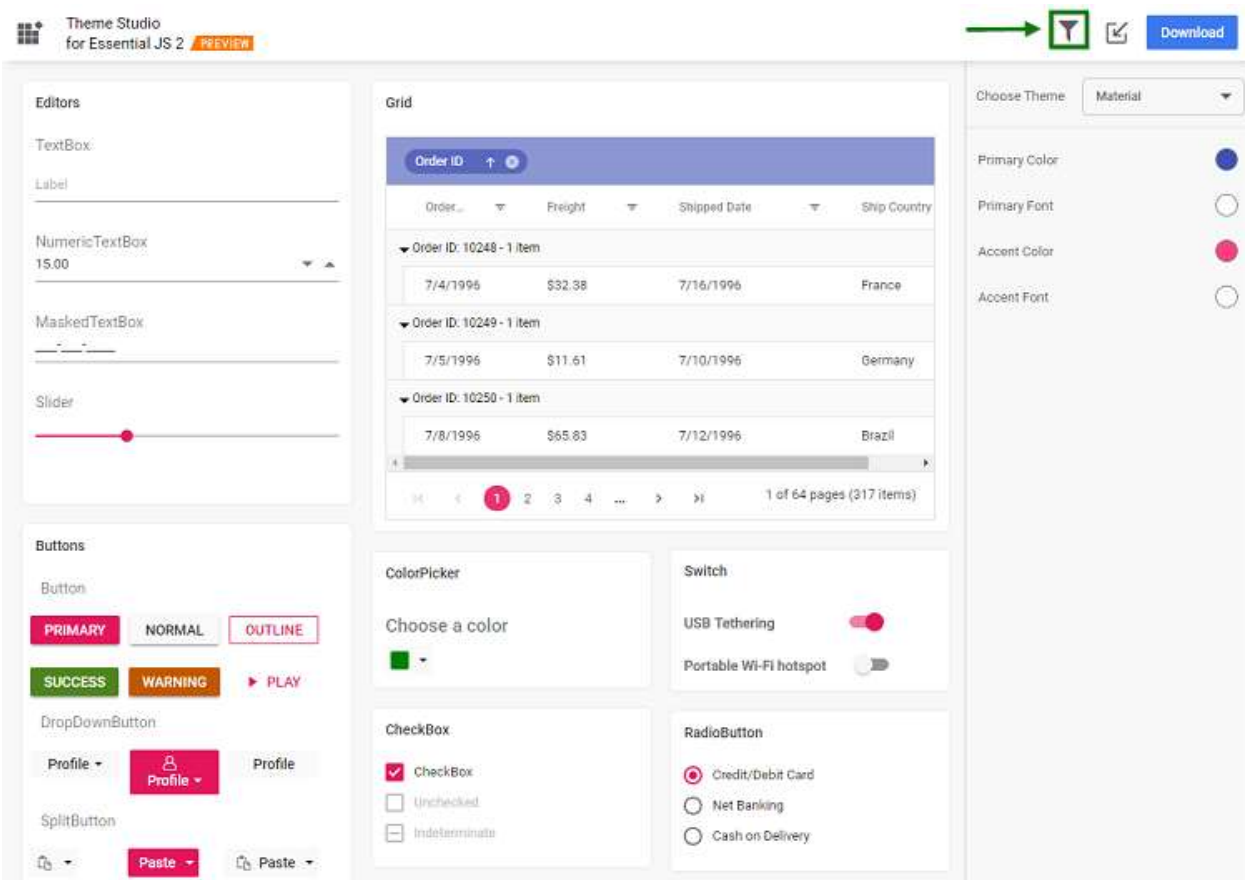
Step 4: After selecting colors with the color pickers, the Essential JS 2 components will have the newly selected colors applied to them in the components preview section.



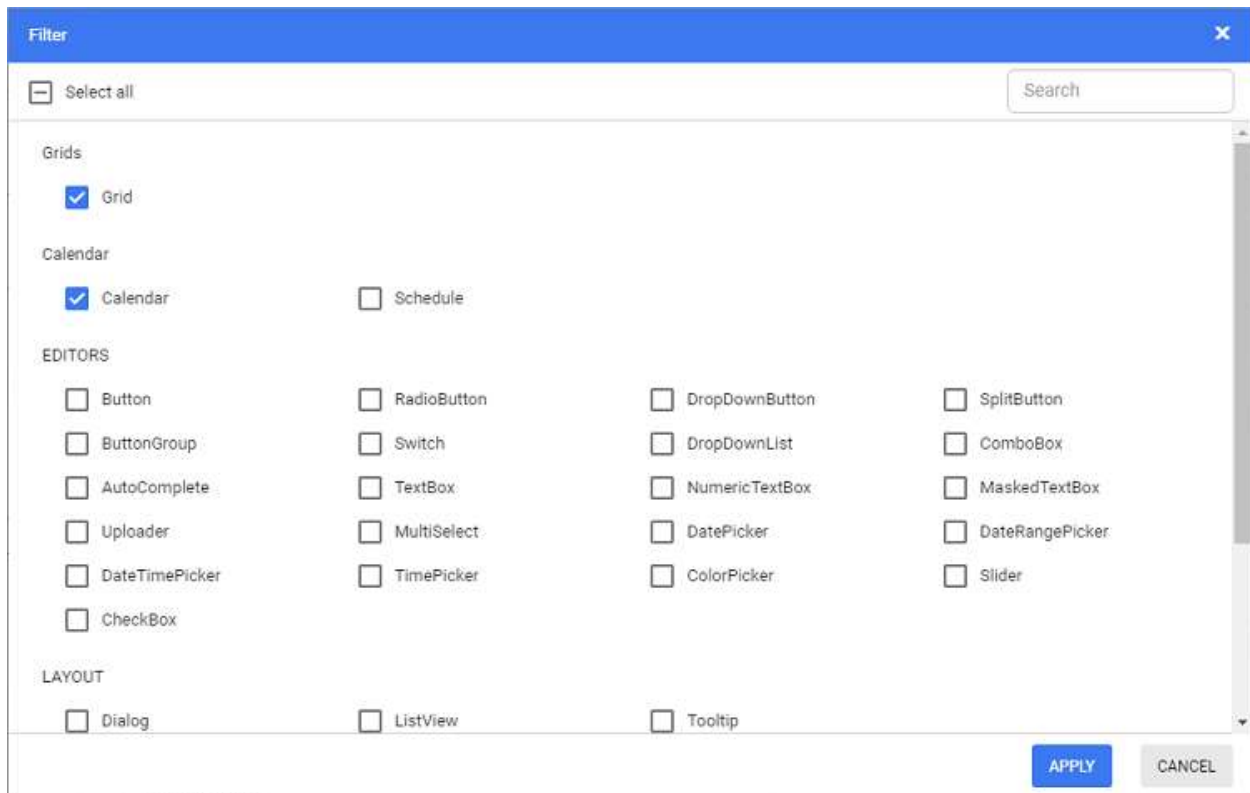
Filtering a Specific List of Components

Using the theme studio, you can apply custom themes to a list of specific components. This option is useful when you have integrated a selective list of Syncfusion Angular UI Components in your application. The theme studio will filter the selected components and customize the final output for those components' styles alone, reducing the final output file size.

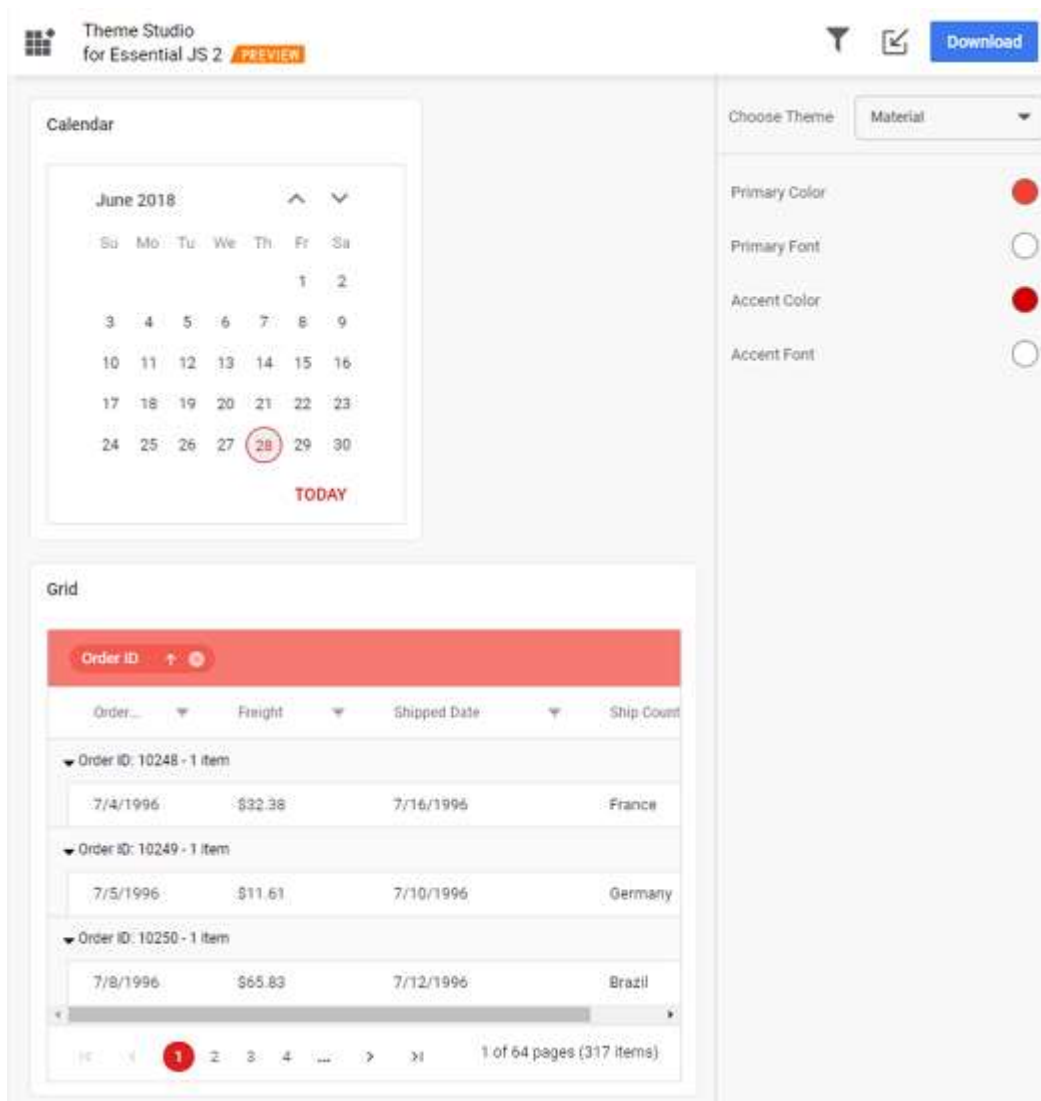
Step 1: Click the Filter icon in the top right corner and select the components whose theme you want to customize.



Step 2: Click the Apply button in the Filter dialog. Now, only the selected components will be rendered in the components preview section.



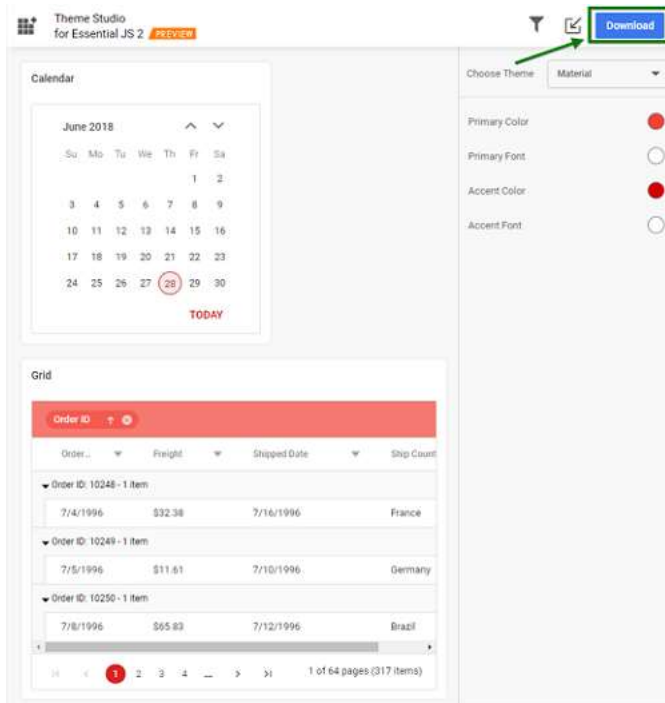
Step 3: Now you can customize the colors in the theme customization section for the components you selected.



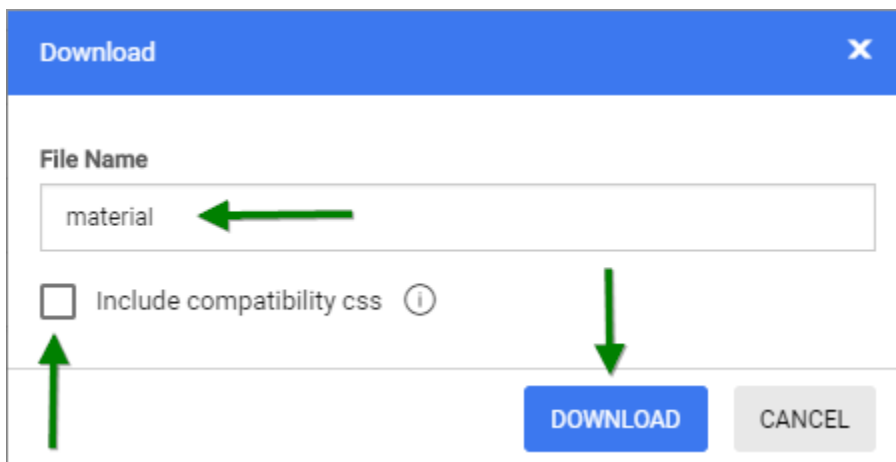
[Download the Customized Theme](#)

You can download the custom styles after customizing the theme colors.

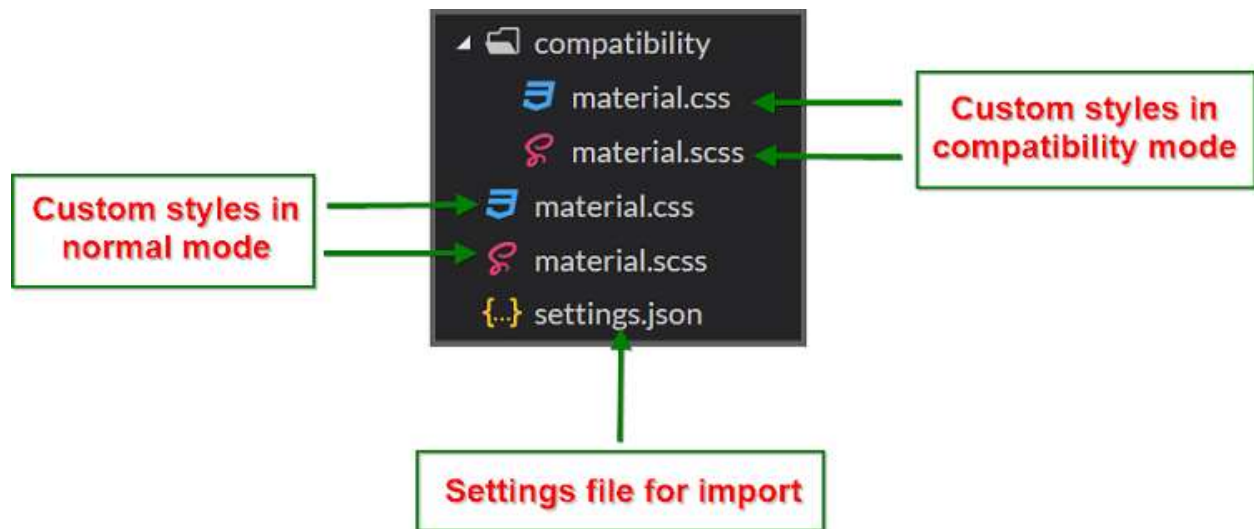
Step 1: Click the Download button in the top right corner. The Download dialog will open.



Step 2: Assign a theme name in the File Name field and click the Download button. If your application uses both Essential JS 1 and Essential JS 2 components, then select the Include compatibility CSS checkbox before downloading the theme. This option will generate the custom theme for Essential JS 2 compatibility styles, which are compatible as Essential JS 1 styles. Refer this [link](#) for more details about Essential JS 1 and Essential JS 2 compatibility.



Step 3: The download styles will come as a zip file that contains SCSS and CSS files for the selected Syncfusion Angular UI Components. The current settings are stored in the `settings.json` file.



Using Customized Theme in a Web Application

You can directly use the customized CSS file in the web application.

Step 1: Copy/paste the customized CSS file from the download folder into your application at any folder.
Example: `styles/{file-name}.css`.

Step 2: Refer the customized CSS file reference in the `index.html` or `shared/_layout.cshtml` main page head section.

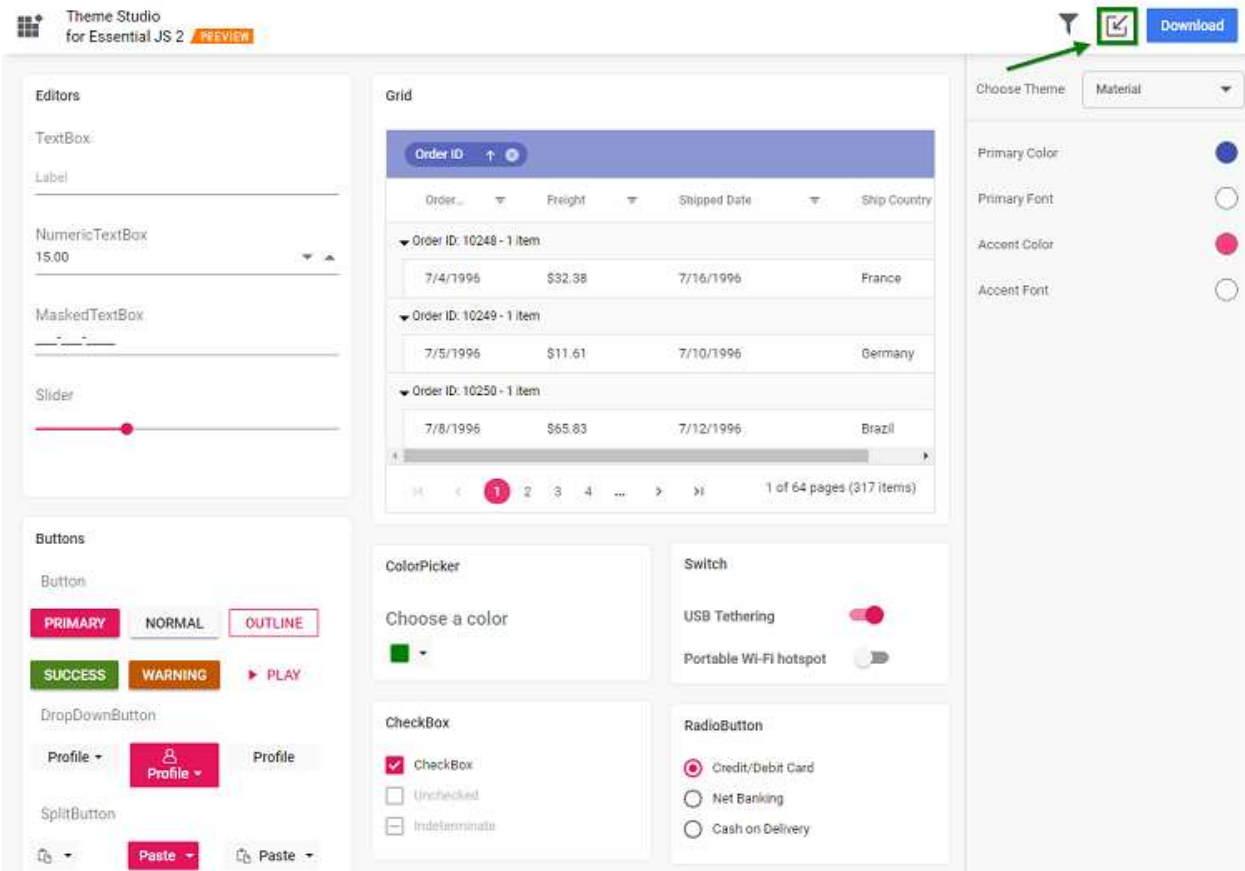
```
<head>
<link href="styles/{file-name}.css" rel="stylesheet"/>
</head>
```

If you are using Essential JS 1 and Essential JS 2 components in a same web application, then you have to copy/paste the customized CSS file from the `compatibility` folder in the download location.

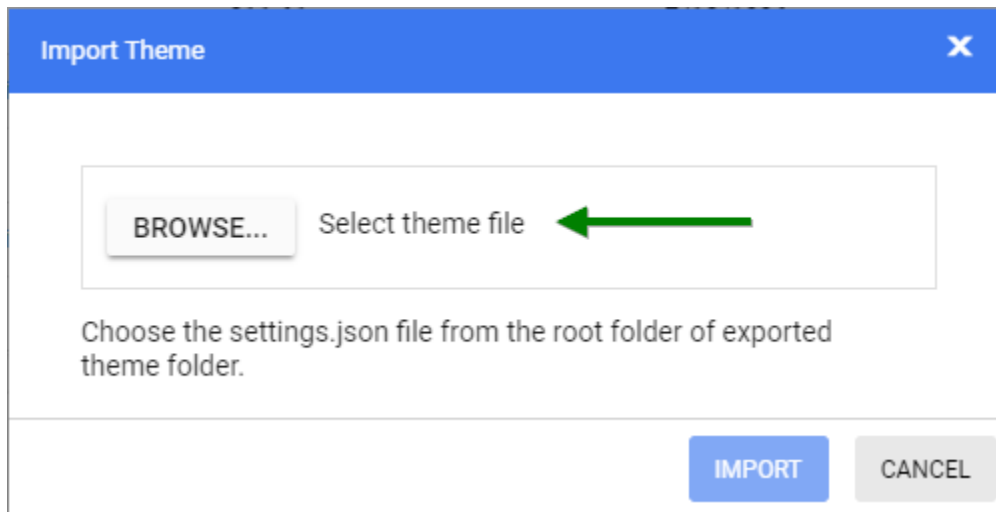
Import Previously Changed Settings into the Theme Studio

When you want to change your application theme and UI design in the future, you won't need to customize the Syncfusion Angular UI Components from scratch in the theme studio. Just import the old `settings.json` file to review and update your stored settings in the theme studio application.

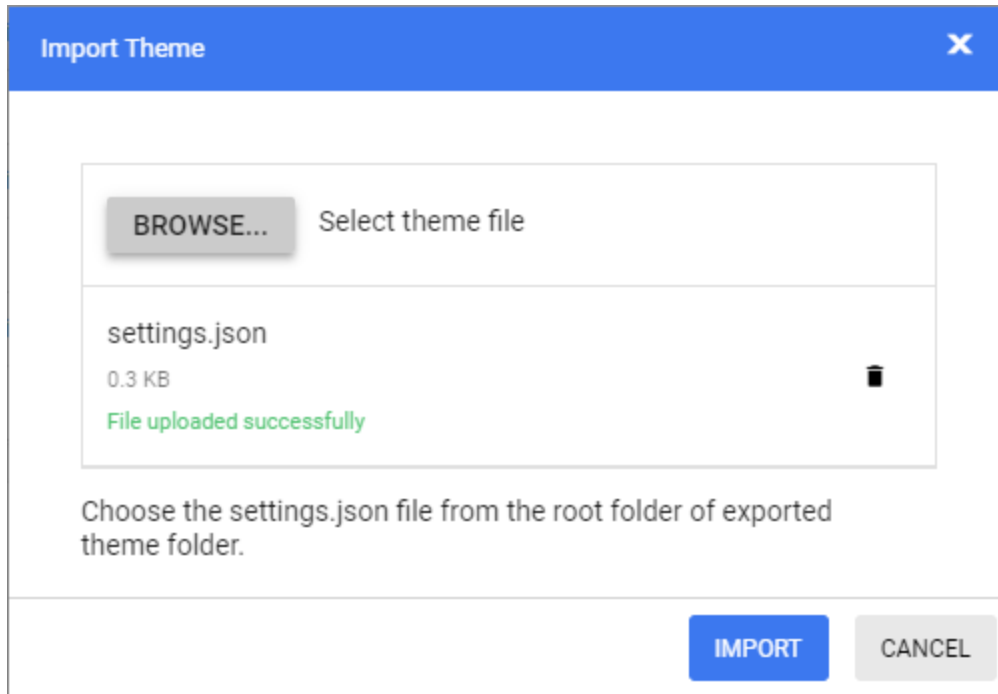
Step 1: Click the Import icon in the top right corner.



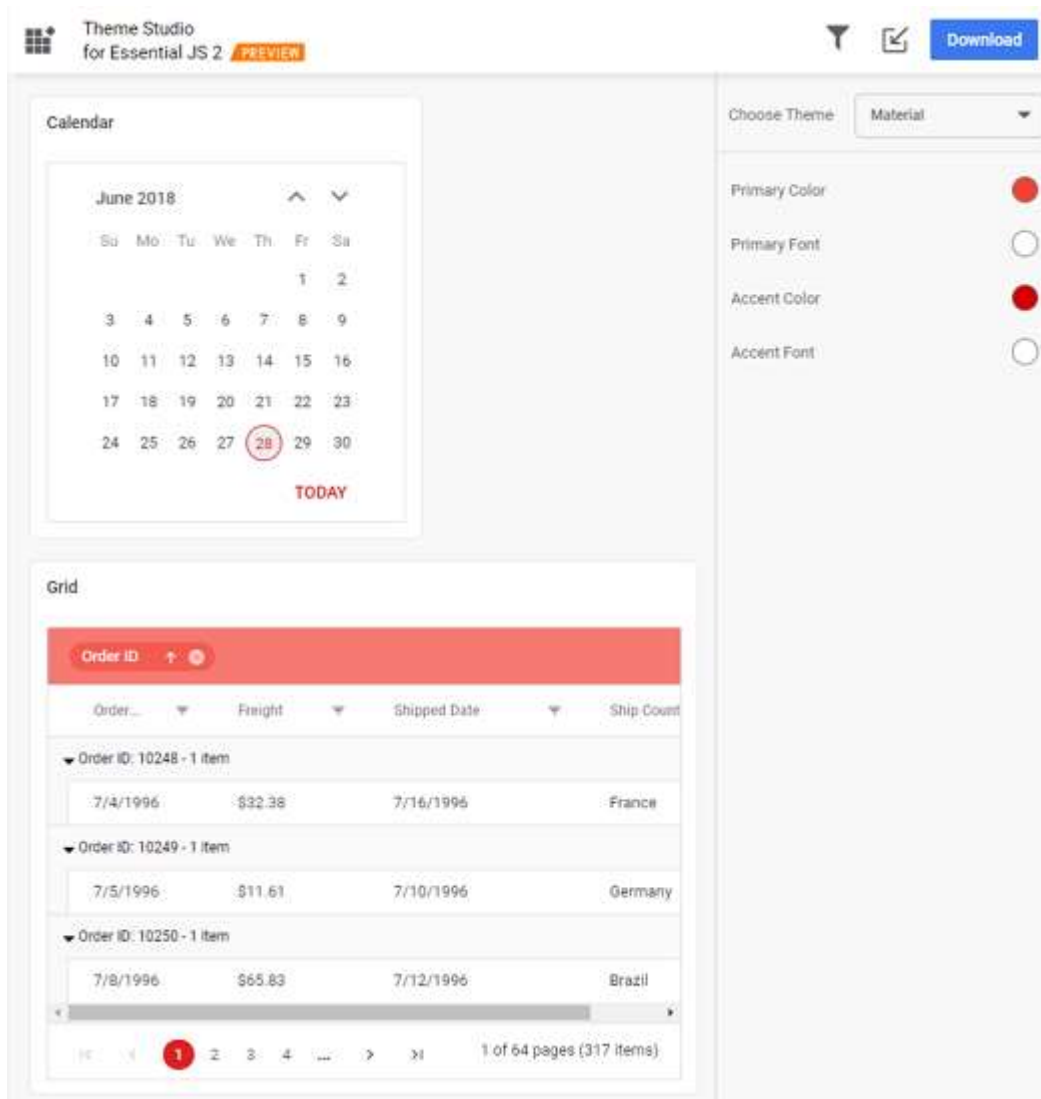
Step 2: The Import Theme dialog will open. Click the Browse button and select a `settings.json` file you exported previously.



Step 3: Click the Import button.



Step 4: The stored data will be reflected in the theme studio application. Now you can change the theme colors based on your latest design and export the theme again.



Step 5: The exported file will contain your latest changes. You can just replace the older custom style with the newer one to refresh your application.

Tailwind Theme

The Syncfusion Angular UI Components provides tailwind theme support for EJ2 components by considering the UI design principles and theme colors. Syncfusion developed tailwind theme by using the design principles of tailwind ui.

Material 3 Theme with CSS Variables

[Material 3](#) is the latest version of Google's open-source design system, Material Design. It's the successor to [Material 2](#), which is a popular design system used in many apps. It has been specifically designed to align seamlessly with the new visual style and system UI introduced in Android 12 and above.

[CSS variables](#), also known as custom properties, are entities defined by CSS authors that contain specific values that can be reused throughout a CSS file. They are identified by their name, which must begin with two hyphens (--) followed by an identifier. These variables can be assigned any valid CSS value, such as colors, lengths, or fonts. To retrieve the value of a CSS variable, the `var()` function is used.

Material 3 - Syncfusion Angular Components

Syncfusion has introduced the Material 3 theme across all EJ2 Components, featuring both **light** and **dark** variants. This theme utilizes **CSS variables** to allow easy customization of Component colors in CSS format. With this implementation, users can seamlessly switch between light and dark color schemes, providing a flexible solution to meet their preferences and application needs.

Kindly note that in the Material 3 theme, Syncfusion utilizes CSS variables with `rgb()` values for color variables. The use of hex values in this context may lead to improper functionality. For example, in previous versions of the Material theme or other themes, the primary color variable was defined as follows: `$primary: #6200ee;`. However, in the Material 3 theme, the primary color variable is defined as follows: `--color-sf-primary: 98, 0, 238;`.

How does Syncfusion Material 3 theme utilize CSS Variables?

Syncfusion's Material 3 theme incorporates support for CSS variables, utilizing `rgb()` values for customizing colors. For more information you can refer this [documentation](#) for color variables of material 3 theme.

```
`CSS
:root {
--color-sf-black: 0, 0, 0;
--color-sf-white: 255, 255, 255;
--color-sf-primary: 103, 80, 164;
--color-sf-primary-container: 234, 221, 255;
--color-sf-secondary: 98, 91, 113;
--color-sf-secondary-container: 232, 222, 248;
--color-sf-tertiary: 125, 82, 96;
--color-sf-tertiary-container: 255, 216, 228;
--color-sf-surface: 255, 255, 255;
--color-sf-surface-variant: 231, 224, 236;
--color-sf-background: var(--color-sf-surface);
--color-sf-on-primary: 255, 255, 255;
--color-sf-on-primary-container: 33, 0, 94;
--color-sf-on-secondary: 255, 255, 255;
--color-sf-on-secondary-container: 30, 25, 43;
--color-sf-on-tertiary: 255, 255, 255;
}
```

How to get the Material 3 theme?

To access the Material 3 theme provided by Syncfusion, you have two primary options,

- Package
- CDN links

| | Light | Dark |

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

| Package | [Material 3 Light](#) | [Material 3 Dark](#) |

| CDN | [Material 3 Light](#) | [Material 3 Dark](#) |

Color Customization in Material 3

CSS variables allows you to dynamically change color values in real-time using JavaScript. This flexibility enables you to create interactive experiences where colors can adjust based on user interactions or other dynamic factors.

Customization using CSS

Here you can find the example for Material 3 customization using Css class.

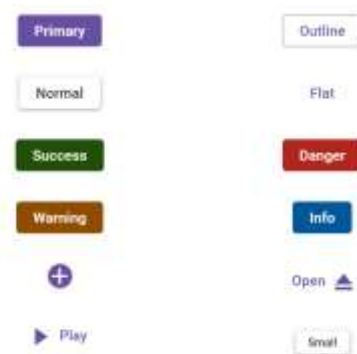
APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { enableRipple } from '@syncfusion/ej2-base'
import { FormsModule } from '@angular/forms'
import { Component } from '@angular/core';
@Component({
  imports: [
    ButtonModule,
    FormsModule
  ],
  standalone: true,
  selector: 'app-root',
  styleUrls:['./style.css'],
  template:`<div>
    <!-- Primary Button - Used to represent a primary action. -->
    <button ej2-button cssClass="e-primary">Button</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));
```

Default primary value



```

:root {
  --color-sf-black: 0, 0, 0;
  --color-sf-white: 255, 255, 255;
  --color-sf-primary: 255, 255, 255;
  --color-sf-primary-container: 234, 221, 250;
  --color-sf-secondary: 98, 81, 113;
  --color-sf-secondary-container: 232, 222, 248;
  --color-sf-tertiary: 125, 82, 96;
  --color-sf-tertiary-container: 255, 216, 228;
  --color-sf-surface: 279, 255, 255;
  --color-sf-surface-variant: 231, 224, 236;
  --color-sf-background: var(--color-sf-surface);
  --color-sf-on-primary: 255, 255, 255;
  --color-sf-on-primary-container: 33, 0, 94;
  --color-sf-on-secondary: 255, 255, 255;
  --color-sf-on-secondary-container: 30, 25, 43;
  --color-sf-on-tertiary: 255, 255, 255;
  --color-sf-on-tertiary-container: 55, 11, 30;
  --color-sf-on-surface: 28, 27, 31;
  --color-sf-on-surface-variant: 73, 69, 78;
  --color-sf-on-background: 38, 27, 31;
  --color-sf-outline: 123, 110, 128;
  --color-sf-outline-variant: 196, 196, 197;
  --color-sf-shadow: 0, 0, 0;
  --color-sf-surface-tint-color: 100, 90, 164;
  --color-sf-inverse-surface: 49, 48, 51;
}

```

Customized primary value



```

:root {
  --color-sf-black: 0, 0, 0;
  --color-sf-white: 255, 255, 255;
  --color-sf-primary: 30, 25, 43;
  --color-sf-primary-container: 234, 221, 250;
  --color-sf-secondary: 98, 81, 113;
  --color-sf-secondary-container: 232, 222, 248;
  --color-sf-tertiary: 125, 82, 96;
  --color-sf-tertiary-container: 255, 216, 228;
  --color-sf-surface: 279, 255, 255;
  --color-sf-surface-variant: 231, 224, 236;
  --color-sf-background: var(--color-sf-surface);
  --color-sf-on-primary: 255, 255, 255;
  --color-sf-on-primary-container: 33, 0, 94;
  --color-sf-on-secondary: 255, 255, 255;
  --color-sf-on-secondary-container: 30, 25, 43;
  --color-sf-on-tertiary: 255, 255, 255;
  --color-sf-on-tertiary-container: 55, 11, 30;
  --color-sf-on-surface: 28, 27, 31;
  --color-sf-on-surface-variant: 73, 69, 78;
  --color-sf-outline: 123, 110, 128;
  --color-sf-outline-variant: 196, 196, 197;
  --color-sf-shadow: 0, 0, 0;
  --color-sf-surface-tint-color: 100, 90, 164;
  --color-sf-inverse-surface: 49, 48, 51;
}

```

With this CSS variable support, you can effortlessly customize the color variable values for Syncfusion Angular Components.

Material 3 Light Theme

Syncfusion has implemented the Material 3 theme, offering both Light and Dark variants. In the Material 3 light theme, there are distinct class variables for both light and dark modes, providing flexibility for seamless switching between the two modes within your application.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ButtonModule, CheckBoxModule } from '@syncfusion/ej2-angular-buttons'
import { enableRipple } from '@syncfusion/ej2-base'
import { SwitchModule } from '@syncfusion/ej2-angular-buttons'
import { FormsModule } from '@angular/forms'
import { Component } from '@angular/core';
@Component({
  imports: [
    ButtonModule,
    CheckBoxModule,
    SwitchModule,
    FormsModule
  ],
  standalone: true,
  selector: 'app-root',
})

```

```

    styleUrls:['./style.css'],
    template:`<div [ngClass]="{'e-dark-mode': isChecked, 'dark':
isChecked}">
      <ejs-checkbox label="Enable Darkmode" (change)="toggleCheckbox()"></ejs-
checkbox><br/>

      <!-- Primary Button - Used to represent a primary action. -->
      <button ejs-button cssClass="e-primary">Button</button>

      <!-- Success Button - Used to represent a positive action. -->
      <button ejs-button cssClass="e-success">Button</button>

      <!-- Info Button - Used to represent an informative action -->
      <button ejs-button cssClass="e-info">Button</button>

      <!-- Warning Button - Used to represent an action with caution. -->
      <button ejs-button cssClass="e-warning">Button</button>

      <!-- Danger Button - Used to represent a negative action. -->
      <button ejs-button cssClass="e-danger">Button</button>
    </div>`
  })
  export class AppComponent {
    public className:string="";
    public checked:boolean=true;
    public isChecked = false;
    toggleCheckbox() {
      this.isChecked = !this.isChecked;
      if (this.isChecked) {
        document.body.classList.add('dark');
      }
      else{
        document.body.classList.remove('dark');
      }
    }
  }
}

```

MAIN.TS

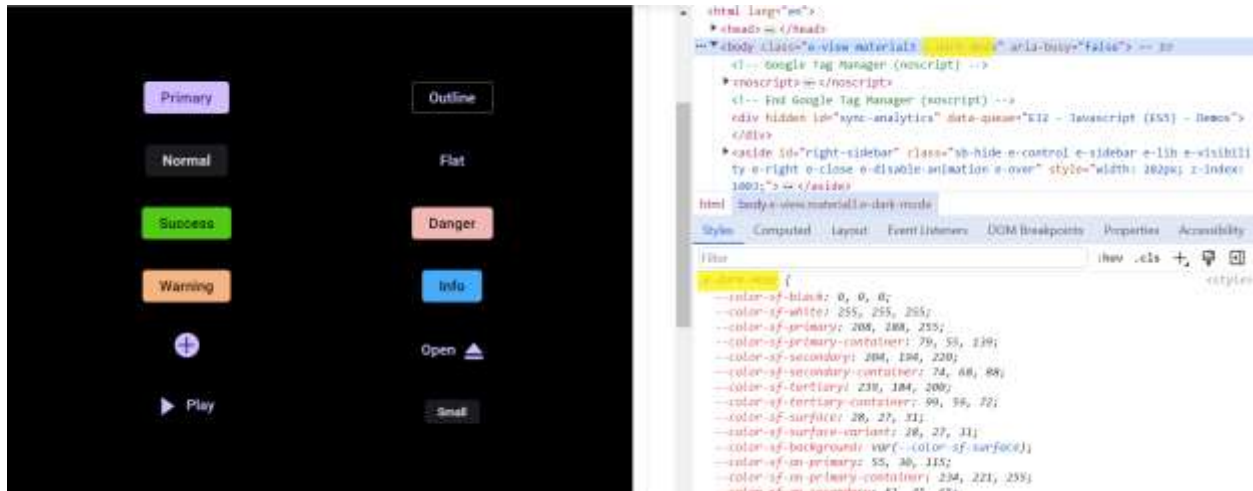
```

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

```

How to switch dark mode?

With CSS variable support, transitioning between light and dark theme modes has become effortless. To activate dark mode, just append the `e-dark-mode` class to the body section of your application. Once applied, the theme seamlessly switches to dark mode. Please refer to the example image below for visual guidance.



ThemeStudio Application

The ThemeStudio application now includes seamless integration with the Material 3 theme, offering a comprehensive solution for customization requirements. This enhancement enables users to effortlessly customize and personalize their themes.

Access the Syncfusion ThemeStudio application, featuring the Material 3 theme, via the following link:

[Link to Syncfusion ThemeStudio with Material 3 Theme](#)

How To

Loading themes without internet access

Syncfusion incorporates the Material theme as a default built-in theme for all its components. By default, material theme refers to Google fonts, which necessitates internet access. However, this may not be suitable for applications that operate without an internet connection. To address this limitation, Syncfusion provides `offline-theme` file to reference Material styles with local fonts, ensuring seamless usage even in environments without internet connectivity.

Material theme package

The [ej2-material-theme](#) package enables users to reference styles without the online font through `offline-theme` file as an NPM package. To utilize this package, follow the steps provided below.

Adding the material theme package

Run the following command to install the `ej2-material-theme` package in the application

```
`bash
npm i @syncfusion/ej2-material-theme
`
```

If you want to utilize [ej2-material-dark](#) theme run the below command

```
`bash
npm i @syncfusion/ej2-material-dark-theme
`
```

Importing styles

Once installing the package, import the styles in your application by adding a link path to your styles folder. Make sure the path does not include references to the Roboto font.

```
`bash
```

```
@import '../node_modules/@syncfusion/ej2-material-theme/styles/offline-theme/material.css';
```

```
,
```

Loading offline theme for tailwind

Similar to our **material theme**, we have offline-Theme designed specifically for our **tailwind users**. This feature allows you to experience the full range of Tailwind theme benefits without the need for online Google Font references.

Tailwind theme package

Same as **material-theme** package [ej2-tailwind-theme](#) enables users to reference styles without the online font through a offline-theme file as an NPM package. To utilize this package, follow the steps provided below.

Adding the tailwind theme package

Run the following command to install the **ej2-tailwind-theme** package in the application.

```
`bash
```

```
npm install @syncfusion/ej2-tailwind-theme
```

```
,
```

If you want to utilize **ej2-tailwind-dark** theme run the below command

```
`bash
```

```
npm install @syncfusion/ej2-tailwind-dark-theme
```

```
,
```

Importing styles

Once installing the package, import the styles in your application by adding a link path to your styles folder. Make sure the path does not include references to the Roboto font.

```
`bash
```

```
@import '../node_modules/@syncfusion/ej2-tailwind-theme/styles/offline-theme/tailwind.css';
```

```
,
```

Syncfusion recognizes the need for applications designed to function without internet connectivity. To bridge this gap, It provides a offline-theme file that reference theme with local fonts. This approach ensures a smooth user experience, making our components adaptable for applications in environments with limited or no internet connectivity.

Common

Accessibility in Syncfusion Angular Components

Accessibility overview

Accessibility in components refers to the practice of designing and building user interface elements in a way that makes them accessible to users with disabilities. This can include a variety of things, such as making sure that text is high-contrast and easy to read, providing alternative text for images, and designing controls and interactions that can be used with a keyboard or assistive technology.

Accessibility standards

The component is said to be accessible when it is constructed in accordance with certain standards that are required to make it accessible.

The accessibility of the components consists of the following standards and aspects:

- [ADA](#) - A law to ensure that people with disabilities have the same opportunities and access as people without disabilities.
- [WCAG 2.2](#) - The Web Content Accessibility Guidelines (WCAG) provide guidelines developed by the World Wide Web Consortium (W3C) to ensure web content is accessible to people with disabilities. [WCAG 2.2](#) establishes a framework of accessibility principles and their associated success criteria. The level of accessibility conformance achieved by a web application is determined by the extent to which it meets these success criteria, categorized into three levels: A, AA, and AAA.
- [Section 508](#) - It is a set of guidelines for making electronic and information technology (EIT) accessible to people with disabilities. These standards apply to federal agencies in the United States, and they are based on the Web Content Accessibility Guidelines (WCAG).
- [WAI-ARIA](#) - WAI-ARIA stands for "Web Accessibility Initiative - Accessible Rich Internet Applications." It is a set of technical specifications and guidelines developed by the World Wide Web Consortium (W3C) as part of the Web Accessibility Initiative (WAI). WAI-ARIA is designed to enhance the accessibility of dynamic web content, particularly web applications and rich internet applications (RIAs), for people with disabilities. WAI-ARIA provides a set of roles, states, and properties that can be added to HTML elements to provide additional context and information about the purpose and behavior of those elements. This can help assistive technologies better understand and interpret web content and interact with web applications.
- [Keyboard navigation](#) - It refers to the ability to use a keyboard to interact with and navigate through a user interface. It is an important aspect of web accessibility, as it allows people who cannot use a mouse or other pointing device to access and use web content and applications.

Syncfusion Angular components adhere to these established standards.

Accessibility compliance

The accessibility support provided by Syncfusion Angular components is based on a collection of methodologies for adhering to and applying [recognized standards and technical specifications](#) to ensure an intuitive experience for people with disabilities.

There are several methodologies of accessibility validation that can be performed on the Syncfusion Angular components, such as:

- The [WAI-ARIA patterns](#) are followed by the Syncfusion Angular components to enable appreciable accessibility.
- Each Angular component is subjected to manual testing with a screen reader and also automated test cases to ensure the component's required attributes.
- Attributes are allocated and updated correctly during interaction as well. Each component has been assigned a distinct [Role](#) attribute and its own set of ARIA attributes defined by the [WCAG 2.2](#) specification.

In addition to the methodologies mentioned above, Syncfusion Angular components are constructed to support the following accessibility aspects.

Screen reader support

A screen reader allows people who are blind or visually impaired to use a computer by reading aloud the text that is displayed on the screen. Syncfusion Angular components followed the [WAI-ARIA](#) standards to work properly in the screen readers such as [Narrator](#) for Windows and [Embedded VoiceOver](#) for MAC.

Right-To-Left support

Right-to-Left (RTL) support allows applications to effectively communicate with users who use languages that are written from right to left, such as Arabic, Hebrew, etc. Syncfusion Angular components support the Right-to-Left feature. Refer to the [Right-to-Left documentation](#) to learn more about this support.

Color contrast

Syncfusion Angular components come equipped with [predefined themes](#) that guarantee the presence of satisfactory color contrast, benefiting individuals with visual impairments.

Mobile device support

Syncfusion Angular components are more user-friendly and accessible to individuals using mobile devices, including those with disabilities. These are designed to be responsive, adaptable to various screen sizes and orientations, and touch-friendly.

Keyboard navigation support

Syncfusion Angular components support keyboard navigation, allowing users who rely on alternate methods to effortlessly navigate and interact with the component.

Ensuring accessibility

Ensuring the accessibility of Syncfusion Angular components is crucial for making the product inclusive and user-friendly for individuals with disabilities. This process involves various types of accessibility testing, including:

- **Automated testing:** The Syncfusion Angular component's accessibility levels are ensured through an [accessibility-checker](#) and [axe-core](#) software tools.
- **Manual testing:** This type of testing involves manually evaluating the Syncfusion Angular components. During manual accessibility testing, testers will ensure accessibility using the screen readers such as [Narrator](#) for Windows and [Embedded VoiceOver](#) for MAC.

Syncfusion Angular components will keep improving when there is anything required. It also involves client feedback to make the component more accessible.

Accessibility support for specific components

Consult the component-specific documents below for detailed information about how Syncfusion Angular components ensure compliance with accessibility standards, encompassing Section 508, WAI-ARIA, WCAG 2.2, keyboard navigation, and more.

<style>

table

{

border:0 !important;

line-height: 2!important;

}


```
tr
{
border:0 !important;
}
td
{
border:0 !important;
vertical-align: top;
}
.controlanchorlink
{
text-decoration: none !important;
font-size: 14px !important;
text-align: left !important;
padding: 5px 0px;
letter-spacing: 1px;
}
.controlcategory
{
font-size: 14px !important;
text-align: left !important;
font-weight: bold !important;
letter-spacing: 0.7px;
}
}
```

| GRIDS | DATA
VISUALIZATION | CALENDARS | DROPDOWNS |
|-----------------------------|------------------------------------|---------------------------------|--------------------------------------|
| DataGrid | Accumulation Chart | Scheduler | AutoComplete |
| Pivot Table | Charts | Calendar | ComboBox |
| Tree Grid | 3D Chart | DatePicker | DropDown List |
| Spreadsheet | Stock Chart | DateRangePicker | DropDown Tree |
| | Circular Gauge | DateTime Picker | Multiselect DropDown |
| | | | |

| | | | |
|----------------------------------|----------------------------------|--------------------------------------|------------------------------|
| FILE VIEWERS & EDITORS | Diagram | TimePicker | Mention |
| | HeatMap Chart | Gantt Chart | ListBox |
| In-place Editor | Linear Gauge | INPUTS | NAVIGATION |
| PDF Viewer | Maps | TextBox | Accordion |
| RichTextEditor | Range Selector | Input Mask | AppBar |
| Word Processor | Smith Chart | Masked TextBox | Breadcrumb |
| Image Editor | Sparkline Charts | Numeric TextBox | Carousel |
| LAYOUT | TreeMap | Radio Button | Context Menu |
| Dialog | Bullet Chart | CheckBox | Menu Bar |
| ListView | Kanban | Color Picker | Sidebar |
| Tooltip | BUTTONS | File Upload | Tabs |
| Splitter | Button | Range Slider | Toolbar |
| Dashboard Layout | Button Group | Toggle Switch Button | Ribbon |
| | Dropdown Menu | Signature | TreeView |
| | Progress Button | Rating | File Manager |
| | Split Button | FORMS | Pager |
| | Chips | Query Builder | Stepper |
| | FAB | | NOTIFICATION |
| | Speed Dial | | Message |
| | | | Toast |
| | | | Progress Bar |
| | | | Skeleton |

Animation support in Syncfusion Angular Components

The **Animation** is used to perform animation effects on HTML elements by running a sequence of frames. It can be used to enhance the user experience.

Syncfusion [Animation](#) library supports animating the HTML elements using the [animate](#) method. This method adds the `e-animate`, `e-animation-id` attributes, and CSS style to the HTML element and removes them after the animation effect is completed.

Animation effects

An animation effect refers to the visual change that occurs over a period of time in HTML elements. The [Animation](#) library supports different types of animation [effects](#). The [name](#) property is used to specify the animation effect of an animation.

Here is an example code snippet using the **FadeOut** and **ZoomOut** animation effects:

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Component, ViewChild } from '@angular/core';
import { Animation } from '@syncfusion/ej2-base';
@Component({
  standalone: true,
  selector: 'app-root',
  template: `
    <div #element1 class='animation1'></div>
    <div #element2 class='animation2'></div>
  `
})
export class AppComponent {
  @ViewChild('element1',{static: false})element1: any;
  @ViewChild('element2',{static: false})element2: any;
  ngAfterViewInit() {
    let animation: Animation = new Animation({});
    animation.animate(this.element1.nativeElement, { name: 'FadeOut' });
    animation.animate(this.element2.nativeElement, { name: 'ZoomOut' });
  }
}
```

MAIN.TS

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

Animation duration

Animation [duration](#) is the animation property that specifies the length of time that an animation should take to complete. The animation duration is specified in milliseconds (ms) and determines the total amount of time that an animation should run.

For example, if an animation has a duration of 2 seconds, it will take 2 seconds to complete from start to finish. The duration of an animation affects the overall pace of the animation and can be adjusted to match the desired speed and style of the animation.

The value of the animation duration can be adjusted to change the speed of the animation, with shorter durations resulting in faster animations and longer durations resulting in slower animations.

Here is an example code snippet using the animation effects with a duration of **5000** milliseconds:

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Component, ViewChild } from '@angular/core';
import { Animation } from '@syncfusion/ej2-base';
@Component({
  standalone: true,
  selector: 'app-root',
```

```

template:
  <div #element1 class='animation1'></div>
  <div #element2 class='animation2'></div>
})
export class AppComponent {
  @ViewChild('element1',{static: false})element1: any;
  @ViewChild('element2',{static: false})element2: any;
  ngAfterViewInit() {
    let animation: Animation = new Animation({ duration: 5000 });
    animation.animate(this.element1.nativeElement, { name: 'FadeOut' });
    animation.animate(this.element2.nativeElement, { name: 'ZoomOut' });
  }
}

```

MAIN.TS

```

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

```

Animation delay

The animation [delay](#) is the animation property that specifies the amount of time to wait before starting an animation. The animation delay is specified in milliseconds (ms) and determines the amount of time that should elapse before an animation begins.

For example, if an animation has a delay of 2 seconds, it will wait for 2 seconds before starting. This can be useful in creating more complex animations, where multiple elements are animated in sequence, or in creating animations that start only after a user interaction has taken place.

Here is an example code snippet using the animation effects with a delay of **2000** milliseconds:

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Component, ViewChild } from '@angular/core';
import { Animation } from '@syncfusion/ej2-base';
@Component({
  standalone: true,
  selector: 'app-root',
  template: `
    <div #element1 class='animation1'></div>
    <div #element2 class='animation2'></div>
  `
})
export class AppComponent {
  @ViewChild('element1',{static: false})element1: any;
  @ViewChild('element2',{static: false})element2: any;
  ngAfterViewInit() {
    let animation: Animation = new Animation({ delay: 2000, duration: 5000
  });
    animation.animate(this.element1.nativeElement, { name: 'FadeOut' });
    animation.animate(this.element2.nativeElement, { name: 'ZoomOut' });
  }
}

```

```
}
}
```

MAIN.TS

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

Enable or disable animation globally

Enable or disable animation for all Angular components globally by using the `setGlobalAnimation` method with one of the below options:

- `GlobalAnimationMode.Enable` - Enables the animation for all components, regardless of the individual component's animation settings.
- `GlobalAnimationMode.Disable` - Disables the animation for all components, regardless of the individual component's animation settings.
- `GlobalAnimationMode.Default` - Animation is enabled or disabled based on the component's animation settings.

In the below code snippet, animation is disabled.

~/SRC/SRC/APP.COMPONENT.TS

```
import { GlobalAnimationMode, setGlobalAnimation } from "@syncfusion/ej2-base";
setGlobalAnimation(GlobalAnimationMode.Disable);
```

`setGlobalAnimation` method controls script-level animations only, and it is not applicable for direct CSS-level animations (animations defined from CSS classes or properties).

Deployment

CDN

The CDN links are provided individually for all the scripts and style sheets of Syncfusion Angular UI Components.

The CDN links are provided to following files in the package

1. UMD Files
2. CSS Files

The latest minified versions of all UMD and CSS files are available in CDN:

- <https://cdn.syncfusion.com/ej2/RELEASEVERSION/PACKAGENAME/dist/PACKAGENAME.umd.min.js>
- <https://cdn.syncfusion.com/ej2/RELEASEVERSION/PACKAGENAME/styles/THEMENAME.css>

For example

- <https://cdn.syncfusion.com/ej2/20.4.38/ej2-angular-inputs/dist/ej2-angular-inputs.umd.min.js>
- <https://cdn.syncfusion.com/ej2/20.4.38/ej2-angular-inputs/styles/material.css>

Packages

Syncfusion Angular UI Components packages is published and available in public

[npm](#) registry.

Anatomy of NPM packages

Syncfusion Angular UI Components are shipped as npm packages. Following table explains the purpose of each file available in the package.

| Files | Purpose |
|---|--|
| dist/<packagename>.umd.min.js | dist/<packagename>.umd.js |
| For applications using AMD or Common JS based module loader can be utilize these files. | |
| src/ | This folder contains the script files in ES6 format. It can be utilize Bundling. |
| styles/<themename>.css | styles/<themename>.scss |
| This folder contains the CSS and SCSS files of the package. | |

Drag and Drop for Angular components

Drag and drop is a feature of a user interface that allows users to select an item or items and then move them to a different location or onto another interface element by "dragging" the selected item(s) with a pointing device (such as a mouse) and then "dropping" them at the desired location.

Syncfusion Angular components support drag and drop feature through two libraries. These are [Draggable](#) and [Droppable](#).

Draggable

The Syncfusion's [Draggable](#) library allows users to make any DOM element draggable by passing it as a parameter to the [Draggable](#) constructor. This can be useful for creating interactive and user-friendly interfaces, such as allowing a user to reorder items in a list by dragging them. The below code snippet enables the draggable functionality for the specific DOM element passed to the [Draggable](#) constructor.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Component, ViewChild } from '@angular/core';
import { Draggable } from '@syncfusion/ej2-base';
@Component({
```

```

standalone: true,
  selector: 'app-root',
  template: ` <div id='container'>
    <div #ele class='element'><p class='drag'>Draggable Element </p></div>
  </div> `
})
export class AppComponent {
  @ViewChild('ele',{static: false})element:any;
  ngAfterViewInit() {
    let draggable: Draggable =
      new Draggable(this.element.nativeElement,{ clone: false });
  }
}

```

MAIN.TS

```

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

```

Clone draggable element

Syncfusion provides the option to create a clone of a draggable element while the user is dragging it. It can be achieved by setting the [clone](#) property to `true`. Here's an example of how to create a clone of a draggable element.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Component, ViewChild } from '@angular/core';
import { Draggable } from '@syncfusion/ej2-base';
@Component({
  standalone: true,
  selector: 'app-root',
  template: ` <div id='container'>
    <div #ele class='element'><p class='drag'>Draggable Element </p></div>
  </div> `
})
export class AppComponent {
  @ViewChild('ele',{static: false})element:any;
  ngAfterViewInit() {
    let draggable: Draggable =
      new Draggable(this.element.nativeElement,{ clone: true });
  }
}

```

MAIN.TS

```

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

```

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

Drag area

A drag area is a specific area within a user interface that is designated for drag and drop operations. When an object or element is dragged within a drag area, certain actions or events may be triggered. The user can specify the drag area by using the [dragArea](#) property. Refer to the below sample.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Component, ViewChild } from '@angular/core';
import { Draggable, Droppable, DropEventArgs } from '@syncfusion/ej2-base';
@Component({
  standalone: true,
  selector: 'app-root',
  template: ` <div id='container'>
    <div id='droppable'><p class='drop'>Drop target </p></div>
    <div id='draggable'><p class='drag'>Draggable Element </p></div>
  </div> `
})
export class AppComponent {
  @ViewChild('droppable',{static: false})element: any;
  @ViewChild('draggable',{static: false})element1: any;
  ngAfterViewInit() {
    let draggable: Draggable = new
    Draggable(document.getElementById('draggable') as HTMLElement, { clone:
    false, dragArea: "#droppable" });
  }
}
```

MAIN.TS

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

Droppable

Droppable component refers to an area of a user interface that can receive a draggable component that is being moved by a user. Syncfusion's [Droppable](#) library converts any DOM element into a droppable zone, which accepts draggable elements.

When a draggable component is moved over a droppable component, the [drop](#) event can be triggered. The user can get details about the dropped element through the event argument. Based on the event argument, the user can append the dragged element to the target element.

Refer to the following code snippet to enable droppable zones.

APP.COMPONENT.TS

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



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

import { Component, ViewChild } from '@angular/core';
import { Draggable, Droppable, DropEventArgs } from '@syncfusion/ej2-base';
@Component({
  standalone: true,
  selector: 'app-root',
  template: `<div id='container'>
    <div id='draggable'><p class='drop'>Drop target </p></div>
    <div id='draggable'><p class='drag'>Draggable Element </p></div>
  </div> `
})
export class AppComponent {
  @ViewChild('draggable',{static: false})element: any;
  @ViewChild('draggable',{static: false})element1: any;
  ngAfterViewInit() {
    let draggable: Draggable = new
    Draggable(document.getElementById('draggable') as HTMLElement , {clone:
    false});
    let droppable: Droppable = new
    Droppable(document.getElementById('draggable') as HTMLElement , {
      drop: (e: DropEventArgs) => {
        ((e.droppedElement as HTMLElement).querySelector('.drag') as
        Element).textContent = 'Dropped';
      }
    });
  }
}
```

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 Syncfusion Angular Components

Syncfusion Angular components are rendered with a pre-defined layout or structure that is used to define how the component should be rendered on the user interface. The user wants to customize the appearance of the component and add functionality that is specific to the needs of the application. Syncfusion Angular components have the option to achieve this using template support.

This template support can be achieved using the Angular ng template. It is a way to define a reusable template that can be used across the application. It allows the user to create a template with a specific structure and logic and then use it multiple times throughout the application without repeating the same code.

A template is defined in Angular by using the `<ng-template>` tag within the corresponding component, and it can be given along with the component template name (template) using the # symbol, which can be used to reference the template. The template can include any valid HTML and Angular template syntax, such as directives, bindings, and components.

To access the component data source inside the template, use the `let-data` attribute in the template. Refer to the below code example.

```
`html
<ng-template #template let-data>
<div class="template">{{data.ShipCountry}}</div>
</ng-template>
`
```

For example, the Grid component is shown below with custom template.

```
`js
import { Component, OnInit } from '@angular/core';
@Component({
  selector: 'app-root',
  template: `<div class="control-section">
<ejs-grid #grid [dataSource]='data'>
<e-columns>
<e-column field='OrderID' headerText='OrderID' width='100'></e-column>
<e-column field='CustomerID' headerText='CustomerID' width='100'></e-column>
<e-column field='ShipCountry' headerText='ShipCountry' width='100'>
<ng-template #template let-data>
<div class="template">{{data.ShipCountry}}</div>
</ng-template>
</e-column>
</e-columns>
</ejs-grid>
</div>`
});
export class AppComponent implements OnInit {
  public data: object[];
  ngOnInit(): void {
    this.data = [
      { OrderID: 10248, CustomerID: 'VINET', ShipCountry: 'France' },
      { OrderID: 10249, CustomerID: 'TOMSP', ShipCountry: 'Germany' },
      { OrderID: 10250, CustomerID: 'HANAR', ShipCountry: 'Brazil' }
    ]
  }
}
```

```
];  
}  
};  
,
```

Syncfusion Angular Components - Security

Security is a critical aspect of web applications to protect them from various threats and vulnerabilities. Using HTTPS for data encryption, validating and sanitizing user inputs, and implementing strong authentication measures such as multi-factor authentication are indispensable practices in Web application development.

Syncfusion Angular components are implemented with these security considerations.

Angular includes inherent security functionalities.

- [Security in Angular](#)

Security Vulnerabilities

Security vulnerabilities in web applications refer to weaknesses or flaws in the design, implementation, or configuration of a web application that can be exploited by attackers to compromise the application's integrity, confidentiality, or availability. Here you can see some of the vulnerabilities.

- [Cross-Site Scripting](#) - XSS is a security vulnerability that can occur in web applications. These scripts can steal session cookies, redirect users to malicious websites, or deface the website. XSS vulnerabilities typically arise when the application fails to properly validate or encode user-supplied input before rendering it in the browser.
- [Cross-Site Request Forgery](#) - CSRF is a type of web security vulnerability that allows an attacker to force a logged-in user to perform actions on a web application without their consent or knowledge. CSRF attacks exploit the trust that a web application has in the user's browser by tricking it into sending unauthorized requests to the vulnerable application.
- [Injection Attacks](#) - These occur when an attacker injects malicious code (such as SQL injection, XML injection, or command injection) into input fields or parameters of a web application. If the application does not properly sanitize or validate user inputs, it can execute unintended commands or gain unauthorized access to sensitive data.

Syncfusion Angular components provide support for implementing web applications with enhanced security features.

Security Considerations

Security holds significant importance in software development, and the incorporation of security measures from the outset of the development process is vital for the protection of applications. Syncfusion takes a thorough approach to security in the development of Angular components, encompassing all critical aspects. The following considerations provide a comprehensive overview of security measures.

- [Content Security Policy](#)
- [HTML Sanitizer](#)
- [Browser Storage](#)

Content Security Policy

[Content Security Policy](#) (CSP) is a one of the security feature, that helps the detect the cross-site-scripting(XSS) attacks and malicious code injection. It will throw the errors and warnings while using the inline-styles and inline scripts, eval, new Function, etc in your applications.

To implement Content Security Policy (CSP) in your application, include a `<meta>` tag with specified CSP directives. Syncfusion Angular components have been designed and implemented with adherence to these CSP directives, ensuring enhanced security. These directives are below.

CSP Directives

The following directives are essential for utilizing Syncfusion Angular components.

| Directives | Description | Examples |
|------------------------|---|--|
| | | |
| <code>style-src</code> | Defines the allowed sources for loading stylesheets. This helps mitigate style-based attacks by restricting the locations from which styles can be applied. | <code>style-src 'self' https://cdn.syncfusion.com/ https://fonts.googleapis.com/ 'unsafe-inline';</code> |
| <code>font-src</code> | Defines the allowed sources for loading fonts. It helps prevent font-related security issues by restricting the locations from which fonts can be loaded. | <code>font-src 'self' https://fonts.googleapis.com/ https://fonts.gstatic.com/ data:cdn.syncfusion.com 'unsafe-inline';</code> |
| <code>img-src</code> | Specifies the allowed sources for loading images. It helps control from where images can be displayed on the web page. | <code>img-src 'self' data:"</code> |

Utilizing a web worker within the Spreadsheet Control for exporting necessitates the addition of a specific directive to ensure proper functionality during the export process.

```
worker-src 'self' 'unsafe-inline' * blob;;
```

CSP Sources

The following sources refer to the origins from which resources such as styles, images, fonts are allowed to be loaded and executed on a web page.

| Source | Description | Examples |
|----------------------------|---|---|
| | | |
| <code>self</code> | Refers to the origin from which the protected document is being served, including the same URL scheme and port number | <code>style-src 'self'</code> |
| <code>data</code> | Enables a website to fetch resources using the data scheme, such as loading Base64-encoded images. | <code>img-src 'self' data:</code> |
| <code>unsafe-inline</code> | Allows the use of inline resources, such as inline <code>style</code> elements. | <code>style-src 'self' https://fonts.googleapis.com/ 'unsafe-inline'</code> |

To know more information about the CSP, refer this [documentation](#).

HTML Sanitizer

An HTML sanitizer is a tool or program that helps remove potentially malicious or harmful code from HTML documents. This type of sanitizer is commonly used in web applications to prevent cross-site scripting (XSS) attacks, which can inject malicious code into a website and compromise user data. HTML

sanitizers typically work by analyzing HTML code and removing any potentially dangerous or unwanted elements, such as script tags, inline styles, or event handlers. Other aspects of the HTML may also be modified or cleaned up, such as removing extra whitespace or fixing malformed code.

To avoid the risk of code injection, Syncfusion has provided the [enableHtmlSanitizer](#) API into its UI components. This ensures that HTML strings submitted by users are sanitized, enhancing security measures against potential threats.

When this property is enabled, the HTML string undergoes a thorough sanitization process before being rendered in the component. This approach ensures that user inputs containing potential security threats are meticulously filtered, addressing the risk of XSS and contributing to the overall security robustness of our components in the face of potential attacks.

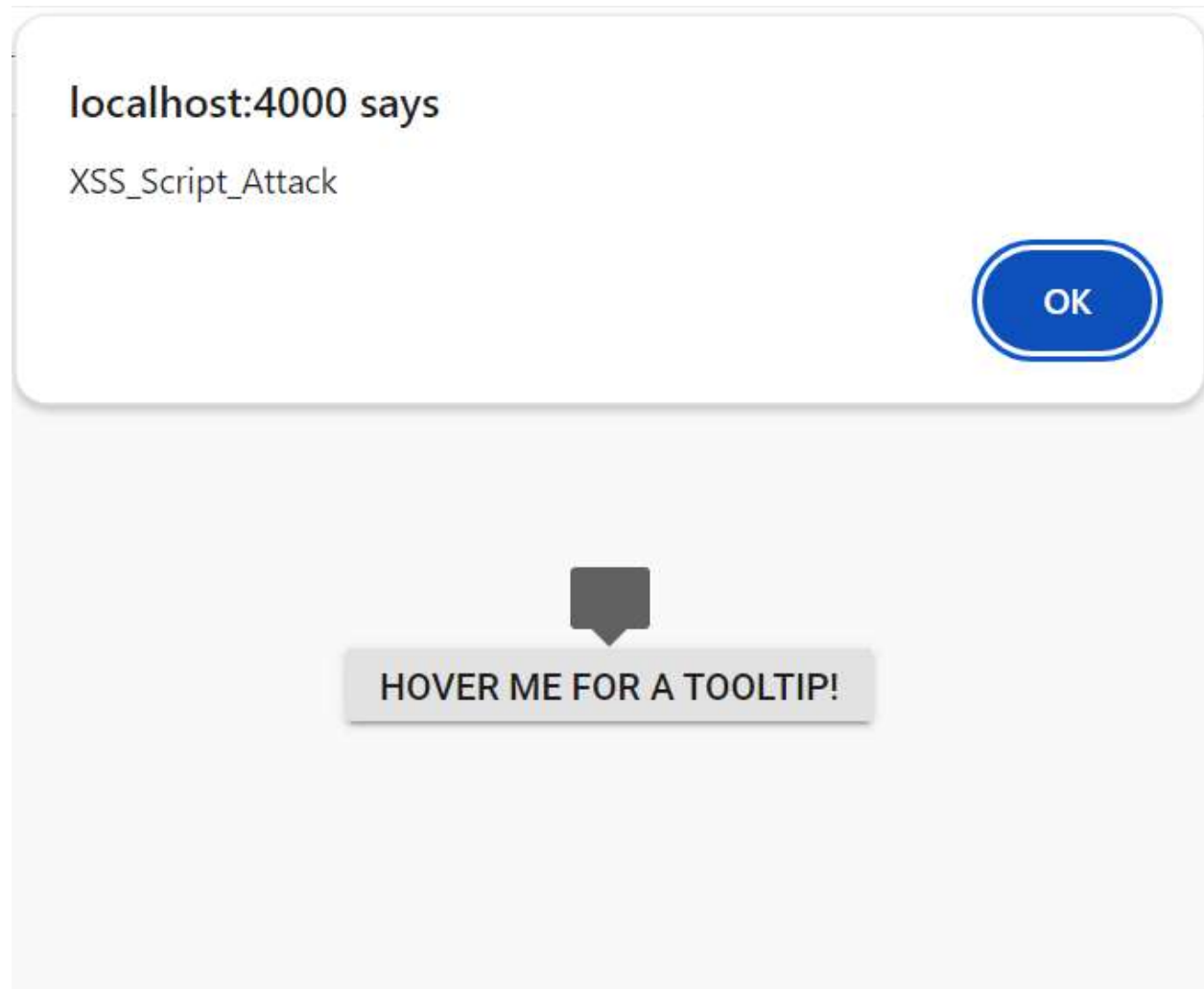
To sanitize input values in a web application using Syncfusion sanitizer, you can use the following code.

```
`ts
import { SanitizeHtmlHelper } from '@syncfusion/ej2-base';
let html: string = '<script>alert("XSS");</script>';
let sanitizedHtml: string = SanitizeHtmlHelper.sanitize(html);
`
```

When `enableHtmlSanitizer` is `true`, the content will be sanitized and displays the code.



When `enableHtmlSanitizer` is `false` or not included this property, the malicious code will be interpreted as script, and the alert pop-up window will be open.



APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { TooltipModule } from '@syncfusion/ej2-angular-popups'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'

import { Component, ViewEncapsulation } from '@angular/core';
import { Draggable } from '@syncfusion/ej2-base';
@Component({
  imports: [
    TooltipModule,
    ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `
    <div id='container' style="display: inline-block; position: relative;
    left: 50%;top: 100px;transform: translateX(-50%);">
      <ej2-tooltip id='tooltip' content='<img src=text
    onerror=alert("XSS Script Attack") \/>' target="#target">
```

```

        <button ej-button id="target">Hover me for a tooltip!</button>
      </ej-tooltip>
    </div>`,
    encapsulation: ViewEncapsulation.None
  })
  export class AppComponent {
  }

```

MAIN.TS

```

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

```

Browser Storage

Browser storage refers to the mechanisms provided by web browsers to store data locally on a user's device. Syncfusion Angular components utilize the following storage options only.

- Local Storage

Local Storage

[Local Storage](#) is a type of web storage mechanism provided by web browsers that allows web applications to store data locally on a user's device. It provides a simple key-value pair storage interface and is accessible via Angular.

Syncfusion Angular components utilize local storage only when persistence is enabled.

Globalization

Globalization

Globalization is the combination of adapting the control to various languages by parsing and formatting the date or numbers (Internationalization (L18N)) and adding cultural-specific customizations and translating the text (Localization (L10N)). The Syncfusion Angular UI components are specific to the **American English (en-US)** culture by default.

The globalization in Syncfusion Angular UI is enabled by

- [Localization](#) - It allows you to localize the text content of the Syncfusion Angular UI Components.
- [Internalization](#) - It provides support for formatting and parsing date and number objects.

Internationalization

The Internationalization library provided by Syncfusion enables formatting and parsing of date and number objects using official [Unicode CLDR](#) JSON data. By default, the **en-US** locale is set as the default culture, and **USD** is set as the default currency code for all Syncfusion Angular UI Components.

Loading CLDR-JSON Data

It is necessary to load the following CLDR data using the `loadCldr` function for cultures other than **en-US**.

| File Name | Path |

```
| ----- | ----- |
| ca-gregorian | cldr/main/en/ca-gregorian.json |
| timeZoneNames | cldr/main/en/timeZoneNames.json |
| numbers | cldr/main/en/numbers.json |
| numberingSystems | cldr/supplemental/numberingSystems.json |
| currencies | cldr/main/en/currencies.json |
```

Note: For `en`, dependency files are already loaded in the library.

Installing CLDR data

CLDR data is available as an npm package. So, you can install it by adding the below command to our package.

```
`bash
npm install cldr-data
```

Binding to i18n library

The `i18n` library to use the CLDR data to format, parse number and date/time values in a way that is appropriate for the `en` culture. The `loadCldr` function takes two arguments, `enNumberData` and `enTimeZoneData`, which are the CLDR data for numbers and time zones, respectively, for the `en` culture.

```
`typescript
import { loadCldr } from '@syncfusion/ej2-base';
import enNumberData from 'cldr-data/main/en/numbers.json';
import enTimeZoneData from 'cldr-data/main/en/timeZoneNames.json';
loadCldr(enNumberData, enTimeZoneData);
```

Changing global Culture and Currency code

To set the default culture and the currency code for all Syncfusion Angular UI Components, you can use the methods `setCulture` for setting the default locale and `setCurrencyCode` for setting the currency code.

```
`typescript
import { setCulture, setCurrencyCode } from '@syncfusion/ej2-base';
setCulture('ar');
setCurrencyCode('QAR');
```

Note: If global culture is not set, then `en-US` is set as the default locale, and `USD` is set as the default currency code.

*Manipulating Numbers**Supported format string*

Based on the [NumberFormatOptions](#) number formatting and parsing operations are processed. You need to specify some or all of the following properties mentioned in the table below

| No | Properties | Description |
|----|---------------------------------------|---|
| 1 | <code>format</code> | Denotes the format to be set .Possible values are
1. N - denotes numeric type
2. C - denotes currency type
3. P - denotes percentage type.
Ex: <code>formatNumber(1234344,{format:'N4'})</code> .
>Note: If no format is specified it takes numeric as default format type. |
| 2 | <code>minimumFractionDigits</code> | Indicates the minimum number of fraction digits . Possible values are 0 to 20. |
| 3 | <code>maximumFractionDigits</code> | Indicates the maximum number of fraction digits. Possible values are 0 to 20. |
| 4 | <code>minimumSignificantDigits</code> | Indicates the minimum number of significant digits. Possible values are 1 to 21.
>Note: If <code>minimumSignificantDigits</code> is given it is mandatory to give <code>maximumSignificantDigits</code> |
| 5 | <code>maximumSignificantDigits</code> | Indicates the maximum number of significant digits. . Possible values are 1 to 21.
>Note: If <code>maximumSignificantDigits</code> is given it is mandatory to give <code>minimumSignificantDigits</code> |
| 6 | <code>useGrouping</code> | Indicates whether to enable the group separator or not . By default grouping value will be true. |
| 7 | <code>minimumIntegerDigits</code> | Indicates the minimum number of the integer digits to be placed in the value. Possible values are 1 to 21. |
| 8 | <code>currency</code> | Indicates the currency code which needs to be considered for the currency formatting. |

Note: The `minimumIntegerDigits`, `minimumFractionDigits` and `maximumFractionDigits` are categorized

as group one, `minimumSignificantDigits` and `maximumSignificantDigits` are categorized as group two.

If group two properties are defined, then group one properties will be ignored.

Custom number formatting and parsing

Custom number formatting and parsing can also be achieved by directly specifying the pattern in the **format** property of `NumberFormatOptions`. One or more of the custom format specifiers listed in the table below can be used to create custom number format.

| Specifier | Description | Input | Format Output |
|-----------|---|---|---------------------|
| 0 | Replaces the zero with the corresponding digit if one is present. Otherwise, zero appears in the result string. | <code>instance.formatNumber(123,{format: '0000' })</code> | <code>'0123'</code> |

| # | Replaces the "#" symbol with the corresponding digit if one is present; otherwise, no digit appears in the result string. | `instance.formatNumber(1234,{format: '####' })` | '1234' |

| . | Denotes the number of digits allowed after the decimal points if it's not specified then no need to specify decimal point values. | `instance.formatNumber(546321,{format: '###0.##0#'})` | '546321.000' |

| % | Percent specifier denotes the percentage type format. | `instance.formatNumber(1,{format: '0000 %'})` | '0100 %' |

| \$ | Denotes the currency type format based on the global currency code specified. | `instance.formatNumber(13,{format: '$ ###.00'})`; | '\$ 13.00' |

| ; | Denotes separate formats for positive, negative and zero values. | `instance.formatNumber(-120,{format: '###.##;(###.00);-0'})`; | '(120.00)' |

| 'String' (single Quotes) | Denotes the characters enclosed within single Quote(') to be replaced in the resultant string. | `instance.formatNumber(-123.44,{format: '####.## '@'})` | '123.44 @' |

Note: If a custom format pattern is specified, other [NumberFormatOptions](#) properties will not be considered.

Number Parsing

``getNumberParser``

The [getNumberParser](#) method, which will return a function that parses a given string based on the [NumberFormatOptions](#) specified.

APP.COMPONENT.TS

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

standalone: true,
  selector: 'app-root',
  template: ` <div id='container'>
    <div>FormattedValue:<span class='format text'>123567.45%</span></div>
    <div>ParsedOutput:<span class='result text'> </span></div>
  </div>`
})
export class AppComponent {
  ngAfterViewInit() {
    let intl: Internationalization = new Internationalization();
    let nParser: Function = intl.getNumberParser({ format: 'P2' ,
useGrouping: false});
    let val: number = nParser('123567.45%');
    (document.querySelector('.result') as Element).innerHTML = val + ' ';
  }
}
```

MAIN.TS

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

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

``parseNumber``

The [parseNumber](#) method, which takes two arguments, the string value and [NumberFormatOptions](#) and returns the numeric value.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Component } from '@angular/core';
import { Internationalization } from '@syncfusion/ej2-base';
@Component({
  standalone: true,
  selector: 'app-root',
  template: ` <div id='container'>
    <div>FormattedValue:<span class='format
text'>$01,234,545.650</span></div>
    <div>ParsedOutput:<span class='result text'> </span></div>
  </div> `
})
export class AppComponent {
  ngAfterViewInit() {
    let intl: Internationalization = new Internationalization();
    let val: number = intl.parseNumber('$01,234,545.650', { format:
'C3', currency: 'USD', minimumIntegerDigits: 8 });
    (document.querySelector('.result') as any).innerHTML = val + '';
  }
}
```

MAIN.TS

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

Number formatting

``getNumberFormat``

The [getNumberFormat](#) method will return a function that formats a given number based on the [NumberFormatOptions](#) specified.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Component } from '@angular/core';
import { Internationalization } from '@syncfusion/ej2-base';
@Component({
  standalone: true,
  selector: 'app-root',
  template: ` <div id='container'>
    <div>Value:<span class='format text'>1234545.65</span></div>
    <div>Formatted Value:<span class='result text'> </span></div>
  </div> `
})
```

```

    </div> `
  })
  export class AppComponent {
    ngAfterViewInit() {
      let intl: Internationalization = new Internationalization();
      let nFormatter: Function = intl.getNumberFormat({ skeleton: 'C3',
currency: 'USD', minimumIntegerDigits: 8 });
      let formattedValue: string = nFormatter(1234545.65);
      (document.querySelector('.result') as Element).innerHTML =
formattedValue as string;
    }
  }
}

```

MAIN.TS

```

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

```

`formatNumber`

The [formatNumber](#) method, which takes two arguments, a numeric value and [NumberFormatOptions](#), and returns the formatted string.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Component } from '@angular/core';
import { Internationalization } from '@syncfusion/ej2-base';
@Component({
  standalone: true,
  selector: 'app-root',
  template: ` <div id='container'>
    <div>Value:<span class='format text'>1234545.65</span></div>
    <div>Formatted Value:<span class='result text'> </span></div>
  </div> `
})
export class AppComponent {
  ngAfterViewInit() {
    let intl: Internationalization = new Internationalization();
    let formattedString: string = intl.formatNumber(12345.65, {
format: 'C5' ,
    useGrouping: false, minimumSignificantDigits: 1,
maximumSignificantDigits: 3 });
    (document.querySelector('.result') as Element).innerHTML =
formattedString;
  }
}

```

MAIN.TS

```

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

```

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

Manipulating DateTime

Supported format string

Date formatting and parsing operations are performed based on the [DateFormatOptions](#). You need to specify some or all of the following properties mentioned in the table below.

Options	Descriptions
---	---
Type	It specifies the type of format to be used supported types . 1. date 2. dateTime 3. time Based on the type specified the supported skeletons are given below 1. short 2. medium, 3. long 4. full Ex: <code>formatDate(new Date(), {type: 'date', skeleton: 'medium'})</code> Note: If not type specified then date type is set by default
skeleton	Specifies the format in which the dateTime format will process

<!-- markdownlint-disable MD036 -->

Date type skeletons

skeleton	Option input	Format Output
---	---	---
short	{type: 'date', skeleton: 'short'}	11/4/16
medium	{type: 'date', skeleton: 'medium'}	Nov 4, 2016
long	{type: 'date', skeleton: 'long'}	November 4, 2016
full	{type: 'date', skeleton: 'full'}	Friday, November 4, 2016

Time type skeletons

skeleton	Option input	Format Output
---	---	---
short	{type: 'time', skeleton: 'short'}	1:03 PM
medium	{type: 'time', skeleton: 'medium'}	1:03:04 PM
Long	{type: 'time', skeleton: 'long'}	1:03:04 PM GMT+5
full	{type: 'time', skeleton: 'full'}	1:03:04 PM GMT+05:30

DateTime type skeletons

Skeleton	Option input	Format Output
---	---	---
short	{type: 'dateTime', skeleton: 'short'}	11/4/16, 1:03 PM
medium	{type: 'dateTime', skeleton: 'medium'}	Nov 4, 2016, 1:03:04 PM
Long	{type: 'dateTime', skeleton: 'long'}	November 4, 2016 at 1:03:04 PM GMT+5
full	{type: 'dateTime', skeleton: 'full'}	Friday, November 4, 2016 at 1:03:04 PM GMT+05:30

Additional skeletons

Apart from the standard date type formats, additional formats are supported by using the additional skeletons given in the below table.

skeleton	Option input	Format Output
---	---	---
d	{skeleton:'d'}	7
E	{skeleton:'E'}	Mon
Ed	{skeleton:'Ed'}	7 Mon
Ehm	{skeleton:'Ehm'}	Mon 12:43 AM
EHm	{skeleton:'EHm;'}};	Mon 12:43
Ehms	{skeleton:'Ehms'}	Mon 2:45:23 PM
EHms	{skeleton:'EHms'}}	Mon 12:45:45
Gy	{skeleton:'Gy'}	2016 AD
GyMMM	{skeleton:'GyMMM'}	: Nov 2016 AD
GyMMMd	{skeleton:'GyMMMd'}	Nov 7, 2016 AD
GyMMMEd	{skeleton:'GyMMMEd'}	Mon, Nov 7, 2016 AD
h	{skeleton:'h'}	12 PM
H	{skeleton:'H'}	12
hm	{skeleton:'hm'}	12:59 PM
Hm	{skeleton:'Hm'}	12:59
hms	{skeleton:'hms'}	12:59:13 PM
Hms	{skeleton:'Hms'}	12:59:13
M	{skeleton:'M'}	11
Md	{skeleton:'Md'}	11/7
MEd	{skeleton:'hms'}	Mon, 11/7
MMM	{skeleton:'MMM'}	Nov
MMMEd	{skeleton:'MMMEd'}	Mon, Nov 7
MMMd	{skeleton:'MMMEd'}	Nov 7
ms	{skeleton:'ms'}	59:13
y	{skeleton:'y'}	2016
yM	{skeleton:'yM'}	11/2016
yMd	{skeleton:'yMd'}	11/7/2016
yMEd	{skeleton:'yMEd'}	Mon, 11/7/2016

```
| yMMM | {skeleton:'yMMM' } | Nov 2016 |
| yMMMd | {skeleton:'yMMMd'} | Nov 7, 2016 |
| yMMMEd | {skeleton:'yMMMEd'} | Mon, Nov 7, 2016 |
| yMMM | {skeleton:'yMMM'} | Nov 2016 |
```

Note: Culture specific format skeletons are also supported.

Custom formats

To use the custom date and time formats, we need to specify the date/time pattern directly in the *format* property.

A custom format string must contain one or more of the following standard date/time symbols.

Symbols	Description
-----	Denotes the era in the date
G	Denotes the year.
y	Denotes month.
M / L	Denotes the day of week.
E / c	Denotes the day of month.
d	Denotes the hour. <i>h</i> for 12 hour and <i>H</i> for 24 hours format.
h / H	Denotes minutes.
m	Denotes seconds.
s	Denotes the am/pm designator it will only be displayed if hour is specified in the <i>h</i> format.
a	Denotes the time zone.
z	To display words in the formatted date you can specify the words with in the single quotes

Custom format example

```
`typescript
import { Component } from '@angular/core';
import { Internationalization } from '@syncfusion/ej2-base';

@Component({
  selector: 'app-root',
  templateUrl: 'default.html'
})
export class AppComponent {
  ngAfterViewInit() {
    let intl: Internationalization = new Internationalization();
```

```
let formattedString: string = intl.formatDate(new Date('1/12/2014 10:20:33'), { format: '\year:\y'
\month:\ MM' });
//Output: "year:2014 month:01"
}
}
`
```

Note: If the format property is given as an option, other properties are not considered.

<!-- markdownlint-enable MD036 -->

Date Parsing

`getDateParser`

The `getDateParser` method will return a function that parses a given string based on the [DateFormatOptions](#) specified.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Component } from '@angular/core';
import { Internationalization } from '@syncfusion/ej2-base';
@Component({
  standalone: true,
  selector: 'app-root',
  template: ` <div id='container'>
    <div>Formatted value:<span class='format text'>Friday, November 4, 2016
at 1:03:04 PM GMT+05:30</span></div>
    <div>parsed Value:<span class='result text'> </span></div>
  </div> `
})
export class AppComponent {
  ngAfterViewInit() {
    let intl: Internationalization = new Internationalization();
    let dParser: Function = intl.getDateParser({skeleton: 'full', type:
'datetime'});
    let val: Date = dParser('Friday, November 4, 2016 at 1:03:04 PM
GMT+05:30');
    (document.querySelector('.result') as Element).innerHTML =
val.toString();
  }
}
```

MAIN.TS

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


``parseDate``

The date object is returned by the [parseDate](#) method, which takes two arguments, a string value and [DateFormatOptions](#).

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Component } from '@angular/core';
import { Internationalization } from '@syncfusion/ej2-base';
@Component({
  standalone: true,
  selector: 'app-root',
  template: ` <div id='container'>
    <div>Formatted value:<span class='format text'>11/2016</span></div>
    <div>parsed Value:<span class='result text'> </span></div>
  </div> `
})
export class AppComponent {
  ngAfterViewInit() {
    let intl:Internationalization = new Internationalization();
    let val: number | any = intl.parseDate('11/2016',{skeleton: 'yM'});
    (document.querySelector('.result') as Element).innerHTML =
    val.toString();
  }
}
```

MAIN.TS

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

Date Formatting

``getDateFormat``

The [getDateFormat](#) method, which will return a function that formats a given date object based on the [DateFormatOptions](#) specified.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Component } from '@angular/core';
import { Internationalization } from '@syncfusion/ej2-base';
@Component({
  standalone: true,
  selector: 'app-root',
  template: ` <div id='container'>
    <div>DateValue:<span class='format text'>new Date('1/12/2014
    10:20:33')</span></div>
    <div>Formatted Value:<span class='result text'> </span></div>
  </div> `
})
export class AppComponent {
```

```

    ngAfterViewInit() {
        let intl: Internationalization = new Internationalization();
        let dFormatter: Function = intl.getDateFormat({ skeleton: 'full',
type: 'dateTime' });
        let formattedString: string = dFormatter(new Date('1/12/2014
10:20:33'));
        (document.querySelector('.result') as Element).innerHTML =
formattedString;
    }
}

```

MAIN.TS

```

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

```

`formatDate`

The [formatDate](#) method, which takes two arguments, the date object and [DateFormatOptions](#), returns the formatted string.

APP.COMPONENT.TS

```

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

import { Component } from '@angular/core';
import { Internationalization } from '@syncfusion/ej2-base';
@Component({
    standalone: true,
    selector: 'app-root',
    template: ` <div id='container'>
        <div>DateValue:<span class='format text'>new Date('1/12/2014
10:20:33')</span></div>
        <div>Formatted Value:<span class='result text'> </span></div>
        <div> `
})
export class AppComponent {
    ngAfterViewInit() {
        let intl: Internationalization = new Internationalization();
        let date: Date = new Date();
        let formattedString: string = intl.formatDate(new Date('1/12/2014
10:20:33'), { skeleton: 'GyMMM' });
        (document.querySelector('.result') as Element).innerHTML =
formattedString;
    }
}

```

MAIN.TS

```

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

```

Getting started with Localization

Localization library allows you to localize the text content of the Syncfusion Angular UI Components. This is useful if you want to display the UI in a language other than English.

Loading translations

To load a translation object in your application, you can use the load function from the @syncfusion/ej2-base module. This function takes an object that contains the translations for various languages, with the keys being the language codes and the values being the translation objects.

For example, to load translations for English and French, you can do the following:

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, GroupService, PageService } from '@syncfusion/ej2-angular-grids'
import { L10n, setCulture } from '@syncfusion/ej2-base';
import { Component, OnInit } from '@angular/core';
import { data } from './datasource';
import { PageSettingsModel } from '@syncfusion/ej2-angular-grids';
setCulture('de-DE');
L10n.load({
  'de-DE': {
    'grid': {
      'EmptyRecord': 'Keine Aufzeichnungen angezeigt',
      'GroupDropArea': 'Ziehen Sie einen Spaltenkopf hier, um die
Gruppe ihre Spalte',
      'UnGroup': 'Klicken Sie hier, um die Gruppierung aufheben',
      'EmptyDataSourceError': 'DataSource darf bei der Erstaustellung
nicht leer sein, da Spalten aus der dataSource im AutoGenerate
Spaltenraster',
      'Item': 'Artikel',
      'Items': 'Artikel'
    },
    'pager': {
      'currentPageInfo': '{0} von {1} Seiten',
      'totalItemsInfo': '({0} Beiträge)',
      'firstPageTooltip': 'Zur ersten Seite',
      'lastPageTooltip': 'Zur letzten Seite',
      'nextPageTooltip': 'Zur nächsten Seite',
      'previousPageTooltip': 'Zurück zur letzten Seit',
      'nextPagerTooltip': 'Zum nächsten Pager',
      'previousPagerTooltip': 'Zum vorherigen Pager'
    }
  }
});
@Component({
  imports: [
    GridModule
  ],
  providers: [GroupService, PageService]
  standalone: true,
  selector: 'app-root',
```

```

        template: `<ejs-grid [dataSource]='data' [locale]='"de-DE"'
[allowGrouping]='true' [allowPaging]='true' [pageSettings]='pageOptions'
height='220px'>
            <e-columns>
                <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=120></e-column>
                <e-column field='CustomerID' headerText='Customer ID'
width=150></e-column>
                <e-column field='ShipCity' headerText='Ship City'
width=150></e-column>
                <e-column field='ShipName' headerText='Ship Name'
width=150></e-column>
            </e-columns>
        </ejs-grid>`
    })
    export class AppComponent implements OnInit {
        public data?: Object[];
        public pageOptions?: PageSettingsModel;
        ngOnInit(): void {
            this.data = data;
            this.pageOptions = { pageSize: 6 };
        }
    }
}

```

MAIN.TS

```

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

```

Changing current locale

The current locale can be changed for all the Syncfusion Angular UI Components in your application by invoking `setCulture` function with your desired culture name.

```

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

L10n.load({
    'fr-BE': {
        'Button': {
            'id': 'Numéro de commande',
            'date': 'Date de commande'
        }
    }
});

setCulture('fr-BE');
`

```

Note: Before changing a culture globally, you need to ensure that locale text for the concerned culture is loaded through `L10n.load` function.

Angular Reactive Form Validator

Syncfusion's Angular UI Component provides custom built-in Reactive Form validators. These validators will help you validate your input component on the client side before submitting.

Here the list of validators available.

- Min
- Max
- CreditCard
- Date
- Datelso
- digits
- Email
- MaxLength
- MinLength
- Number
- Required
- Tel
- Url
- Range
- Rangelength

Validator	Description	Code Snippet
min	This validator check whether the form control value is less than min value and it makes the form validation failed if it not satisfied.	Syntax: <code>FormValidators.min (18)</code> <code>this.reactForm = new FormGroup({age: new FormControl("", [FormValidators.min(8)]) });</code>
max	This validator check whether the form control value is not greater than max value and it makes the form validation failed if it not satisfied.	Syntax: <code>FormValidators.max(18)</code> <code>this.reactForm = new FormGroup({age: new FormControl("", [FormValidators.max(8)]});</code>
date	The date validator is validate the given control value is date format string.	Syntax: <code>FormValidators.date</code> <code>this.reactForm = new FormGroup({ dob: new FormControl("", [FormValidators.date]) });</code>
dateISO	The datelso validator is validate the given input is ISO standard format. Example : 2008-09-15T15:53:00	Syntax : <code>FormValidator.dateISO</code> <code>this.reactForm = new FormGroup({ dob: new FormControl("", [FormValidators.dateISO]) });</code>
digits	The digit validator validates the given input is numeric that allows dot digits.	Syntax : <code>FormValidator.digits</code> <code>this.reactForm = new FormGroup({ currency: new FormControl("", [FormValidators.digits]) });</code>

| email | The email validator validates the given input is email format string. Syntax :
FormValidator.email | this.reactForm = new FormGroup({ email: new FormControl(",
 [FormValidators.email]) }); |

| maxLength | This validator check whether the form control value length is not greater than max value and it makes the form validation failed if it not satisfied. Syntax:
FormValidators.maxLength(150) | this.reactForm = new FormGroup({ comments: new
 FormControl(", [FormValidators.maxLength(150)]) }); |

| minLength | This validator check whether the form control value length is not less than min value and it makes the form validation failed if it not satisfied. Syntax: **FormValidators.minLength(100)** |
 this.reactForm = new FormGroup({ comments: new FormControl(", [FormValidators.
 minLength(100)])); |

| number | This validator validate the form control value is a number type or not. Syntax:
FormValidator.number | this.reactForm = new FormGroup({ age: new FormControl(",
 [FormValidators.number])}); |

| required | This validator validates the form controls value is should not be empty or null or undefined. Syntax: **FormValidator.required** | this.reactForm = new FormGroup({ name: new
 FormControl(", [FormValidators.number])}); |

| tel | This validator validates the form controls value is should be telephone number. Syntax:
FormValidator.tel | this.reactForm = new FormGroup({ Phone: new FormControl(", [FormValidators.
 tel])}); |

| range | This validator validates the form controls value is should be within the specific range value. Syntax: **FormValidator.range(18,50)** | this.reactForm = new FormGroup({ age: new FormControl(",
 [FormValidators.range(18,50)]) }); |

| rangeLength | This validator validates the form controls value is should be within the specific range length. Syntax: **FormValidator.rangeLength(100,150)** | this.reactForm = new FormGroup({age: new
 FormControl(", [FormValidators.rangeLength(100,150)]) }); ||

Creating Angular Reactive Form with Syncfusion Angular UI Validator

Step 1: Get started with an Angular Reactive Form by using the following link [Angular Reactive form](#).

Step 2: To add Syncfusion Angular UI Validation, you'll need to import the below validator class from @syncfusion/ej2-angular-inputs.

```
,
import { FormValidators } from '@syncfusion/ej2-angular-inputs';
,
```

Step 3: Add the required validator to your form group component.

```
,
this.reactForm = new FormGroup({
'check': new FormControl(", [FormValidators.required]),
'email_check': new FormControl(", [FormValidators.email]),
'date_check': new FormControl(", [FormValidators.date]),
```

```
'city': new FormControl('', [FormValidators.required]),
'state': new FormControl('', [FormValidators.required]),
});
`
```

Simple Sample with Syncfusion Angular UI Validation.

[create this as Documentation Sample](#)

Right-To-Left support in Syncfusion Angular Components

Right-to-Left (RTL) support allows applications to effectively communicate with users who use languages that are written from right to left, such as Arabic, Hebrew, etc.

Syncfusion Angular UI components support for right-to-left (RTL) by setting the `enableRtl` property to `true`. This adds the class name `e-rtl` to the component element and renders all Syncfusion Angular components in a right-to-left direction.

Enable RTL for all components

To enable Right-To-Left (RTL) support for all components, users can set the `enableRtl` property directly in their application. Here is an example code snippet using the `ListView` component:

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ListViewModule } from '@syncfusion/ej2-angular-lists'
import { enableRtl } from '@syncfusion/ej2-base';
import { Component } from '@angular/core';
// Enables Right to left alignment for all controls
enableRtl(true);
@Component({
  imports: [

    ListViewModule
  ],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-listview id='sample-list' [dataSource]='data'
[fields]='fields' showHeader='true' headerTitle='Social Media'>
  <ng-template #template let-data="">
    <span class='{{data.class}} icon'><span
class='media'>{{data.socialMedia}}</span></span>
  </ng-template></ejs-listview>`
})
export class AppComponent {
  public data: Object = [
    { class: 'facebook', socialMedia: 'Facebook', id: 'media1' },
    { class: 'twitter', socialMedia: 'Twitter', id: 'media2' },
    { class: 'tumblr', socialMedia: 'Tumblr', id: 'media4' },
    { class: 'google-plus', socialMedia: 'Google Plus', id: 'media5' },
    { class: 'skype', socialMedia: 'Skype', id: 'media6' },
    { class: 'vimeo', socialMedia: 'Vimeo', id: 'media7' },
    { class: 'instagram', socialMedia: 'Instagram', id: 'media8' },
    { class: 'youtube', socialMedia: 'YouTube', id: 'media9' },
  ];
};
```

```
public fields: Object = { text: 'socialMedia' };
}
```

MAIN.TS

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

Enable RTL for an individual component

To enable Right-To-Left (RTL) support for an individual component, users can set the `enableRtl` property in the component's options. Here is an example code snippet using the `ListView` component:

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ListViewModule } from '@syncfusion/ej2-angular-lists'
import { Component } from '@angular/core';
@Component({
  imports: [
    ListViewModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-listview id='sample-list' [dataSource]='data'
[fields]='fields' showHeader='true'
enableRtl='true' headerTitle='Social Media'>
<ng-template #template let-data="">
<span class='{{data.class}} icon'><span
class='media'>{{data.socialMedia}}</span></span>
</ng-template></ejs-listview> `
})
export class AppComponent {
  public data: Object = [
    { class: 'facebook', socialMedia: 'Facebook', id: 'media1' },
    { class: 'twitter', socialMedia: 'Twitter', id: 'media2' },
    { class: 'tumblr', socialMedia: 'Tumblr', id: 'media4' },
    { class: 'google-plus', socialMedia: 'Google Plus', id: 'media5' },
    { class: 'skype', socialMedia: 'Skype', id: 'media6' },
    { class: 'vimeo', socialMedia: 'Vimeo', id: 'media7' },
    { class: 'instagram', socialMedia: 'Instagram', id: 'media8' },
    { class: 'youtube', socialMedia: 'YouTube', id: 'media9' },
  ];
  public fields: Object = { text: 'socialMedia' };
}
```

MAIN.TS

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


Schematics

The Syncfusion Angular components now supports the [schematics](#).

The [NPM](#) packages are installed by using the Angular CLI [add](#) command.

Creating Angular application

To create an Angular application, instal the [Angular CLI](#) globally using the following command.

,

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

,

Then, create a new Angular application using the following command.

,

```
ng new my-app
```

,

This will download all the required files and initialize the NPM install.

Syncfusion package initialization

All the Syncfusion Angular packages can be installed using the following command in the CLI application.

,

```
ng add 'package-name'
```

,

For example,

,

```
ng add @syncfusion/ej2-angular-grids
```

,

The above command does the following configuration to your Angular app,

- Adds `@syncfusion/ej2-angular-grids` package and its peer dependencies to your `package.json` file.
- Imports the `GridModule` in your application module `app.module.ts`.
- Registers the Syncfusion UI default theme (material) in the `angular.json` file.

Adding specific modules from multiple component package

The EJ2 Angular Schematics support adding a specific module from the component package to the `app.module.ts` file.

While initializing the package, we can pass the preferred module as a parameter using the following command.

,

```
ng add @syncfusion/<ej2-angular-package-name> --module=module1,module2,module_3
```

NOTE: While passing module names, there should be no space between them. If any, it will be ignored.

For example,

```
ng add @syncfusion/ej2-angular-popups --module=tooltip
```

```
ng add @syncfusion/ej2-angular-inputs --module=slider,colorpicker,maskedtextbox
```

```
ng add @syncfusion/ej2-angular-navigation --module=treeview,tab,contextmenu
```

Invalid and misspelled module names

When you pass valid and invalid module names, the schematics will add all the valid modules and throw an error for the invalid modules.

For example,

```
ng add @syncfusion/ej2-angular-popups --module=tooltip,treeview
```

Here, the `tooltip` is a valid module, but the `treeview` is invalid since it does not belong to `@syncfusion/ej2-angular-popups` package. Schematics add only the `tooltip` but it will throw the following error message for `treeview`. It is also applicable for a misspelt module name.

The `treeview` module is not a part of the package, `@syncfusion/ej2-angular-popups`. The available modules are `Tooltip`, `Dialog`.

State Persistence in Syncfusion Angular components

Syncfusion Angular UI components support persisting their state across page refreshes or navigation. To enable this feature, set the `enablePersistence` property to `true` for the desired component. This stores the component's state in the browser's `localStorage` object on the `unload` event of the page. For example, the `enablePersistence` property can be set for the `Grid` component, as shown in the following code snippet.

APP.COMPONENT.TS

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

```

import { PageService, SortService, FilterService, GroupService } from
 '@syncfusion/ej2-angular-grids'
import { Component, OnInit } from '@angular/core';
import { data } from './datasource';
import { PageSettingsModel } from '@syncfusion/ej2-angular-grids';
@Component({
  imports: [

    GridModule
  ],
  providers: [PageService,
    SortService,
    FilterService,
    GroupService]
  standalone: true,
  selector: 'app-root',
  template: `<ejs-grid [dataSource]='data' [enablePersistence]='true'
[allowPaging]="true" [pageSettings]='pageSettings'>
    <e-columns>
      <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=90></e-column>
      <e-column field='CustomerID' headerText='Customer ID'
width=120></e-column>
      <e-column field='Freight' headerText='Freight'
textAlign='Right' format='C2' width=90></e-column>
      <e-column field='OrderDate' headerText='Order Date'
textAlign='Right' format='yMd' width=120></e-column>
    </e-columns>
  </ejs-grid>`
})
export class AppComponent implements OnInit {
  public data?: object[];
  public pageSettings?: PageSettingsModel;
  ngOnInit(): void {
    // this.data = data;
    this.pageSettings = { pageSize: 6 };
  }
}

```

MAIN.TS

```

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

```

State Persistence supported components and properties

The following table illustrates the list of Syncfusion Angular components that are supported with state persistence and the properties that are stored in the `localStorage`.

<!-- markdownlint-disable MD033 -->

component Name	Properties
----------------	------------

Grid	<ul style="list-style-type: none">• Columns• filterSettings• searchSettings• groupSettings• pageSettings• selectedRowIndex• scrollPosition
Accordion	<ul style="list-style-type: none">• expandedIndices
Tabs	<ul style="list-style-type: none">• selectedItem
Schedule	<ul style="list-style-type: none">• currentView• selectedDate• scrollLeft• scrollTop
Kanban	<ul style="list-style-type: none">• columns• dataSource• swimlaneToggleArray
Chart	<ul style="list-style-type: none">• zoomFactor• zoomPosition
Maps	<ul style="list-style-type: none">• zoomSettings
Pivot Table	<ul style="list-style-type: none">• dataSourceSettings• pivotValues• gridSettings• chartSettings• displayOption
TreeGrid	<ul style="list-style-type: none">• columns• pageSettings• searchSettings• filterSettings• selectedRowIndex• sortSettings
Switch, Checkbox	<ul style="list-style-type: none">• checked
RadioButton	<ul style="list-style-type: none">• checked• value

ColorPicker, ListBox, In-placeEditor, RichTextEditor, Autocomplete, Calendar, ComboBox, DatePicker, DropDownList, MaskedTextBox, NumericTextBox, Textbox, TimePicker, Multiselect, DateTimePicker, Slider, Dropdown Tree	<ul style="list-style-type: none"> • value
QueryBuilder	<ul style="list-style-type: none"> • rule
Splitter	<ul style="list-style-type: none"> • paneSettings
DateRangePicker	<ul style="list-style-type: none"> • startDate • endDate • value
Uploader	<ul style="list-style-type: none"> • filesData
ListView	<ul style="list-style-type: none"> • cssClass • enableRtl • htmlAttributes • enable • fields • animation • headerTitle • sortOrder • showIcon • height • width • showCheckBox • checkBoxPosition
TreeView	<ul style="list-style-type: none"> • selectedNodes • checkedNodes • expandedNodes
Dashboard Layout	<ul style="list-style-type: none"> • panels
File Manager	<ul style="list-style-type: none"> • view • path • selectedItems
Sidebar	<ul style="list-style-type: none"> • type • position • isOpen

<!-- markdownlint-enable MD033 -->

Check out the following component's document to learn more about the state persistence.

- [Grid](#)
- [TreeGrid](#)
- [Pivot Table](#)
- [Gantt](#)
- [Kanban](#)
- [Schedule](#)
- [QueryBuilder](#)
- [Tabs](#)

How To

Update Syncfusion npm package

The latest Syncfusion npm package can be updated with the help of [npm-check-updates](#) package.

1. You can install the `npm-check-update` package globally to use this as CLI.

```
`bash
npm install -g npm-check-updates
ncu -u -f /^@syncfusion/
`
```

This will update the package.json file to latest version for all `@syncfusion` packages.

2. Now, run the following commands to update the packages in `node_modules` and remove the duplicate package which is already installed.

```
`bash
npm update
npm dedupe
`
```

Updating a specific npm package

Run the following commands from the command prompt in the application root to update a specific npm package in `node_modules` and remove the installed duplicate package.

```
`bash
npm update @syncfusion/ej2-grids
npm update @syncfusion/ej2-angular-grids
npm dedupe
`
```

How to use SCSS in Angular CLI

Sass (SCSS) is a powerful CSS pre-processor that allows you to use variables, nested rules, and functions to make writing CSS easier and more efficient. In this guide, we will show you how to use SCSS in Angular CLI to customize the styles of your Angular Syncfusion components.

SASS variables location

The SASS variables for Essential JS 2 components can be found in the following location,

```
node_modules/@syncfusion/package-name/styles/themename.scss
```

For example, the location for the navigation component's SASS variables is,

```
node_modules/@syncfusion/ej2-angular-grids/styles/material.scss
```

Initialization of SCSS variables

To use the SCSS variables in your project, you need to import the styles of the required component in the `src/styles.scss` file.

```
`typescript
@import "ej2-grids/styles/material.scss"
`
```

Configuring Node-Sass in Angular CLI json

To avoid SCSS compilation issues and to map the SCSS file path, you need to add the `stylePreprocessorOptions` to the `.angular-cli.json` file. You should add this option in two places under the `apps` object,

1. `angular-cli.json -> {}build -> {}options`
2. `angular-cli.json -> {}test -> {}options`

This allows you to use the following paths globally in your Angular app.

```
`typescript
"stylePreprocessorOptions": {
  "includePaths": [
    "node_modules/@syncfusion"
  ]
},
`
```

An angular sample with SCSS compilation to render the Essential JS 2 Grid component can be downloaded from the following [GitHub link](#).

Overriding styles

You can override the control styles in Syncfusion Angular components by replacing the sass variable values like this,

```
`
// SASS Variable override
`
```

```
$accent: black;
$primary: rgb(0, 255, 157);
// syncfusion styles
@import '../node_modules/@syncfusion/ej2-angular-grids/styles/material.scss';
`
```

Angular-CLI Version 8 With SASS

In version 8, the Angular Team moved away from `node-sass` in favor of `sass`. However, you can still use `node-sass` manually by running the following command:

```
`bash
npm install node-sass --save-dev
`
```

By following the steps outlined in this guide, you should now be able to use SCSS in Angular CLI to customize the styles of your Essential JS 2 components.

How to use Custom CSS File in Angular Application

This guide explains the process of using custom styles generated by the [Theme Editor](#) in an Angular application that uses Essential JS2 Angular components.

Create an Angular application

Follow the [documentation](#) to create an Angular application that includes Essential JS2 Angular components.

Generating custom CSS file

Use the [theme-studio](#) to generate a custom CSS file.

Adding custom CSS in Angular application

Place the generated custom CSS file inside the `./src/` directory of your Angular application.

Custom style mapping

In the `./angular.json` file, add the path to the custom CSS file under the `architect/build/options/styles` property, as shown below,

```
`typescript
"styles": [
  "src/styles.css",
  "src/custom-material.css"
],
`
```

Run Angular application

Start the Angular application and the Essential JS2 Angular components will now display with the custom styles.

How to resolve Content Security Policy (CSP) errors

Enabling the strict Content Security Policy (CSP) may cause the following issue with the Syncfusion Angular components in your application.

Image loading

Syncfusion license banner utilize the image from **base64**, which is not allowed on strict CSP-enabled sites. To overcome this restriction, it is necessary to add the [img-src data:](#) directive in the meta tag or consider [registering the license key](#).

Troubleshooting

Compatibility Issues with Syncfusion Angular Packages and Latest Angular CLI

This article offers solutions to problems that you can run into while utilising the Syncfusion Angular UI components.

Are Syncfusion Angular packages compatible with the latest Angular CLI?

Yes, Syncfusion Angular IVY packages are fully compatible with the latest Angular CLI.

If Syncfusion Angular packages are compatible with Latest Angular CLI, what version should we prepare?

It is recommended to use version 21.1.39 or the latest available version of the Syncfusion Angular packages. For more information check that the [version compatibility](#).

How can I check the compatibility of Syncfusion Angular packages with a specific version of Angular CLI?

Syncfusion provides a compatibility [documentation](#) that outlines the supported versions of Angular CLI.

What compatibility issues were encountered while using Syncfusion Angular NGCC packages with the latest version of Angular CLI?

Some of the new features included in Angular CLI 16 were incompatible with the NGCC component package, such as the modified Ivy rendering engine. An error notice appeared as a result, claiming that Angular Ivy is incompatible with the library (@syncfusion/ej2-angular-grids) that declares GridModule.

Where can I find more information about this issue?

More information about the compatibility issue can be found on the official Angular documentation [page](#).

What is the cause of the error message encountered during testing?

The error message encountered during the compile time and suggests that the library (@syncfusion/ej2-angular-grids) which declares GridModule is not compatible with Angular Ivy. This is due to the incompatibility between the NGCC component package and some of the new features introduced in Angular CLI 16.

This likely means that the library (@syncfusion/ej2-angular-grids) which declares GridModule has not been processed correctly by ngcc, or is not compatible with Angular Ivy.

How can I resolve the compatibility issue encountered when using Syncfusion Angular NGCC packages with the latest version of Angular CLI?

One way to resolve the issue is to use IVY compiled [packages](#) instead of NGCC compiled packages.

Are there any other ways to resolve the compatibility issue?

Another way to resolve the issue is to downgrade the version of Angular CLI to a version that is compatible with the NGCC component package. However, this may not be a recommended solution as it may involve downgrading other packages in your project and can lead to compatibility issues with other dependencies.

Content Security Policy

Content Security Policy (CSP) is a security feature implemented by web browsers that helps to protect against attacks such as cross-site scripting (XSS) and data injection. It limits the sources from which content can be loaded on a web page.

To enable strict [Content Security Policy \(CSP\)](#), certain browser features are disabled by default. In order to use Syncfusion Angular components with strict CSP mode, it is essential to include following directives in the CSP meta tag.

- Syncfusion components are rendered with calculated **inline styles** and **base64** font icons, which are blocked on a strict CSP-enabled site. To allow them, add the [style-src 'self' 'unsafe-inline'](#); and [font-src 'self' data:;](#) directives in the meta tag as follows.

HTML

```
<meta http-equiv="Content-Security-Policy" content="default-src 'self';  
style-src 'self' 'unsafe-inline';  
font-src 'self' data:;" />
```

- Syncfusion **material** and **tailwind** built-in themes contain a reference to the [Roboto's external font](#), which is also blocked. To allow them, add the [external font](#) reference to the [style-src](#) and [font-src](#) directives in the above meta tag.

The resultant meta tag is included within the `<head>` tag and resolves the CSP violation on the application's side when utilizing Syncfusion Angular components with material and tailwind themes.

HTML

```
<head>  
...  
<meta http-equiv="Content-Security-Policy" content="default-src 'self';  
style-src 'self' https://fonts.googleapis.com/ 'unsafe-inline';  
font-src 'self' https://fonts.googleapis.com/ https://fonts.gstatic.com/  
data:;" />  
</head>
```

Note: From the 2023 Vol2 - 22.1 release onwards, the Content Security Policy for Syncfusion Angular components has been enhanced. The usage of the `unsafe-eval` directive has been eliminated from the CSP meta tag.

[View the Angular sample enabled with strict CSP in Github](#)

See also

- [How to resolve the Content Security Policy \(CSP\) errors](#)

Frameworks and Feature

Tree Shaking

Tree shaking is an optimization technique used in modern JavaScript and TypeScript development using bundling tools like Webpack or Rollup. Its primary purpose is to eliminate dead or unused code from the application's bundle, resulting in smaller, more efficient, and faster-loading web applications.

This section explains how tree shaking works in Angular and how to implement it with Syncfusion Angular components in the application.

Tree Shaking in Angular

The Angular CLI uses Webpack by default for bundling script files, which has supported Tree Shaking since version 2. Generally, it works based on the static structure of ES2015 module syntax, which includes `import` and `export` statements. If your code does not use the exported methods or functions from your application or libraries, it becomes unused code in the browser. So, tree shaking removes all such unused code from the output bundle. This leads to improved performance with faster load times due to minimal JavaScript code, resulting in a better user experience.

Note: Ahead Of Time (AOT) compilation process performs tree shaking in the Angular application, which was enabled by default starting from Angular version 9. To know more about AOT compilation, refer to this [documentation](#).

Using Syncfusion components with Tree Shaking

Syncfusion Angular components support Tree Shaking by default and do not require any additional changes at the application level.

Implementing Tree Shaking in an Angular Application

The following steps demonstrate how to create an Angular application with Syncfusion components and bundle it with tree shaking.

1.Create an Angular application with Syncfusion Angular DataGrid component as described in the [Getting Started](#) using the Angular Standalone. 2.Use the following code snippet that shows how to enable Tree Shaking in the `angular.json` configuration file.

ANGULAR.JSON

```
"configurations": {  
  "production": {  
    "optimization": true,  
    "outputHashing": "all",  
    "sourceMap": false,  
    "namedChunks": false,  
    "aot": true,  
    "extractLicenses": true  
  }  
},
```

3.Run the `ng build --configuration=production` or `ng serve --configuration=production` command to build or serve the application with Tree Shaking enabled. 4.After build the application, users can utilize the [bundle analyzer](#) tool to ensure that tree shaking works properly in their application.

[View the Angular Tree Shaking sample on GitHub](#)

By following these steps and implementing tree shaking in the Angular application, users can optimize the performance and enhance the overall user experience.

Bundle size for Syncfusion Angular Grid component

The following table demonstrates the size of the Grid module and includes the addition of some features to it in the Angular application.

Module	Raw Size	Transfer Size
Empty App	199.45 kB	56.22 kB
GridModule	1.13 MB	249.94 kB
PageService	1.31 MB	274.79 kB
SortService	1.32 MB	276.41 kB
FilterService	1.68 MB	333.16 kB
GroupService	1.73 MB	340.22 kB

Note: Make sure you are using `GridModule` for the DataGrid component instead of `GridAllModule`. Using `GridAllModule` will increase the bundle size by including all features of DataGrid. To know about the feature modules of DataGrid, refer to this [documentation](#).

See also

- [Ahead-Of-Time \(AOT\) compilation](#)

Ahead-of-Time (AOT) Compilation in Angular

Ahead-of-Time (AOT) Compilation is a technique that compiles Angular's templates and components into browser-ready JavaScript code during the build process itself. This can improve the initial loading time of the application. From angular v9, the AOT compilation is enabled by default. To know more about AOT compilation, refer to this [documentation](#).

This section explains the benefits of using AOT compilation and how to implement it in applications.

Why use AOT compilation

The following are the benefits of using AOT compilation in Angular applications.

- AOT compilation translates Angular templates into optimized JavaScript code during the build process, which leads to faster initial loading times for your application.
- By pre-compiling templates, AOT eliminates the need for the Angular compiler to be part of the final bundle, resulting in smaller bundle sizes and reduced runtime overhead.
- AOT compilation catches template errors at build time rather than runtime, making it easier to identify and fix issues early in the development process.

Using Syncfusion components with AOT

All [Syncfusion Angular components](#) are compatible with AOT compilation.

Implementation of AOT Compilation in an Angular Application

1. From the angular v9, the Ivy compiler enables the AOT compilation by default in the `angular.json` configuration file. If you're using Angular versions prior to 9, ensure that `aot` is set to `true` in the `angular.json` file.

ANGULAR.JSON

```
"build": {  
  ...  
  "options": {  
    ...  
    "aot": true,  
  }  
}
```

To know more about AOT compilation with Ivy, refer to this [documentation](#).

2. AOT compilation improves the initial rendering performance of the application. To achieve this performance improvement, build the application using the following command:

CMD

```
ng build --configuration=production
```

See also

- [Angular Ivy compiler](#)
- [Angular Tree Shaking](#)

Angular Universal: Server-side Rendering in Angular Frameworks

Angular is a widely-used client-side web development framework. However, by default, it runs only on the client-side. Many web development tools are designed for server-side frameworks like Asp.Net WebForms and Asp.Net MVC. To bridge this gap, Angular provides a feature called Angular Universal that allows for server-side rendering (SSR) of Angular applications.

This documentation provides information about Angular Universal and how to implement it with Syncfusion Angular components in an Angular Universal application.

What is Angular Universal

[Angular Universal](#) is a technology for rendering Angular components on the server and sending the pre-rendered HTML to the client, where JavaScript takes over and adds interactivity. This has several benefits, such as improved SEO, faster load times, and better accessibility.

Why use Server-side Rendering

Server-side rendering basically allows an application to render faster in a browser and enables users to view the application's UI before it becomes fully interactive. There are three main reasons to create an Angular Universal application.

- Search engines can easily index content in SSR applications because the initial HTML is fully rendered on the server, making it more accessible to web crawlers.

- Users see the content sooner because the server sends the pre-rendered HTML, reducing the time it takes for the page to become interactive.
- SSR can be especially beneficial for users on slower devices or those with limited processing power, as more work is performed on the server.

Create an Angular Universal application

Syncfusion Angular UI Components support the server-side rendering of an Angular Universal application. The following steps demonstrate how to create an Angular application with Syncfusion components and enable server-side rendering in the Angular application.

1.Create an Angular application with Syncfusion Angular DataGrid component as described in the [Getting Started](#) using the Angular CLI. 2.To add the Server-Side Rendering (SSR) into the application, run the following command.

CMD

```
ng add @nguniversal/express-engine
```

3.After installing the above command, enable Client [Hydration](#). Hydration is the process that restores the server-side rendered application on the client. To enable hydration, import the [provideClientHydration](#) function and add it to the `providers` section of the `app.module.ts` file as shown below.

APP.MODULE.TS

```
import {provideClientHydration} from '@angular/platform-browser';
// ...
@NgModule({
  // ...
  providers: [ provideClientHydration() ],
  bootstrap: [ AppComponent ]
})
export class AppModule {
  // ...
}
```

4.Run the following command to build and serve the application in both prerender and render(ssr) modes of the Angular universal.

CMD

```
npm run dev:ssr
```

[View the Angular Universal sample on GitHub](#)

See also

- [Angular Universal](#)
- [Getting Started ASP.NET Core with Angular](#)
- [Getting Started ASP.NET MVC with Angular](#)

Angular Lazy Loading

This section explains how to implement Lazy Loading with Essential JS2 Angular components in Angular, which improves the initial loading time of the application by loading only the necessary modules on demand.

Lazy Loading

Lazy loading is a technique that loads additional payload only when needed, which can improve the overall performance and user experience of your Angular application. By using code splitting, you can lazy load the Syncfusion components and routes in Angular. This can reduce the initial loading time of the application.

Creating a Syncfusion component in Angular

To create a Syncfusion component in Angular, refer to the [getting started](#) documentation. Additionally, you can refer to the Angular [lazy-loading](#) documentation for more information on how to implement lazy loading in your application.

Here's an example of an Angular application that has routing enabled and uses lazy loading to load the **CustomersComponent** and **OrdersComponent**.

Here's an example of an Angular application that has routing enabled and uses lazy loading to load the **CustomersComponent** and **OrdersComponent**.

In the **customers.component.ts** file, the Syncfusion Calendar component is added as follows,

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

@Component({
  selector: 'app-customers',
  template: <ejs-calendar [value]='dateValue' [min]='minDate' [max]='maxDate'></ejs-calendar>
})
export class CustomersComponent implements OnInit {
  public month: number = new Date().getMonth();
  public fullYear: number = new Date().getFullYear();
  public dateValue: Date = new Date(this.fullYear, this.month, 11);
  public minDate: Date = new Date(this.fullYear, this.month, 9);
  public maxDate: Date = new Date(this.fullYear, this.month, 15);
  constructor() { }
  ngOnInit() {
  }
}
```

In the **orders.component.ts** file, the Syncfusion Grid component is added as follows,

```

`typescript
import { Component, OnInit } from '@angular/core';
import { DataManager, ODataV4Adaptor } from '@syncfusion/ej2-data';
@Component({
  selector: 'app-orders',
  template: ` <ejs-grid [dataSource]='data'>
    <e-columns>
    <e-column field='OrderID' headerText='Order ID' textAlign='Right' width=120></e-column>
    <e-column field='CustomerID' headerText='Customer ID' width=150></e-column>
    <e-column field='ShipCity' headerText='Ship City' width=150></e-column>
    <e-column field='ShipName' headerText='Ship Name' width=150></e-column>
    </e-columns>
  </ejs-grid>`,
})
export class OrdersComponent implements OnInit {
  public data: DataManager;
  constructor() { }
  ngOnInit() {
    this.data = new DataManager({
      url: 'https://services.odata.org/V4/Northwind/Northwind.svc/Orders/?$top=7',
      adaptor: new ODataV4Adaptor()
    });
  }
}
`

```

In the `app-routing.module.ts` file, we've implemented code splitting to dynamically import the components.

```

`typescript
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
const routes: Routes = [
  {
    path: 'customers',

```



```

loadChildren: () => import('./customers/customers.module').then(m => m.CustomersModule)
},
{
  path: 'orders',
  loadChildren: () => import('./orders/orders.module').then(m => m.OrdersModule)
}
];
@NgModule({
  imports: [
    RouterModule.forRoot(routes)
  ],
  exports: [RouterModule],
  providers: []
})
export class AppRoutingModule { }
`

```

In the above code block, we are using `loadChildren` property in the routing configuration to lazily load the `CustomersModule` and `OrdersModule` when the user navigates to the corresponding routes. This way, the application only loads the necessary modules on demand, improving the initial loading time and performance of the application.

Testing

Component Testing in Jasmine/Karma Environment for Angular

This guide explains on how to configure Syncfusion [Angular components](#) within the test bed, enabling interaction with component instances or selectors, leveraging the built-in testing support with the [Jasmine](#) test framework in projects created with [Angular](#) CLI. With this setup, you can seamlessly implement [Unit Testing](#), [Integration Testing](#) and [End-to-End Testing](#) with our components.

Setting Up the Test Environment

To initiate the testing process, we need to configure and integrate an [Angular](#) application with our [Syncfusion Angular components](#). In this guide, we will specifically concentrate on testing the Syncfusion [grid](#) component. Also if you notice a license banner in karma automation browsers, it means that you need to obtain a Syncfusion license key and register it in your application using [npx command](#).

Unit Testing

Unit testing allows you to test specific features or functionalities in your codebase, focusing on individual units such as functions or classes, also helps to ensure that each part of your code works as expected on its own. This allows you to catch and fix issues early in the development process, leading to more robust and reliable application. The code snippet provided showcases how to set up Angular's TestBed configuration and how to test the rendering of grid rows in a Syncfusion grid component.

```
`typescript
```

```
// app.component.spec.ts
import { ComponentFixture, TestBed, fakeAsync, tick } from '@angular/core/testing';
import { AppComponent } from './app.component';
import { GridModule, PageService, EditService } from '@syncfusion/ej2-angular-grids';
describe('AppComponent', () => {
  let component: AppComponent;
  let fixture: ComponentFixture<AppComponent>;
  beforeEach(async () => {
    fixture = await TestBed.configureTestingModule({
      imports: [GridModule],
      providers: [PageService, EditService],
      declarations: [AppComponent],
    }).compileComponents();
    fixture = TestBed.createComponent(AppComponent);
    component = fixture.componentInstance;
  });
  it('Test rendered Grid rows using Selector', fakeAsync(async () => {
    fixture.detectChanges();
    const compiled = fixture.nativeElement as HTMLElement;
    tick(500);
    expect(compiled.querySelector('.e-grid') as HTMLElement).toBeDefined();
    expect((compiled.querySelector('.e-grid') as HTMLElement).classList.length).toBe(9);
    const rows: HTMLElement[] = [].slice.call(compiled.querySelectorAll('.e-row'));
    expect(rows.length).toBe(6);
  }));
  it('Test rendered Grid rows using Instances', fakeAsync(async () => {
    fixture.detectChanges();
    tick(500);
    const component = fixture.componentInstance;
    expect((component as AppComponent).grid?.currentViewData.length).toBe(6);
  }));
});
```

Run `ng test` command to see the test report.

Integration Testing

Integration testing allows you to test a component as a whole, rather than testing individual units separately. This approach treats all units as a single entity and provides a broader view of your application's functionality, helping to uncover issues that may arise from the integration of various components. In the provided code snippet, we have an integration test for the AppComponent with Syncfusion grid using Angular's testing utilities.

```
`typescript
// app.component.spec.ts
import { ComponentFixture, TestBed, fakeAsync, tick } from '@angular/core/testing';
import { AppComponent } from './app.component';
import { GridModule, PageService, EditService } from '@syncfusion/ej2-angular-grids';
const newRecord = {
  OrderID: 10262,
  CustomerID: 'RATTC',
  Freight: 48.29,
  OrderDate: new Date(8379738e5),
};
describe('AppComponent', () => {
  let component: AppComponent;
  let fixture: ComponentFixture<AppComponent>;
  beforeEach(async () => {
    fixture = await TestBed.configureTestingModule({
      imports: [GridModule],
      providers: [PageService, EditService],
      declarations: [AppComponent],
    }).compileComponents();
    fixture = TestBed.createComponent(AppComponent);
    component = fixture.componentInstance;
  });
  it('Add new record dynamically in the Data Grid rows and check the record', fakeAsync(async () => {
    fixture.detectChanges();
    tick(500);
    (component as AppComponent).grid?.addRecord(newRecord, 0);
    tick(500);
```

```
expect(((component as AppComponent).grid?.currentViewData[0] as any).OrderID).toBe(10262);
});
});
`
```

Run `ng test` command to see the test report.

End-to-End Testing

End-to-End testing focuses on evaluating Syncfusion components from the perspective of a user, disregarding the internal structure of the components. In the provided code snippet, the testing framework is programmed to simulate a user's action of clicking on the pager within a Syncfusion grid component. This interaction triggers a response from the application, such as fetching and displaying additional records. The test then visually inspects the outcome of this interaction to ensure that the expected behavior is achieved.

```
`typescript
// app.component.spec.ts
import { ComponentFixture, TestBed, fakeAsync, tick } from '@angular/core/testing';
import { AppComponent } from './app.component';
import { GridModule, PageService, EditService } from '@syncfusion/ej2-angular-grids';

const newRecord = {
  OrderID: 10262,
  CustomerID: 'RATTC',
  Freight: 48.29,
  OrderDate: new Date(8379738e5),
};

describe('AppComponent', () => {
  let component: AppComponent;
  let fixture: ComponentFixture<AppComponent>;

  beforeEach(async () => {
    fixture = await TestBed.configureTestingModule({
      imports: [GridModule],
      providers: [PageService, EditService],
      declarations: [AppComponent],
    }).compileComponents();
    fixture = TestBed.createComponent(AppComponent);
    component = fixture.componentInstance;
  });
```

```
it('Render the Data Grid and navigate its pages', fakeAsync( async () => {  
  fixture.detectChanges();  
  tick(500);  
  const compiled = fixture.nativeElement as HTMLElement;  
  (component as AppComponent).grid?.addRecord(newRecord, 0);  
  tick(500);  
  const event = new MouseEvent('click', {  
    bubbles: true,  
    cancelable: true,  
    view: window  
  });  
  (compiled.querySelector('.e-pager-default') as HTMLElement).dispatchEvent(event);  
  tick(500);  
  expect((component as AppComponent).grid?.currentViewData.length).toBe(3);  
  });  
});  
`
```

Run `ng test` command to see the test report.

See also

- [Unit Jasmine Testing Documentation](#)
- [Selenium E2E Testing using Protractor in Angular](#)

Angular Cypress Testing

This document explains how to perform the E2E and Component testing with Syncfusion Angular components in Angular web applications using Cypress.

Cypress

Cypress is a JavaScript-based End-to-End (E2E) testing framework and a next-generation front-end testing tool designed for modern web applications. Cypress is designed to make testing web applications easier, more efficient, and reliable.

For more information about Cypress, refer to this [documentation](#).

Integrate Cypress with Angular

To integrate Cypress with Angular, follow the below steps.

1.Create the angular application and add the Syncfusion DataGrid component by following the [getting started](#) documentation. 2.Once create the application, run the below command to install the Cypress.

CMD

```
ng add @cypress/schematic
```

```

1 Using package manager: npm
✓ Found compatible package version: @cypress/schematic@2.5.1.
✓ Package information loaded.

The package @cypress/schematic@2.5.1 will be installed and executed.
Would you like to proceed? Yes
✓ Packages successfully installed.
? Would you like the default 'ng e2e' command to use Cypress? [ Protractor to Cypress Migration Guide: https://on.cypress.io/protractor-to-cypress?cli=true ] Yes
? Would you like to add Cypress component testing? This will add all files needed for Cypress component testing. Yes
CREATE cypress.config.ts (264 bytes)
CREATE cypress/tsconfig.json (139 bytes)
CREATE cypress/e2e/spec.cy.ts (143 bytes)
CREATE cypress/fixtures/example.json (85 bytes)
CREATE cypress/support/commands.ts (1377 bytes)
CREATE cypress/support/e2e.ts (649 bytes)
CREATE cypress/support/component-index.html (290 bytes)
CREATE cypress/support/component.ts (1123 bytes)
UPDATE package.json (1209 bytes)
UPDATE angular.json (4284 bytes)
✓ Packages installed successfully.

```

Cypress Testing types

Cypress supports two [types of testing](#). Users can choose the testing type based on their requirements.

- [E2E Testing](#)
- [Component Testing](#)

For Cypress testing type comparison, refer to this [documentation](#).

Cypress E2E Testing of Syncfusion Angular Components

The following steps explain how to test the Angular DataGrid component using Cypress E2E testing.

1.Add the following code snippet to test the DataGrid component in the `./cypress/e2e/spec.cy.ts` file.

SPEC.CY.TS

```

describe('My First Test', () => {
  it('Visits the initial project page', () => {
    cy.visit('/')
  })
  it('should contain Grid rows', () => {
    cy.visit('/')
    cy.get('.e-grid').should('be.visible')
    cy.get('.e-grid').find('.e-row').should('have.length', 3)
  })
})

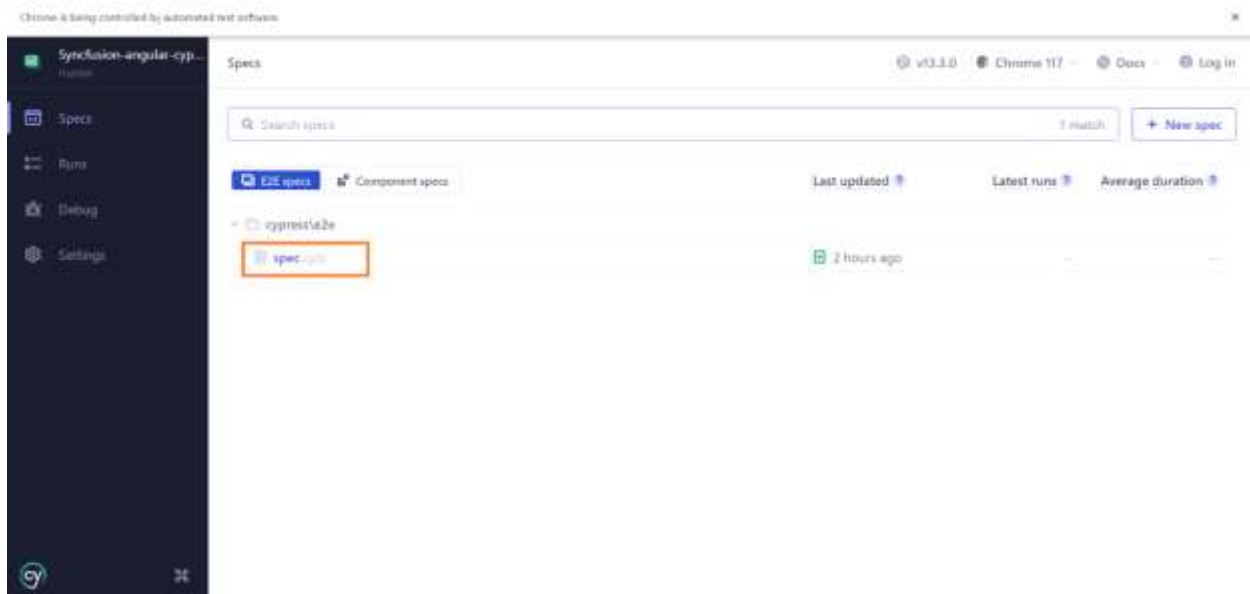
```

2.To start the test cases, run the following command.

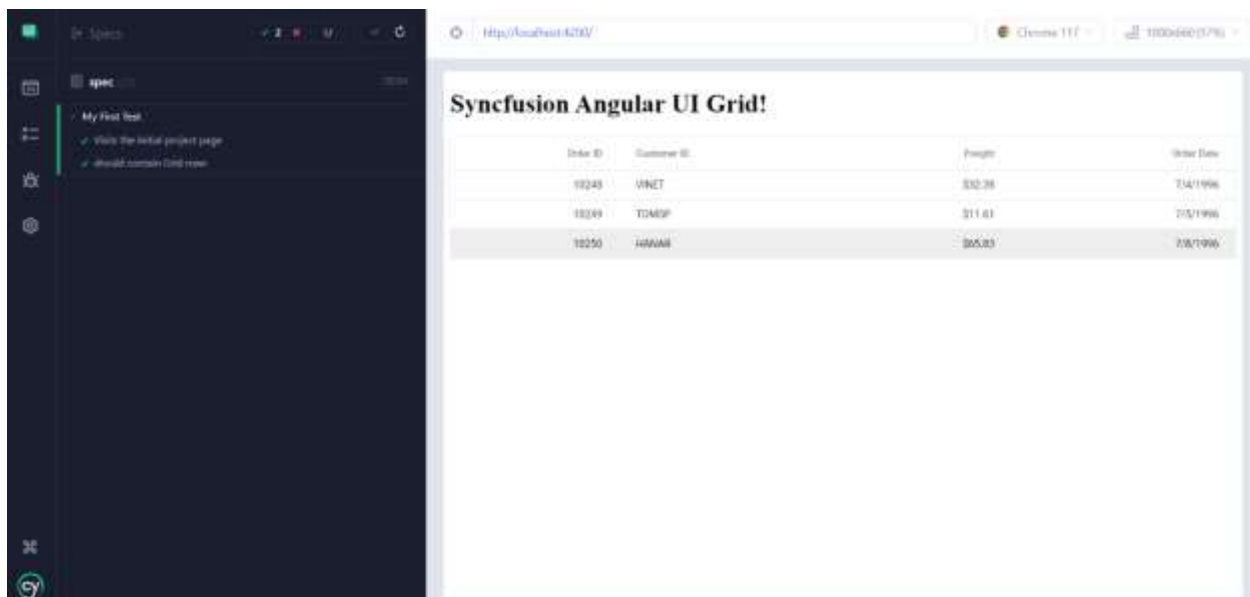
CMD

```
ng e2e
```

3.This will opens the dashboard. Start the E2E testing and click the `spec.cy.ts` file to run the test cases.



4. Once the test cases are completed, the result will be displayed as follows.



For more information about Cypress E2E testing, refer to this [documentation](#).

Cypress Component Testing of SynCFusion Angular Components

The following steps explain how to test the Angular DataGrid component in [Cypress component testing](#).

1. Create a new file `app.component.cy.ts` in the `./src/app` folder. 2. Then add the below code snippet in the `app.component.cy.ts` file to test the DataGrid component.

APP.COMPONENT.CY.TS

```
import { AppComponent } from './app.component';
describe('AppComponent', () => {
  it('should contain syncfusion Grid sample', () => {
    cy.mount(AppComponent)
    cy.get('.e-grid').should('be.visible')
  })
})
```

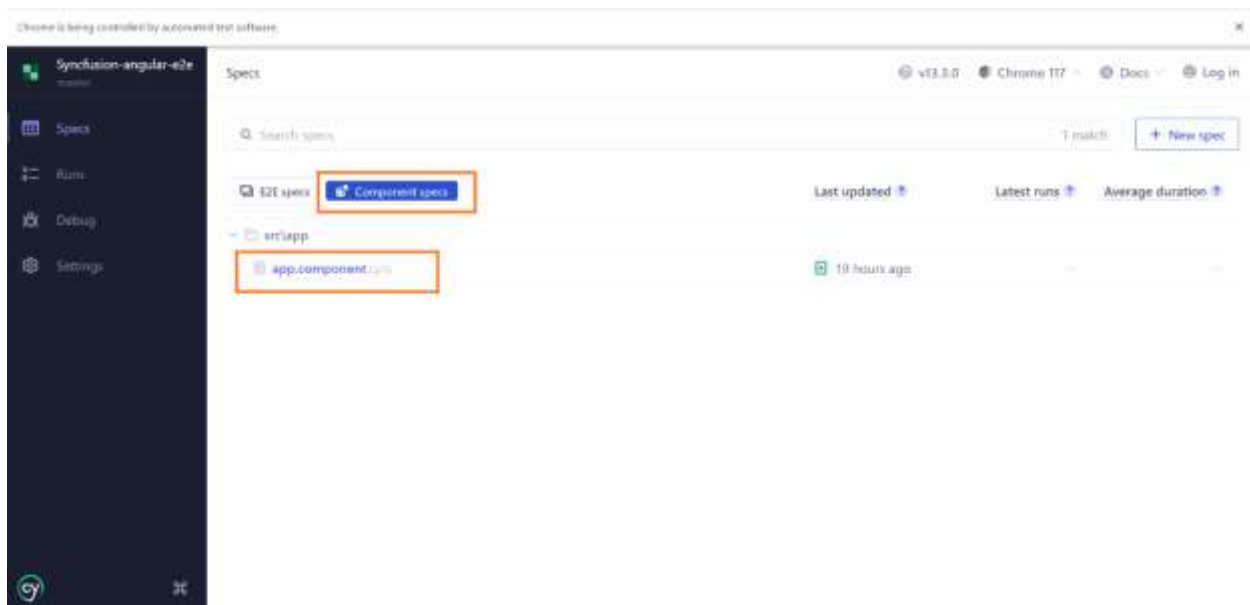
```
  })  
  it('should contain Grid rows', () => {  
    cy.mount(AppComponent)  
    cy.get('.e-grid').find('.e-row').should('have.length', 3)  
  })  
})
```

3.To start the test cases, run the following command.

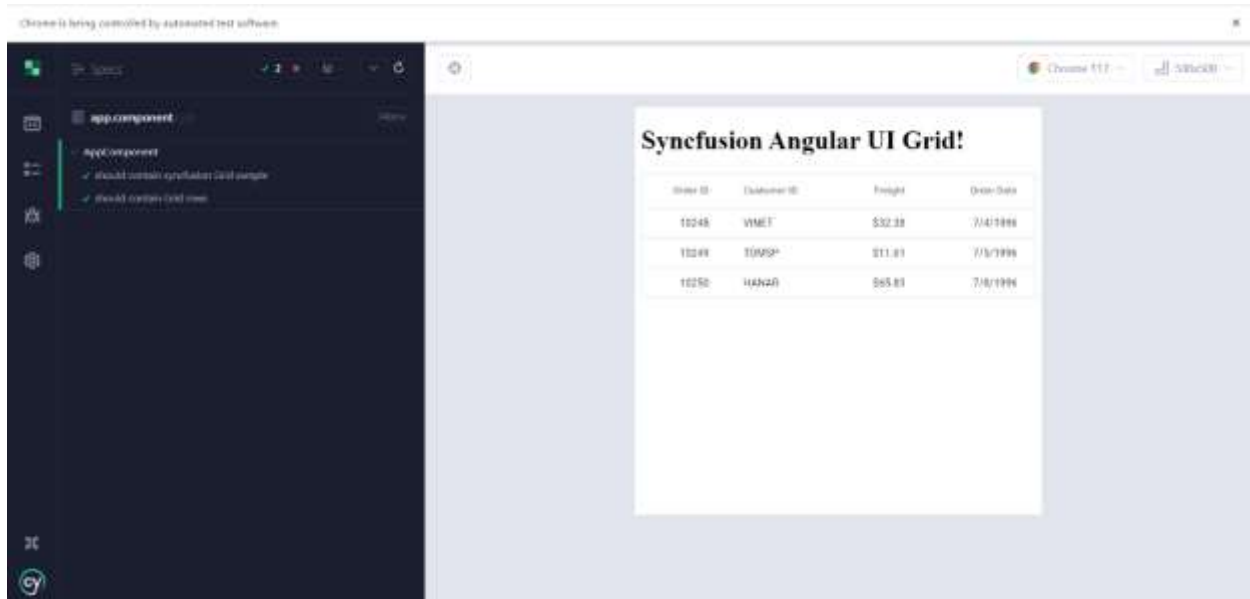
CMD

```
ng e2e
```

4.This will opens the dashboard. Switch to component testing type and click the `app.component.cy.ts` file to run the test cases.



5.Once the test cases are completed, the result will be displayed as follows.



6.To resolve the license banner in the automation browsers, [register the Syncfusion license key](#) in the `./cypress/support/component.ts` file as follows.

APP.COMPONENT.CY.TS

```
import { registerLicense } from '@syncfusion/ej2-base';
// Registering Syncfusion license key
registerLicense('License Key');
```

[View the Syncfusion Angular Cypress testing sample on GitHub](#)

Getting Started for Selenium E2E Testing using Protractor in Angular

This documents explains the selenium E2E testing for Angular web applications using **Protractor**.

Note: Starting from Angular version 12, the Angular CLI no longer includes Protractor by default and has plans to remove Protractor support in the future. As a result, Protractor reached its end-of-life in August 2023. For more information, refer to this [article](#).

Selenium

Selenium is the most preferred framework for End-to-End (E2E) testing of the web applications. But users cannot utilize Selenium directly for the dynamic web applications such as Angular.

Normally, HTML elements are located using CSS selectors such as ID, Class, attributes, and values in Selenium. However, when working with Angular applications, due to synchronization issues, you cannot identify Angular-specific attributes like `ng-model`, `ng-class`, `ng-bind`, and `ng-options` using regular CSS selectors.

Protractor is specifically developed to overcome the above problem in Selenium E2E testing for Angular web applications.

Protractor

Protractor is a wrapper around WebDriverJS, which is the JavaScript binding for the WebDriver API. It is an End-to-End (E2E) testing framework specifically developed to automate Angular web applications using new locator strategies.

For more information about the list of Protractor API, refer to this [documentation](#).

E2E Testing for Angular

Using Protractor, users can test the web application either using **in-build ChromeDriver** of Chrome browser or **Selenium Standalone server**.

Protractor with ChromeDriver

To setup the Protractor with ChromeDriver, set the **directConnect** property to true in the **e2e/protractor.conf.js** configuration file as shown below.

APP.COMPONENT.CY.TS

```
exports.config = {
  allScriptsTimeout: 11000,
  specs: [
    './src/**/*.e2e-spec.ts'
  ],
  capabilities: {
    'browserName': 'chrome'
  },
  directConnect: true,
  baseUrl: 'http://localhost:4200/',
  framework: 'jasmine',
  jasmineNodeOpts: {
    showColors: true,
    defaultTimeoutInterval: 30000,
    print: function() {}
  },
  onPrepare() {
    require('ts-node').register({
      project: require('path').join(__dirname, './tsconfig.json')
    });
    jasmine.getEnv().addReporter(new SpecReporter({ spec: { displayStacktrace:
      true } }));
  }
};
```

The Angular web application created using Angular CLI prior to v12 will automatically include the configuration and setup for the built-in ChromeDriver Protractor E2E testing, along with a sample test case.

Protractor with Standalone Selenium Server

To set up Protractor with a Selenium Standalone server, follow these steps:

1. Open the Protractor configuration **e2e/protractor.conf.js** file.
2. Set the **directConnect** property to false.
3. Set the **seleniumAddress** property to **http://localhost:4444/wd/hub**.
4. Ensure that the Selenium server is up and running before executing the test cases. The Protractor configuration for Selenium standalone as shown in the following code block.

APP.COMPONENT.CY.TS

```
exports.config = {
  allScriptsTimeout: 11000,
```

```

specs: [
  './src/**/*.e2e-spec.ts'
],
capabilities: {
  'browserName': 'chrome'
},
directConnect: false,
baseUrl: 'http://localhost:4200/',
seleniumAddress: 'http://localhost:4444/wd/hub',
framework: 'jasmine',
jasmineNodeOpts: {
  showColors: true,
  defaultTimeoutInterval: 30000,
  print: function() {}
},
onPrepare() {
  require('ts-node').register({
    project: require('path').join(__dirname, './tsconfig.json')
  });
  jasmine.getEnv().addReporter(new SpecReporter({ spec: { displayStacktrace:
    true } }));
}
};

```

To install/update and start the Selenium server, run the following commands before running the test cases and maintain the server in active state until the testing is finished.

CMD

```

webdriver-manager update
webdriver-manager start

```

Testing Syncfusion Angular Component using Protractor

The Syncfusion Angular Components are accessed using Protractor locators like ID, CSS, name, and more. The following code samples explain how to use the Angular button component in Protractor testing.

1.To install the Angular Button component, refer to the [Getting Started](#) documentation. 2.Set the ID to the Button component in the `./src/app/app.component.ts` file as follows.

APP.COMPONENT.TS

```

import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
  template: `<!-- To render Button. -->
<button id="my_button" ejs-button>Button</button>`
})
export class AppComponent { }

```

2.Get the EJ2 Angular Button component using the Protractor locator in the `./e2e/src/app.po.ts` file as follows.

APP.PO.TS

```
element(by.id('my_button')).getText()
```

3.To check the actual and expected value, use the `expect` method in the `./e2e/src/app.e2e-spec.ts` file as follows.

APP.E2E-SPEC.TS

```
import { AppPage } from './app.po';
it('should display welcome message', () => {
  page.navigateTo();
  // console.log(page.getTitleText());
  expect(page.getTitleText()).toEqual('BUTTON');
});
```

4.To start the test cases, run the following command.

PROTRACTOR.CONF.JS

```
ng e2e
```

See also

- [Migration from Protractor to Cypress](#)
- [Cypress Testing of Syncfusion Angular Components](#)

Upgrade

Release History

Every new release of Syncfusion includes exciting new features. Refer to the Syncfusion Angular [release notes](#) to know more about the changes in each releases.

See our [Upgrade Guide](#) for Angular to learn more about the following.

- Breaking Changes
- Bug Fixes
- Features
- Known Issues between your current version and the latest version you are trying to upgrade.

Syncfusion Angular supported versions

Angular version compatibility

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

Version	Syncfusion Angular components version
-----	-----
Angular v17	23.2.4 and above
Angular v16	21.1.39 and above
Angular v15	20.4.38 and above

[Angular v14](#)	20.2.36 and above
[Angular v13](#)	19.4.38 and above
[Angular v12](#)	19.3.43 and above
[Angular v11](#)	18.4.31 and above
[Angular v10](#)	18.2.55 and above
[Angular v9](#)	17.4.51 and above
[Angular v8](#)	17.1.50 and above
[Angular v7](#)	16.3.32 and above

By default, Syncfusion Angular packages ($\geq 20.2.36$) support the [Angular Ivy distribution](#). These packages are compatible with Angular versions 12 and above. After this release ($\geq 20.2.36$), you need to add the suffix `-ngcc` along with the package version (`@syncfusion/ej2-angular-grids:"20.2.38-ngcc"`) in the `package.json` file. For more information on Angular package installation, see [Angular package installation](#).

Syncfusion version information

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

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

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

See also

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

Upgrading Syncfusion JavaScript (Essential JS2)

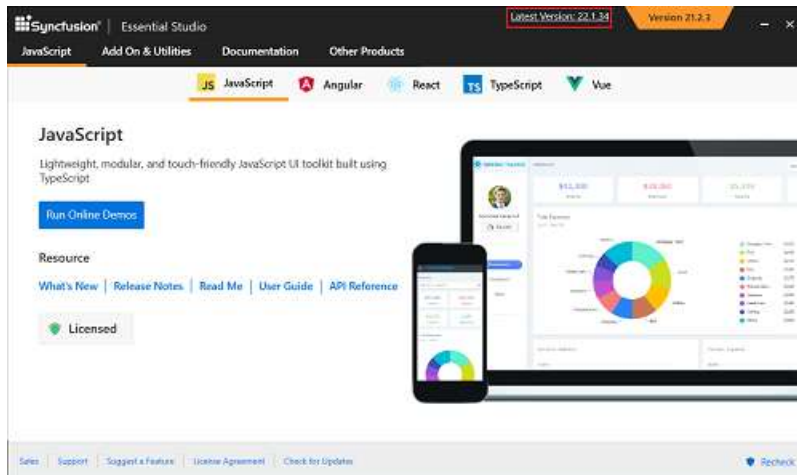
Syncfusion releases new volumes once every three months, with exciting new features. There will be one Service Pack release for these volume releases. Service Pack releases are provided to address major bug fixes in the volume releases.

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

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

Upgrading to the Latest Version

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



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

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

Upgrade from Trial Version to License Version

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

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

Visual Studio Code Integration

Visual Studio Code Integration

Overview

The Syncfusion Essential Studio Web extension for Visual Studio Code allows you to use the Syncfusion JavaScript components (React, Angular, and Vue) easily by configuring the Syncfusion NPM packages and themes.

The Syncfusion Web Extension provides the following support in Visual Studio Code:

[Project-Template](#): Creates Syncfusion Web applications by adding the required Syncfusion JavaScript components.

[Code Snippet](#): Adds a Syncfusion Angular component with various features to the Angular Application's HTML code editor.

Download and Installation

Syncfusion publishes the Visual Studio Code extension in [Visual Studio Code marketplace](#). You can either install it from Visual Studio Code or download and install it from the Visual Studio Code marketplace.

Prerequisites

The following prerequisites software needs to be installed for the Syncfusion Web extension installation and for creating the Syncfusion Web applications along with any one of the Framework(React, Angular, and Vue).

- [Visual Studio Code](#)

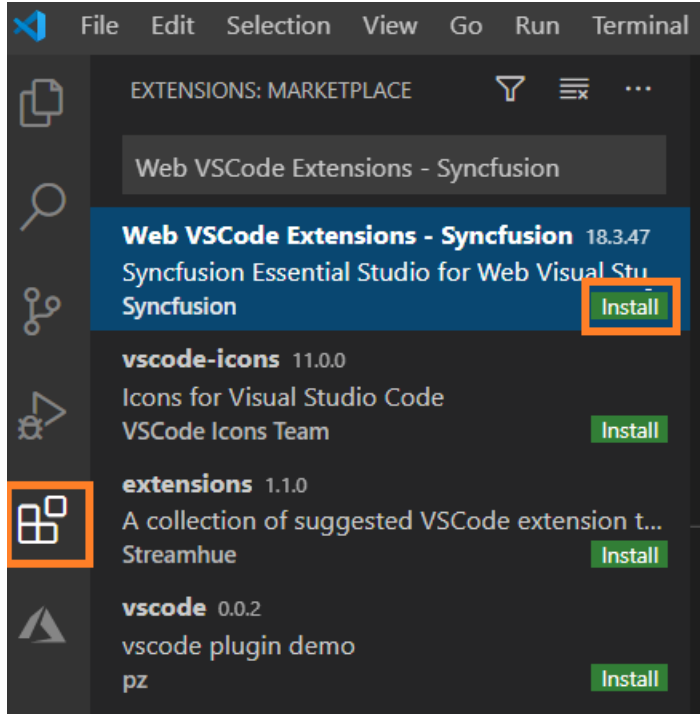
The minimum version of the Visual Studio Code is 1.38.0 to use the Syncfusion Web Extension.

- [C# Extension](#)
- [Node.js](#)

Install through the Visual Studio Code Extensions

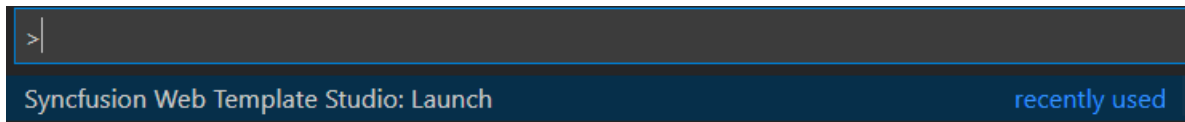
The following steps explain how to install the Syncfusion Web extensions from Visual Studio Code Extensions.

1. Open Visual Studio Code.
2. Go to **View > Extensions**, and open Manage Extensions.
3. Type “**Syncfusion Web**” in the search box.



4. Click the Install button on the “**Web VSCode Extensions - Syncfusion**” extension.

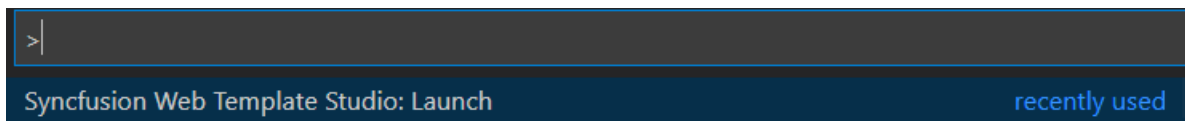
5. After the installation, reload the Visual Studio Code using the **Reload Required** in Visual Studio Code palette.
6. Now, use the Syncfusion Web extensions from the Visual Studio Code palette.



Install from the Visual Studio Code Marketplace

The following steps explain how to download Syncfusion Web applications from the Visual Studio Code Marketplace and install them.

1. Open the [Syncfusion Web Extension](#)
2. Click Install from Visual Studio Code Marketplace. The browser opens the popup with the information like **“Open Visual Studio Code”**. Click Open Visual Studio Code, then [Syncfusion Web Extension](#) will open in Visual Studio Code.
3. Click the Install button in the **“Web VSCode Extensions - Syncfusion”** extension.
4. After the installation, reload the Visual Studio Code using the Reload Required in Visual Studio Code palette.
5. Now, use the Syncfusion Web extensions from the Visual Studio Code palette.



Visual Studio Code Extensions

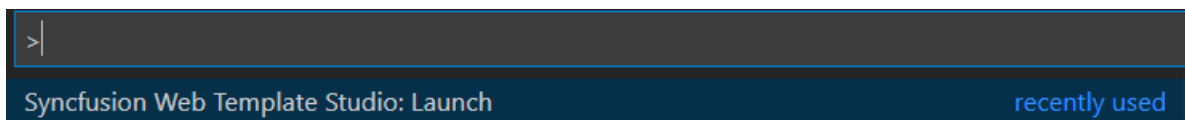
Create project

Syncfusion provides **project templates** for **Visual Studio Code** to create Syncfusion Web applications. The Syncfusion Web Project template creates applications with the selected Framework (React, Angular, and Vue), required Syncfusion NPM packages, component render code for the Grid, Chart, and Scheduler components, and a style to make development with Syncfusion components easier.

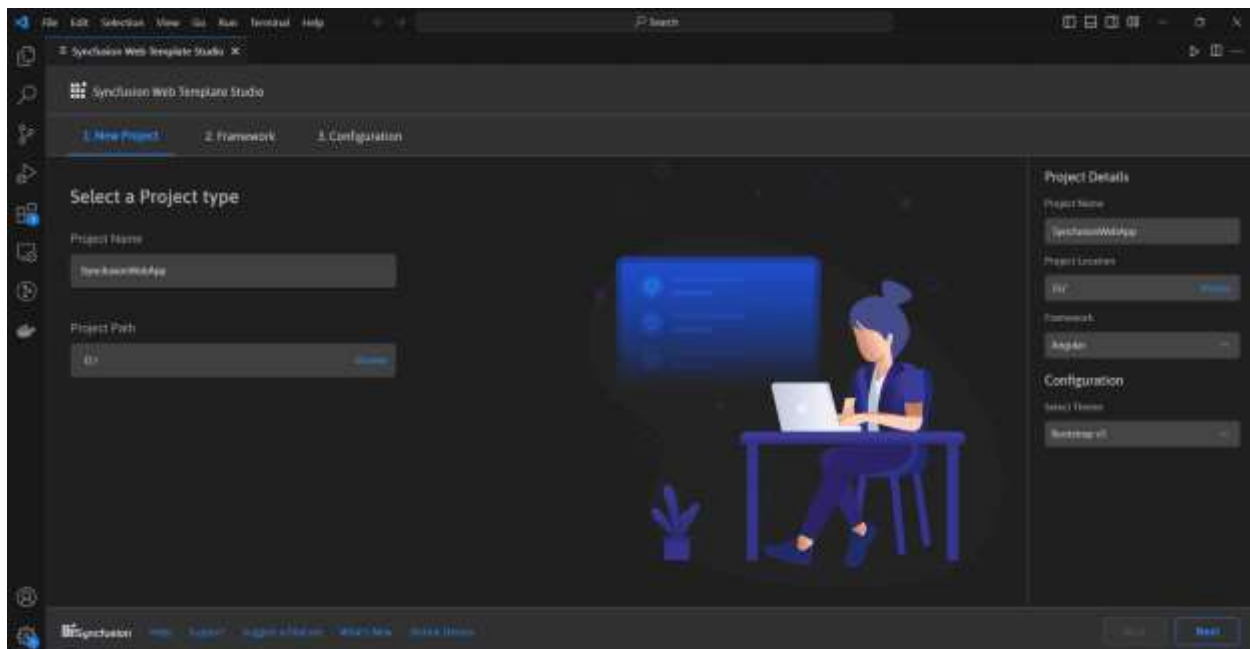
The Syncfusion Visual Studio Code project template provides support for Web project templates from v18.3.0.47.

The steps below help you to create **Syncfusion Web Applications** through the **Visual Studio Code**:

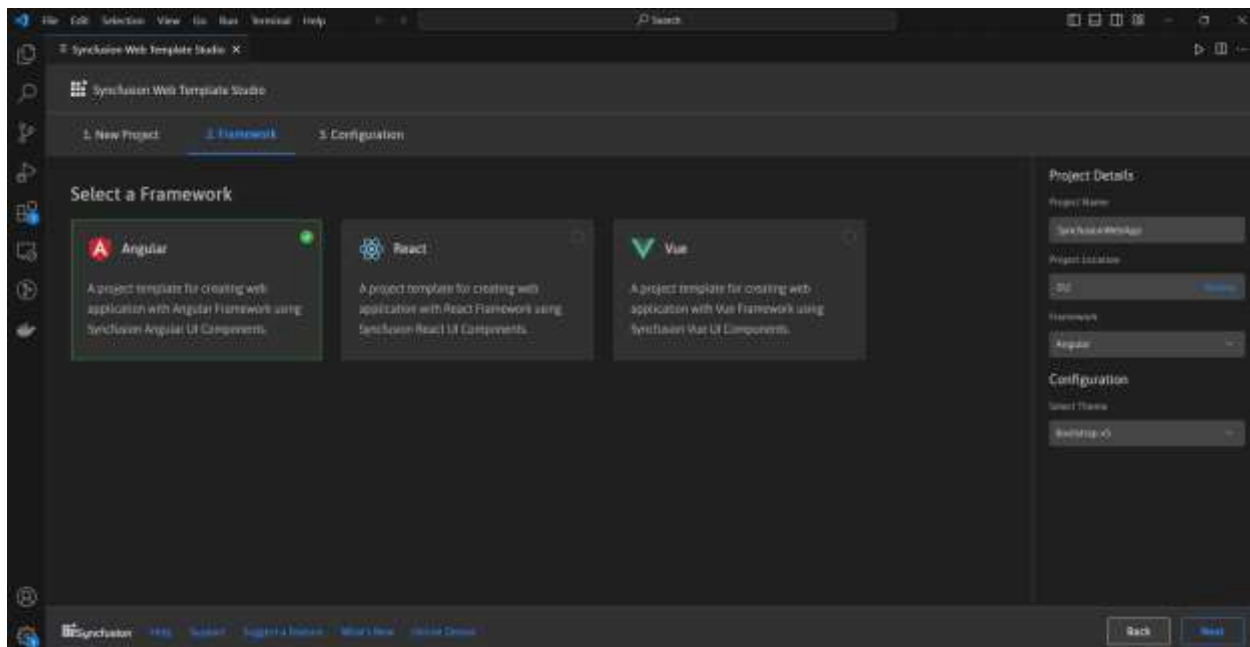
1. In Visual Studio Code, open the command palette by pressing Ctrl+Shift+P. The Visual Studio Code palette opens, search the word Syncfusion, so you can get the templates provided.



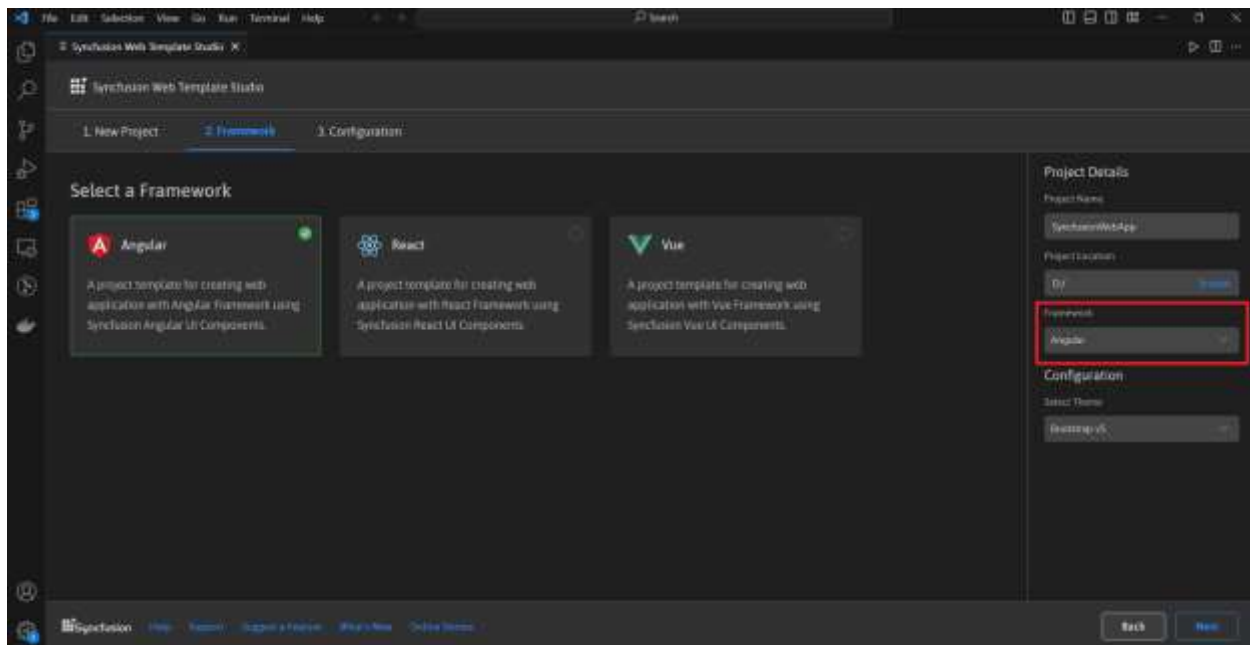
2. Select **Syncfusion Web Template Studio: Launch** and then press enter, Template Studio wizard for configuring the Syncfusion Web app will appear. Provide the require Project Name and Path to create the new Syncfusion Web application along with any one of the Framework (React, Angular, and Vue).



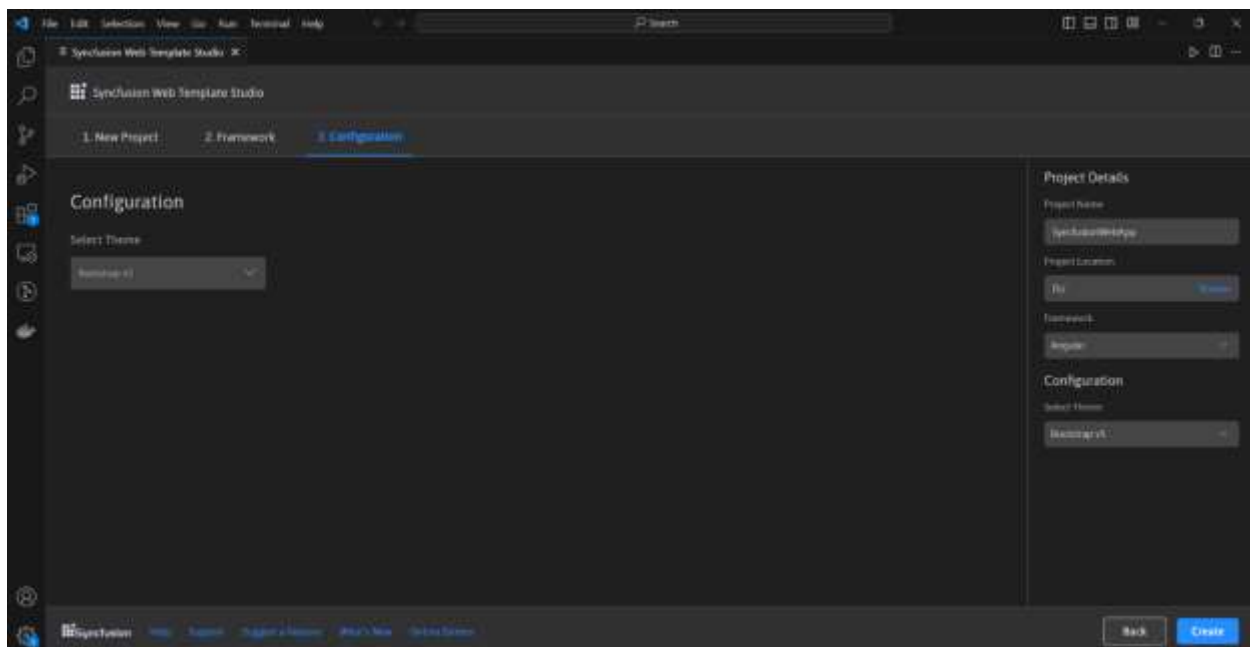
3. Click either **Next** or **Framework** tab, the Framework types will be appearing and choose any one of the Framework:
- Angular
 - React
 - Vue



If you choose the Angular framework, it will appear in the **Project Details** section. You can then create the Angular application.



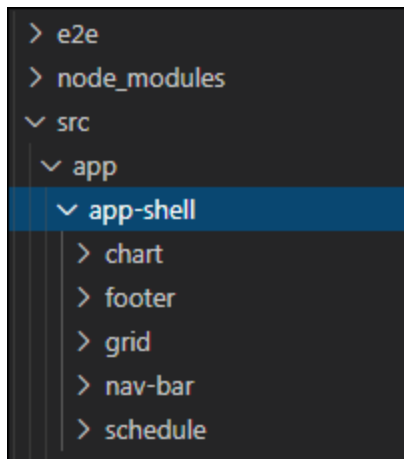
4. Click either **Next** or the **Configuration** tab, and the Configuration section will be loaded. Choose the preferred theme and then click **Create**. The project will be created.



5. The created Syncfusion Web App is configured with the Syncfusion NPM packages, styles, and the component render code for the Syncfusion component added.

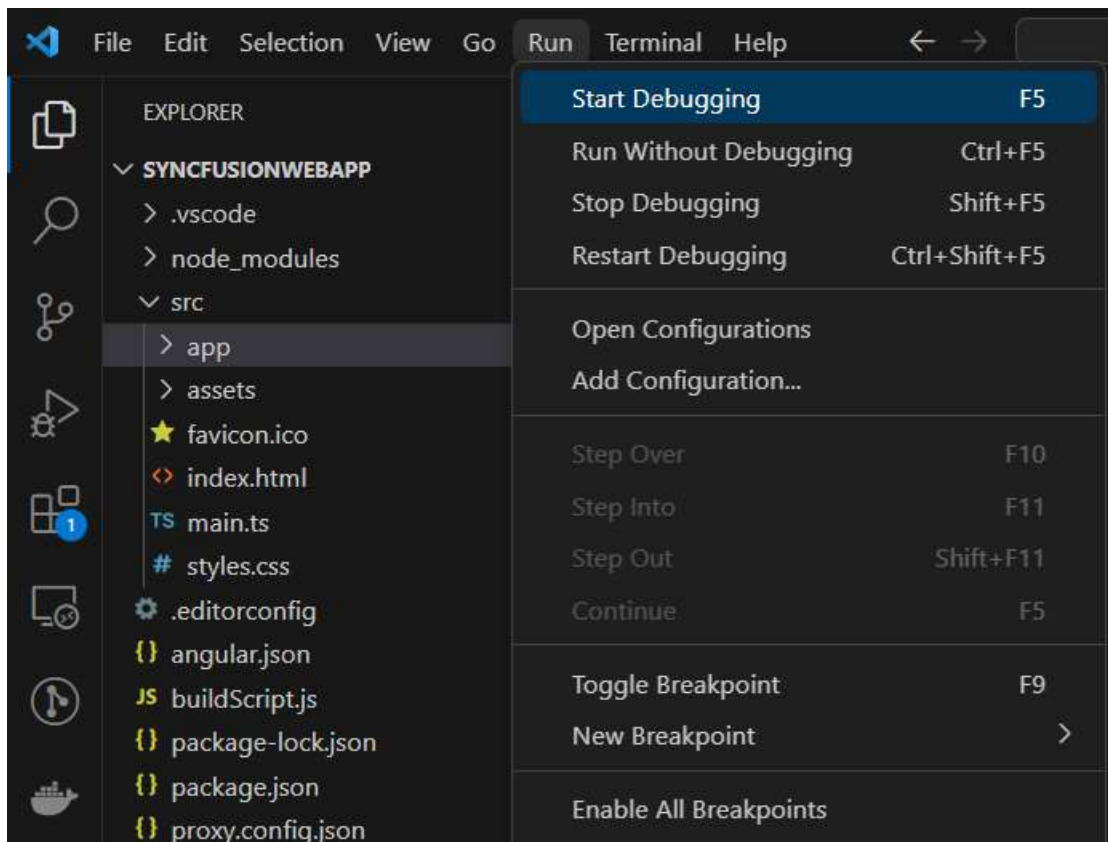
```
"dependencies": {  
  "@angular/animations": "^16.0.0",  
  "@angular/common": "^16.0.0",  
  "@angular/compiler": "^16.0.0",  
  "@angular/core": "^16.0.0",  
  "@angular/forms": "^16.0.0",  
  "@angular/platform-browser": "^16.0.0",  
  "@angular/platform-browser-dynamic": "^16.0.0",  
  "@angular/router": "^16.0.0",  
  "@syncfusion/ej2-angular-charts": "^25.1.35",  
  "@syncfusion/ej2-angular-grids": "^25.1.35",  
  "@syncfusion/ej2-angular-schedule": "^25.1.35",  
  "bootstrap": "5.2.3",  
  "concurrently": "^7.4.0",  
  "rxjs": "~7.8.0",  
  "tslib": "^2.3.0",  
  "zone.js": "~0.13.0"  
},
```

```
# styles.css ×  
src > # styles.css > ...  
1  /* You can add global styles to this file, and also import other style files */  
2  @import '~bootstrap/dist/css/bootstrap.min.css';  
3  @import '../node_modules/@syncfusion/ej2-base/styles/bootstrap5.css';  
4  @import '../node_modules/@syncfusion/ej2-buttons/styles/bootstrap5.css';  
5  @import '../node_modules/@syncfusion/ej2-calendars/styles/bootstrap5.css';  
6  @import '../node_modules/@syncfusion/ej2-dropdowns/styles/bootstrap5.css';  
7  @import '../node_modules/@syncfusion/ej2-inputs/styles/bootstrap5.css';  
8  @import '../node_modules/@syncfusion/ej2-navigations/styles/bootstrap5.css';  
9  @import '../node_modules/@syncfusion/ej2-popups/styles/bootstrap5.css';  
10 @import '../node_modules/@syncfusion/ej2-splitbuttons/styles/bootstrap5.css';  
11 @import '../node_modules/@syncfusion/ej2-angular-grids/styles/bootstrap5.css';  
12 @import '../node_modules/@syncfusion/ej2-base/styles/bootstrap5.css';  
13 @import '../node_modules/@syncfusion/ej2-buttons/styles/bootstrap5.css';  
14 @import '../node_modules/@syncfusion/ej2-calendars/styles/bootstrap5.css';  
15 @import '../node_modules/@syncfusion/ej2-dropdowns/styles/bootstrap5.css';  
16 @import '../node_modules/@syncfusion/ej2-inputs/styles/bootstrap5.css';  
17 @import '../node_modules/@syncfusion/ej2-lists/styles/bootstrap5.css';  
18 @import '../node_modules/@syncfusion/ej2-popups/styles/bootstrap5.css';  
19 @import '../node_modules/@syncfusion/ej2-navigations/styles/bootstrap5.css';  
20 @import '../node_modules/@syncfusion/ej2-angular-schedule/styles/bootstrap5.css';  
21
```



Run the application

1. Click on **F5** or navigate to **Run>Start debugging**



2. After compilation process completed, open the local host link in browser to see the output.



SyncfusionAngularApp24

Grid Chart Schedule

Syncfusion Angular Grid

Drag a column header here to group its column.

Order ID	Customer ID	Freight	Order Date
10248	VINET	\$32.38	7/4/1996
10249	TOUSP	\$11.61	7/5/1996
10250	HANAR	\$65.83	7/8/1996
10251	VICTE	\$41.34	7/9/1996
10252	SUPRO	\$51.30	7/9/1996
10253	HANAR	\$58.17	7/10/1996

1 of 3 pages (15 items)

Add Syncfusion Angular component in the Angular application

The Syncfusion Angular code snippet utility for Visual Studio Code provides snippets for adding a Syncfusion Angular component with various features in the html code editor file of the Angular Application.

The Syncfusion Angular code snippet is available from Essential Studio 2021 Vol 3 (v19.3.0.43).

Add a Syncfusion Angular component

The following steps help you to use the Syncfusion Angular code snippet in your Angular Application.

1. In Visual Studio Code, open an existing Angular Application or create a new Angular Application.
2. Open the html file that you need and place the cursor in required place where you want to add Syncfusion component.
3. You can find the Syncfusion Angular component with the various features by typing the **ejs** word in the format shown below.

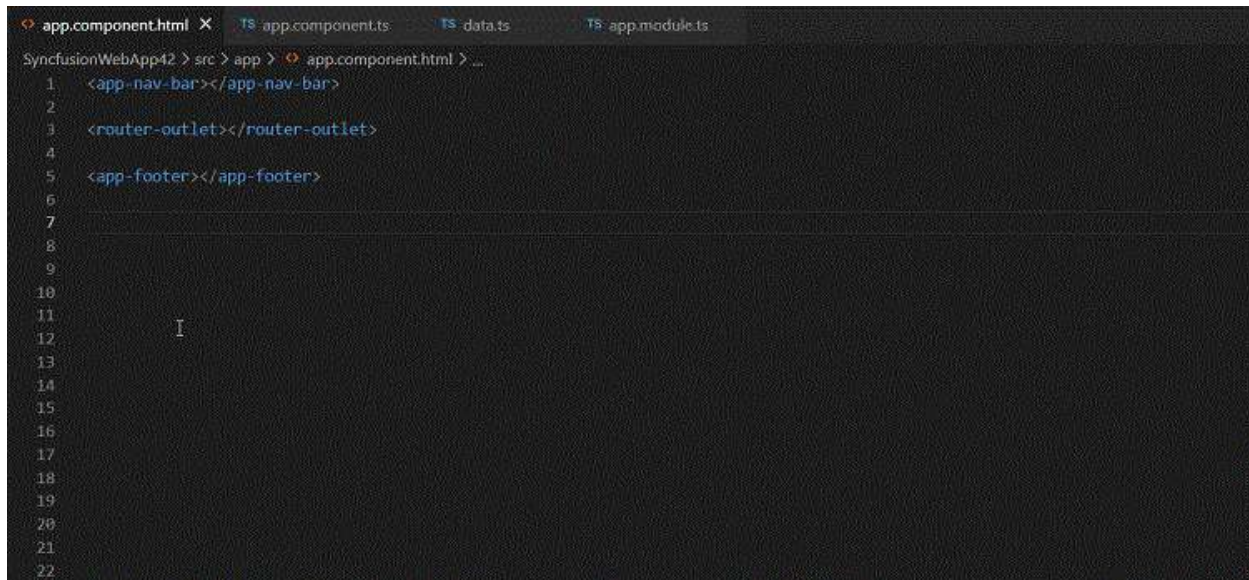
``bash`

`ejs-<Syncfusion component name>-<Syncfusion component feature>`

For Example, `ejs-grid-grouping`

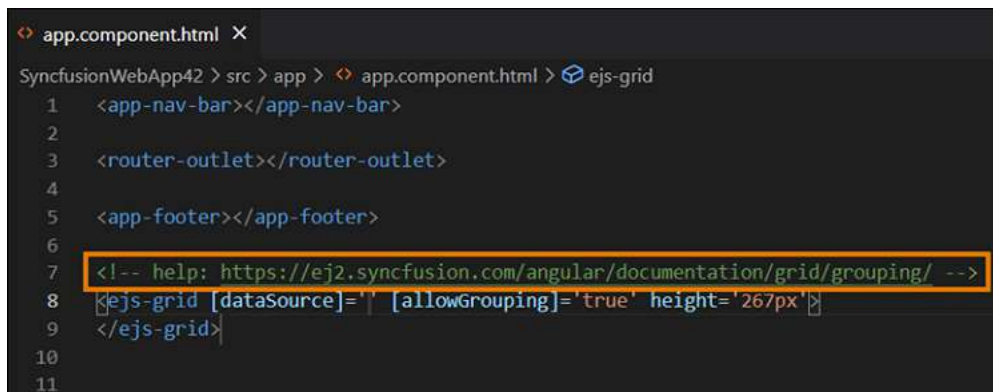
,

4. Choose the Syncfusion component and click the **Enter** or **Tab** key, the Syncfusion Angular component will be added in the html file.



```
app.component.html X TS app.component.ts TS data.ts TS app.module.ts
SyncfusionWebApp42 > src > app > app.component.html > ...
1 <app-nav-bar></app-nav-bar>
2
3 <router-outlet></router-outlet>
4
5 <app-footer></app-footer>
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
```

5. After adding the Syncfusion Angular component to the html file, use the tab key to fill in the required values to render the component with data. You can also find the Syncfusion help link at the top of the added snippet to learn more about the new Syncfusion Angular component feature.



```
app.component.html X
SyncfusionWebApp42 > src > app > app.component.html > ejs-grid
1 <app-nav-bar></app-nav-bar>
2
3 <router-outlet></router-outlet>
4
5 <app-footer></app-footer>
6
7 <!-- help: https://ej2.syncfusion.com/angular/documentation/grid/grouping/ -->
8 <ejs-grid [dataSource]='' [allowGrouping]='true' height='267px'>
9 </ejs-grid>
10
11
```

Configure Angular application with Syncfusion

The Syncfusion Angular snippet only add the code snippet alone in the html file. You need to configure the Angular application with Syncfusion by adding the required Syncfusion Angular NPM, component modules, and themes by manually. To configure, refer the steps below:

1. Open the Angular package.json file and add the required Syncfusion Angular individual NPM package(s) for the Syncfusion Angular components manually. Then navigate to the packages.json file location in the command prompt and run the **npm install** command to restore all the Syncfusion Angular NPM packages.

```
{ } package.json X
SyncfusionWebApp42 > { } package.json > { } dependencies
1 {
2   "name": "syncfusionangularproject",
3   "version": "0.1.0",
4   "private": true,
5   "dependencies": {
6     "@angular/animations": "10.0.2",
7     "@angular/common": "10.0.2",
8     "@angular/compiler": "10.0.2",
9     "@angular/core": "10.0.2",
10    "@angular/forms": "10.0.2",
11    "@angular/localize": "10.0.2",
12    "@angular/platform-browser": "10.0.2",
13    "@angular/platform-browser-dynamic": "10.0.2",
14    "@angular/router": "10.0.2",
15    "@ng-bootstrap/ng-bootstrap": "6.1.0",
16    "@syncfusion/ej2-angular-buttons": "*",
17    "@syncfusion/ej2-angular-charts": "*",
18    "@syncfusion/ej2-angular-gantt": "*",
19    "@syncfusion/ej2-angular-grids": "*",
20    "@syncfusion/ej2-angular-schedule": "*",
```

2. Open your module file and add the required Syncfusion Angular component(s) module entries to render the Syncfusion components in your application.


```
TS app.module.ts X
SyncfusionWebApp42 > src > app > TS app.module.ts > ...
4 import { AppComponent } from './app.component';
5 import { NavBarComponent } from './app-shell/nav-bar/nav-bar.component';
6 import { FooterComponent } from './app-shell/footer/footer.component';
7 import { ServiceInterceptor } from './service.interceptor';
8 import { GridAllModule } from '@syncfusion/ej2-angular-grids';
9
10 @NgModule({
11   declarations: [AppComponent, NavBarComponent, FooterComponent],
12   imports: [
13     BrowserModule, GridAllModule
14   ],
15   providers: [
16     {
17       provide: HTTP_INTERCEPTORS,
18       useClass: ServiceInterceptor,
19       multi: true
20     }
21   ],
22   bootstrap: [AppComponent]
23 })
24 export class AppModule { }
25
```

3. Add the Syncfusion Angular [theme](#) entry in the **style.css** file.

```
# styles.css X
SyncfusionWebApp42 > src > # styles.css > ...
1 /* You can add global styles to this file, and also import other style files */
2 @import '~bootstrap/dist/css/bootstrap.min.css';
3 @import 'https://cdn.syncfusion.com/ej2/material.css';
4
```

Visual Studio Integration

Visual Studio Integration

Overview

The Syncfusion Essential Studio for Angular extensions for Visual Studio that allow you to use the Syncfusion component easier by creating Syncfusion Angular application in Visual Studio and option upgrading the Syncfusion version from the application.

The Syncfusion Angular provides the following supports in Visual Studio:

1. [Project template](#): Creates the Syncfusion Angular application by adding the required Syncfusion Angular components.
2. [Convert project](#): Converts an existing Angular application into a Syncfusion Angular application by adding the required Syncfusion NPM and resource files.

3. [Migrate project](#): Upgrades the existing Syncfusion Angular application from one Essential Studio version to another version.

> In Visual Studio 2017, you can see the Syncfusion menu directly in the Visual Studio menu.

Visual Studio Extensions

Create project

Syncfusion provides the **Visual Studio Project Templates** for create the Syncfusion Angular Application. The Syncfusion Angular application creates the application with the required Syncfusion references, namespaces and CDN links for making the development earlier with the Syncfusion components.

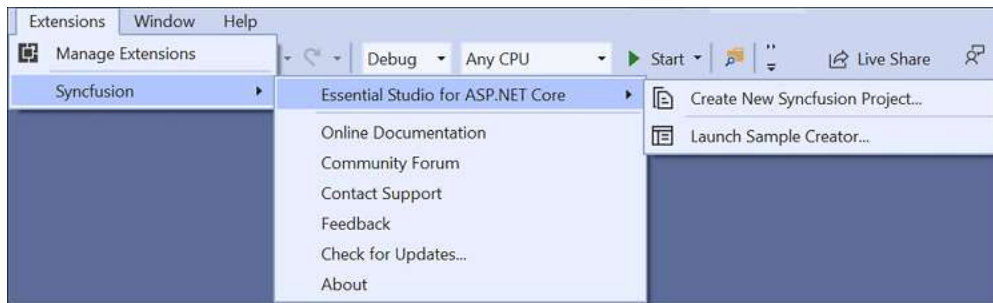
The Syncfusion Angular project templates are available from v17.1.0.47.

The following steps help you to create the Syncfusion Angular application through the Visual Studio:

1. Open the Visual Studio 2017 or later.
2. To create a Syncfusion Angular project, follow either one of the options below:

Option 1:

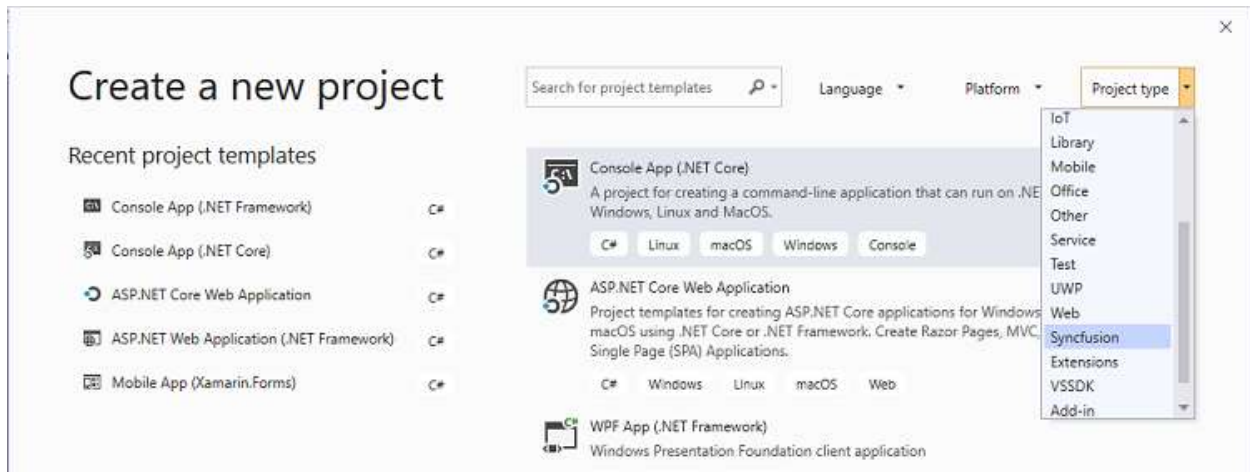
Choose the **Extension->Syncfusion-> Essential Studio for ASP.NET Core -> Create New Syncfusion Project...** in Visual Studio menu.



In Visual Studio 2017, you can see the **Syncfusion** menu directly in the Visual Studio menu.

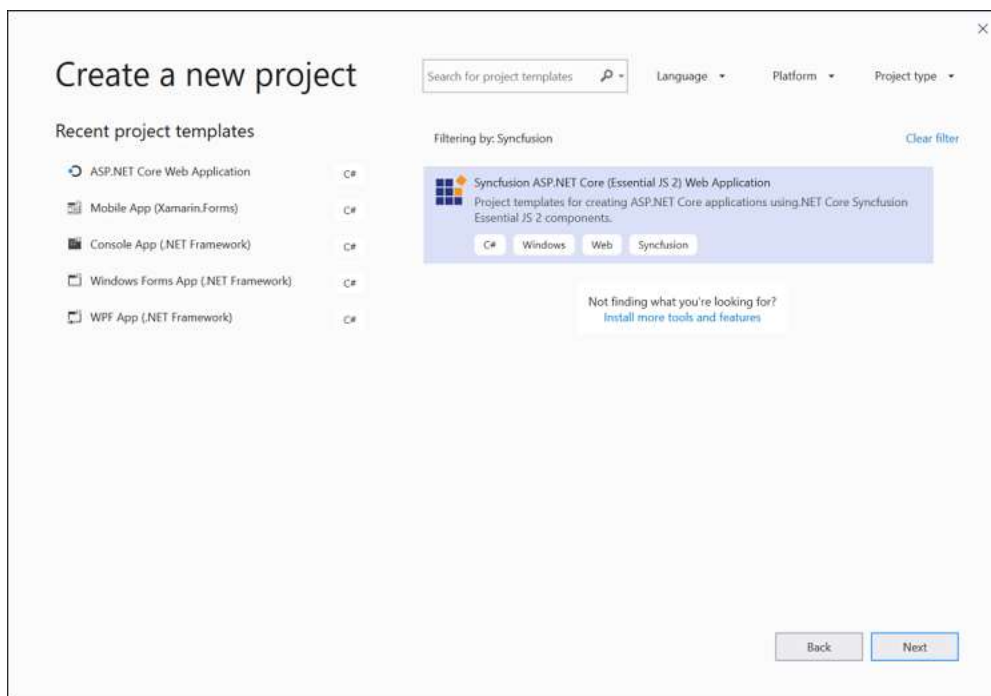
Option 2:

Choose **File > New > Project in Visual Studio**. The Create a new project dialog opens. You can get the **Syncfusion** provided templates by filtering the Project type with Syncfusion or use the **Search option** with the key word of Syncfusion.

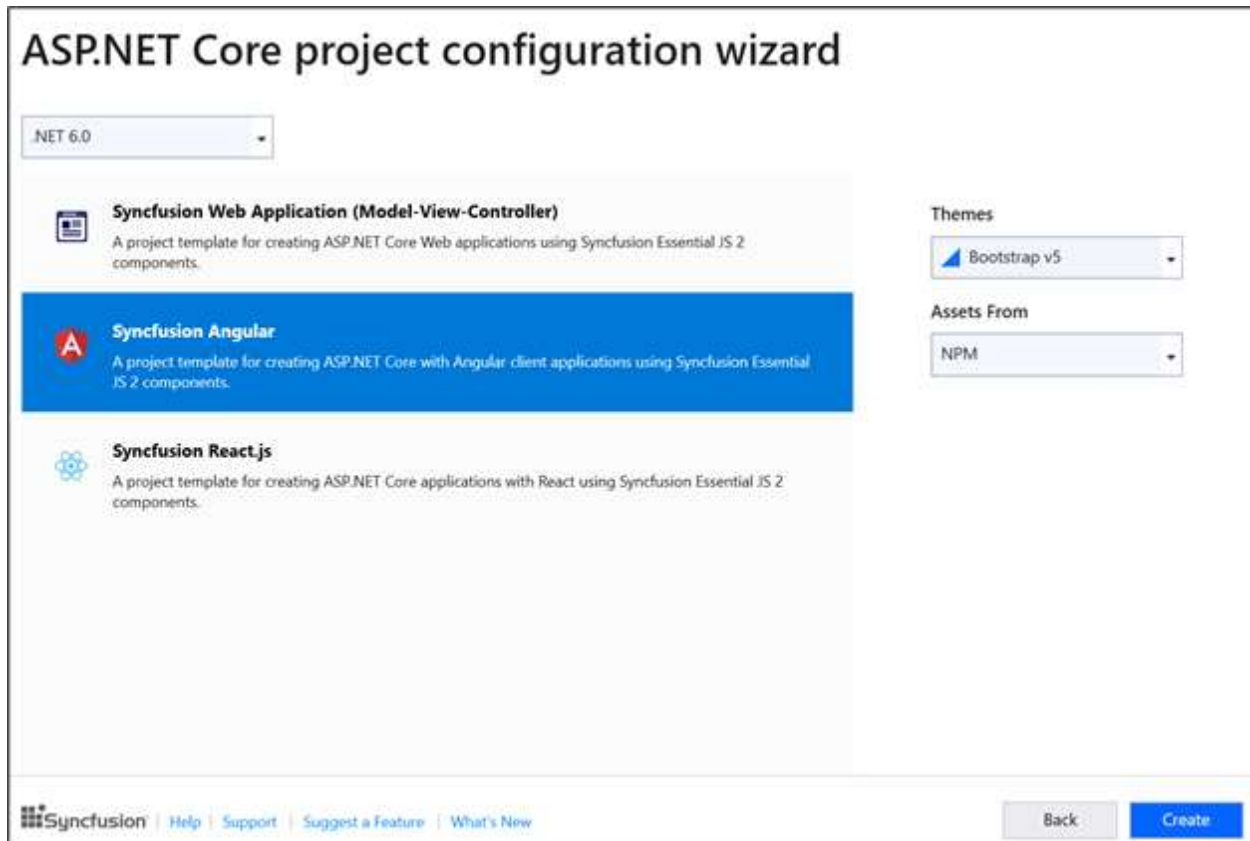


In Visual Studio 2017, choose **File > New > Project** and navigate to **Syncfusion > .NET Core > Syncfusion ASP.NET Core Web Application** in Visual Studio.

3. Select the Syncfusion ASP.NET Core Web Application and choose the Next button.



4. Name the Project, choose the destination location and then click Create button. The Syncfusion ASP.NET Core project configuration wizard appears.



Choose the Syncfusion Angular template and choose required theme and asset.

5. Click the Create button, the Syncfusion Angular application has been created.
6. The created Syncfusion Angular application configured with Syncfusion.
7. The required Syncfusion Angular NPM packages, scripts, and selected styles have been configured with the created Angular application.

Convert Project

Syncfusion Angular conversion is a Visual Studio add-in that converts an existing Angular application into a Syncfusion Angular Web application.

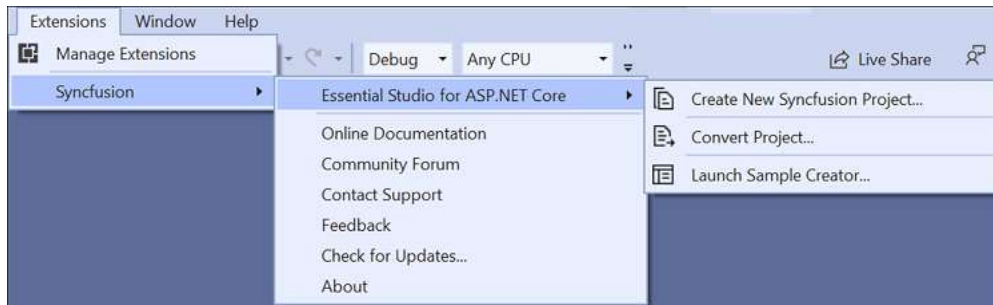
The Syncfusion Angular Project conversion are available from v17.3.0.9.

The steps below help you to convert the Angular application to Syncfusion Angular application through the Visual Studio:

1. Open your existing Angular application or create a new Angular application
2. To open the Syncfusion Project Conversion Wizard, follow either one of the options below:

Option 1:

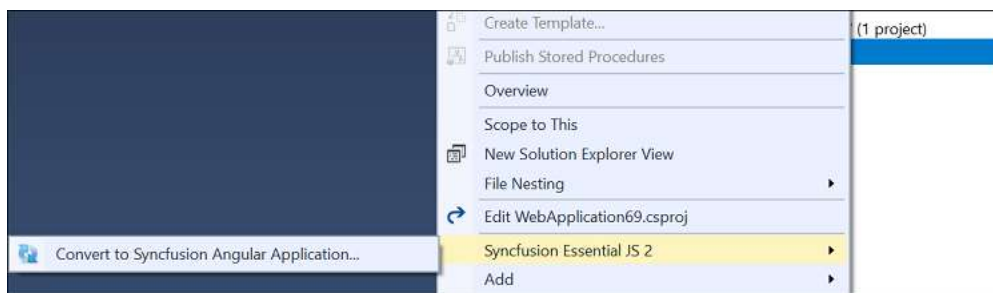
Choose **Extensions-> Syncfusion-> Essential Studio for ASP.NET Core ->Convert Project...** in **Visual Studio** menu.



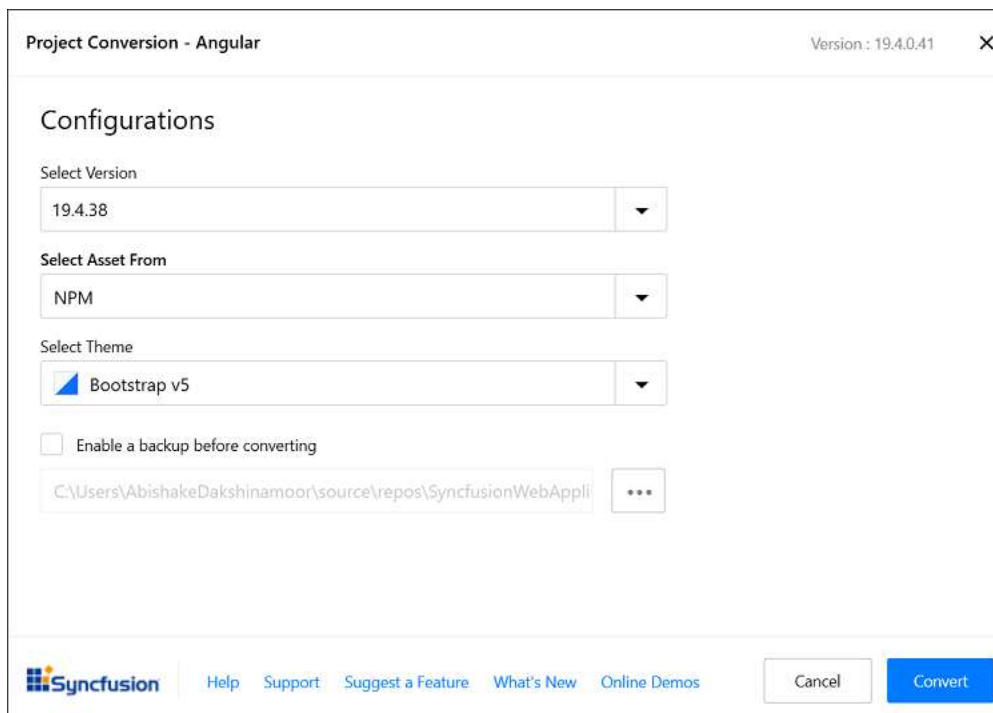
In Visual Studio 2017, you can see the **Syncfusion** menu directly in the Visual Studio menu

Option 2:

Right-click on the **Angular Application** from the Solution Explorer and select the **Syncfusion Web** and choose the **Convert to Syncfusion Angular application...**



3. The **Syncfusion Angular Project Conversion** window will appear. You can choose the required version of Syncfusion Angular version, Assets from, and Themes to convert the application.



The Syncfusion Angular versions are loaded from published Syncfusion Angular NPM package versions and it requires the internet connectivity.

The following configurations are used in the Project conversion wizard.

Assets From: Load the Syncfusion Essential JS 2 assets to Angular Project, from either NPM, CDN, or Installed Location.

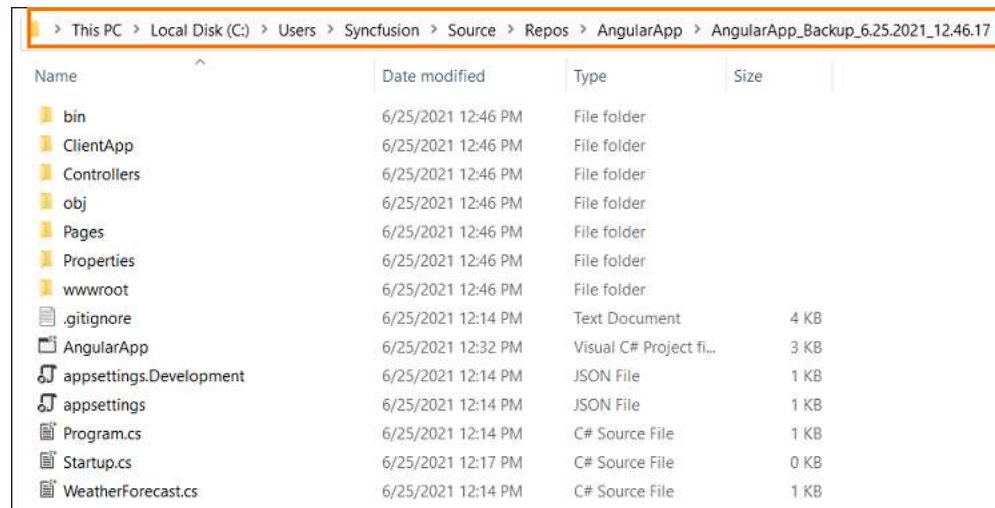
Installed location option will be available only when the Syncfusion Essential JavaScript 2 setup has been installed.

Choose the Theme: Choose the required theme.

4. Check the **“Enable a backup before converting”** checkbox if you want to take the project backup and choose the location.
5. Once the conversion process completed, will get the success message window.



if you enabled project backup before converting, the old project was saved in the specified backup path location, as shown below once the conversion process completed.



6. The required Syncfusion Angular NPM packages with selected version, scripts and selected style are added in the application.

Upgrade Project in Angular Visual studio integration

The Syncfusion Angular migration add-in for Visual Studio allows you to migrate an existing Syncfusion Angular application from one version of Essential Studio version to another version. This reduces the amount of manual work required when migrating the Syncfusion version.

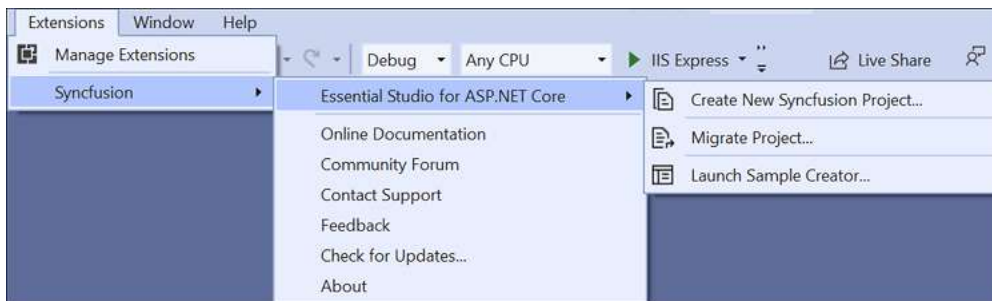
The Syncfusion Angular Project migration are available from v17.3.0.9.

The steps below help you to upgrade the Syncfusion version in **Syncfusion Angular Application** through the **Visual Studio**:

1. Open the Syncfusion Angular application which uses the Syncfusion component.
2. To open Migration Wizard, follow either one of the options below:

Option 1:

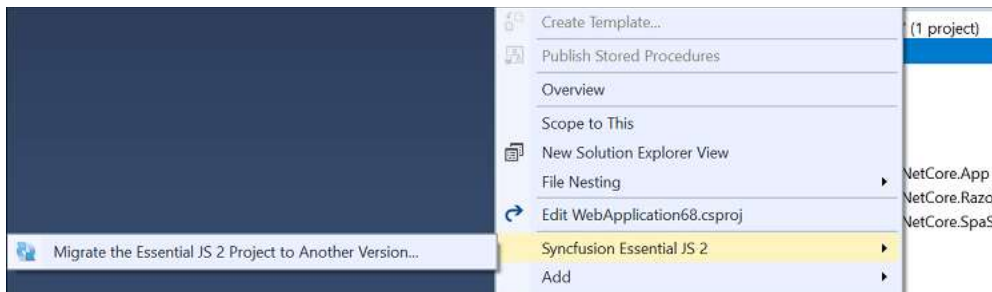
Choose **Extensions-> Syncfusion-> Essential Studio for ASP.NET Core ->Migrate Project...** in **Visual Studio** menu.



In Visual Studio 2017, you can see the **Syncfusion** menu directly in the Visual Studio menu

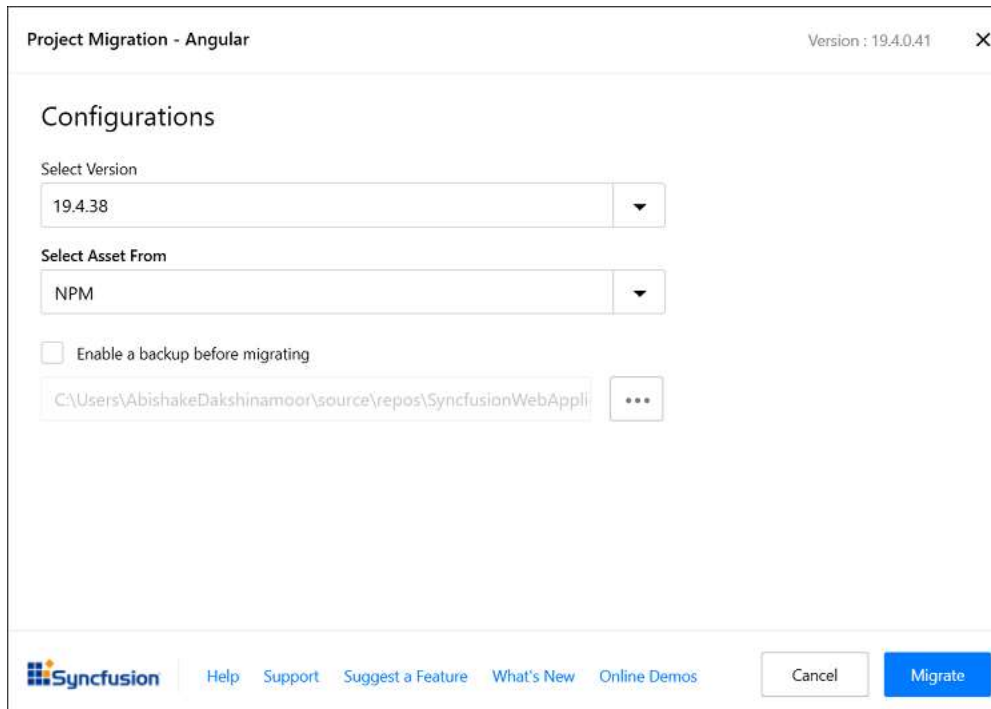
Option 2:

Right-click on the **Application** from the **Solution Explorer** and select the **Syncfusion Web** and choose the **Migrate the Syncfusion ASP.NET Core Project to Another version...**



3. The Syncfusion Project Migration window will appear. You can choose the required version of Syncfusion ANGULAR to migrate.

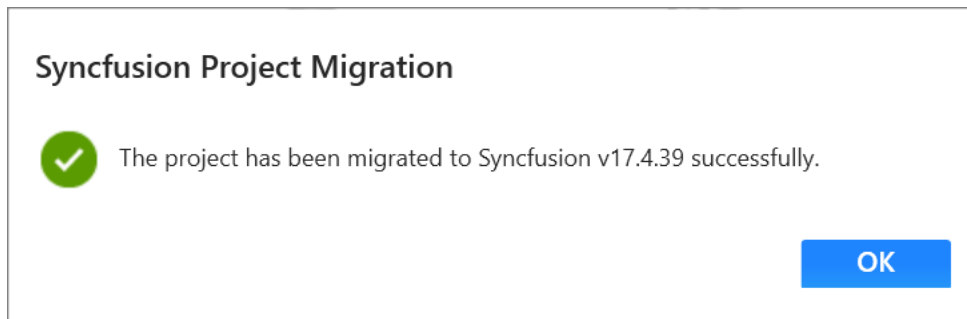
The Syncfusion Angular versions are loaded from published Syncfusion angular NPM packages and it requires the internet connectivity.



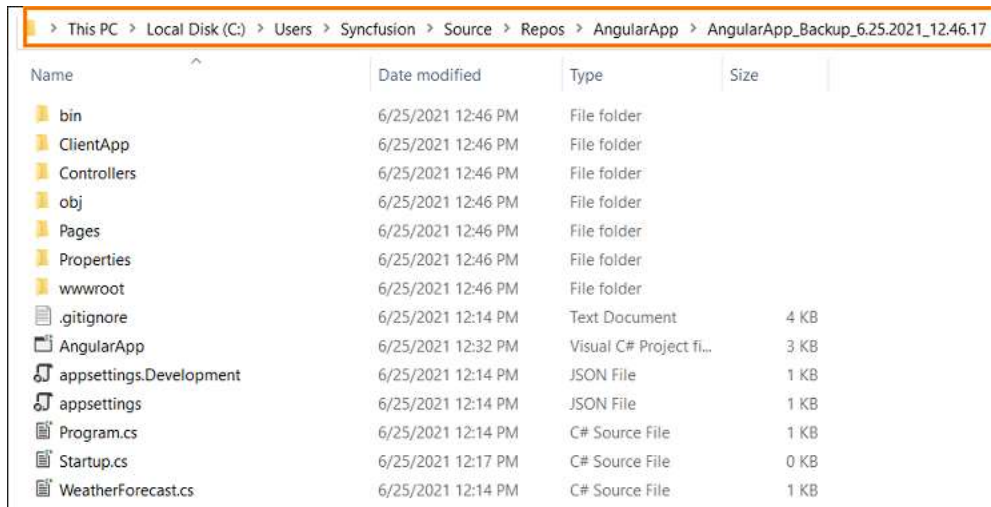
Assets From: Load the Syncfusion Essential JS 2 assets to Angular Project, from either NPM, CDN, or Installed Location.

Installed location option will be available only when the Syncfusion Essential JavaScript 2 setup has been installed.

4. Check the **“Enable a backup before migrating”** checkbox if you want to take the project backup and choose the location.
5. Once the migration process completed, will get the success message window.



if you enabled project backup before migrating, the old project was saved in the specified backup path location, as shown below once the migration process completed



6. The Syncfusion Angular NPM packages, and styles are updated to the selected version in the project.

3D Chart

Getting started with Angular 3D Chart component

This section explains you the steps required to create a simple **Angular 3D Chart** and demonstrate the basic usage of the 3D Chart component in an Angular environment.

Setup angular environment

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

To install Angular CLI use the following command.

```
`bash
```

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

```
,
```

Create an Angular application

Start a new Angular application using below Angular CLI command.

```
`bash
```

```
ng new my-app
```

```
cd my-app
```

```
,
```

Installing Syncfusion 3D Chart package

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

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

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

Ivy library distribution package

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

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

```
`bash
```

```
npm install @syncfusion/ej2-angular-charts --save
```

```
,
```

Angular compatibility compiled package(ngcc)

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

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

```
`bash
```

```
npm install @syncfusion/ej2-angular-charts@ngcc --save
```

```
,
```

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

```
`bash
```

```
@syncfusion/ej2-angular-charts:"20.2.38-ngcc"
```

```
,
```

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

Registering 3D Chart module

Import 3D Chart module into Angular application(`app.module.ts`) from the package `@syncfusion/ej2-angular-charts` [`src/app/app.module.ts`].

```
`typescript
```

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

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

```
// import the ChartModule for the Chart component
```

```
import { Chart3DModule } from '@syncfusion/ej2-angular-charts';
```

```
import { AppComponent } from './app.component';
```

```
@NgModule({
```

```
//declaration of ChartModule into NgModule
```

```
imports: [ BrowserModule, Chart3DModule ],
```

```
declarations: [ AppComponent ],
```

```
bootstrap: [ AppComponent ]
```

```

})
export class AppModule { }

```

- Modify the template in `app.component.ts` file to render the `ej2-angular-charts` component

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

```

`javascript
import { Component, ViewEncapsulation } from '@angular/core';
@Component({
  selector: 'app-container',
  // specifies the template string for the 3D Charts component
  template: <ejs-chart3d id='chart-container'></ejs-chart3d>,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent { }

```

<!-- markdownlint-disable MD033 -->

Now use the `<code>app-container</code>`

 in the index.html instead of default one.

```

<app-container></app-container>

```

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

```

npm start

```

The below example shows a basic 3D Charts.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Chart3DModule } from '@syncfusion/ej2-angular-charts'
import { Component } from '@angular/core';
@Component({
  imports: [
    Chart3DModule
  ],
  standalone: true,

```

```

    selector: 'app-container',
    // specifies the template string for the 3D Chart component
    template: `<ejs-chart3d id='chart-container'></ejs-chart3d>`
  })
  export class AppComponent {
  }

```

MAIN.TS

```

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

```

Module injection

3D Chart component are segregated into individual feature-wise modules. In order to use a particular feature, you need to inject its feature service in the AppModule. In the current application, we are going to modify the above basic 3D Chart to visualize sales data for a particular year. For this application we are going to use column series, tooltip, data label, category axis and legend feature of the 3D Chart. Please find relevant feature service name and description as follows. *ColumnSeries3DService* - Inject this provider to use column series. *Legend3DService* - Inject this provider to use legend feature. *Tooltip3DService* - Inject this provider to use tooltip feature. *DataLabel3DService* - Inject this provider to use datalabel feature. * *Category3DService* - Inject this provider to use category feature.

These modules should be injected to the provider section as follows,

`javascript

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import { Chart3DComponent } from '@syncfusion/ej2-angular-charts';
import { Category3DService, Legend3DService, Tooltip3DService } from '@syncfusion/ej2-angular-charts';
import { DataLabel3DService } from '@syncfusion/ej2-angular-charts';
@NgModule({
  imports: [
    BrowserModule,
  ],
  declarations: [AppComponent, Chart3DComponent],
  bootstrap: [AppComponent],
  providers: [ Category3DService, Legend3DService, Tooltip3DService, DataLabel3DService ]
})
`

```

Populate chart with data

This section explains how to plot below JSON data to the 3D Chart.

```

`javascript
export class AppComponent implements OnInit {
  public chartData: Object[];
  ngOnInit(): void {
    // Data for chart series
    this.chartData = [
      { x: 'Tesla', y: 137429 },
      { x: 'Aion', y: 80308 },
      { x: 'Wuling', y: 76418 },
      { x: 'Changan', y: 52849 },
      { x: 'Geely', y: 47234 },
      { x: 'Nio', y: 31041 },
      { x: 'Neta', y: 22449 },
      { x: 'BMW', y: 18733 }
    ];
  }
}
`

```

Add a series object to the 3D Chart by using [series](#) property. Now map the field names **x** and **y** in the JSON data to the [xName](#) and [yName](#) properties of the series, then set the JSON data to [dataSource](#) property.

Since the JSON contains category data, set the [valueType](#) for horizontal axis to **Category**.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts'
import { Component, OnInit } from '@angular/core';
@Component({
  imports: [
    Chart3DAllModule
  ],
  providers: [Chart3DAllModule]
  standalone: true,
  selector: 'app-container',
  template: `<ejs-chart3d style='display:block;' align='center'
[primaryXAxis]='primaryXAxis' [primaryYAxis]='primaryYAxis'
rotation=7 tilt=10 depth=100 [enableRotation]='enableRotation'>
<e-chart3d-series-collection>

```

```

    <e-chart3d-series [dataSource]='dataSource' type='Column' xName='x'
    yName='y'>
    </e-chart3d-series>
  </e-chart3d-series-collection>
</ejs-chart3d>`
})
export class AppComponent implements OnInit {
  public dataSource?: Object[];
  public primaryXAxis?: Object;
  public primaryYAxis?: Object;
  public enableRotation?: boolean;
  ngOnInit(): void {
    this.dataSource = [
      { x: 'Tesla', y: 137429 }, { x: 'Aion', y: 80308 },
      { x: 'Wuling', y: 76418 }, { x: 'Changan', y: 52849 },
      { x: 'Geely', y: 47234 }, { x: 'Nio', y: 31041 },
      { x: 'Neta', y: 22449 }, { x: 'BMW', y: 18733 }];
    this.primaryXAxis = {
      valueType: 'Category',
      labelRotation: -45,
      labelPlacement: 'BetweenTicks'
    };
    this.primaryYAxis = {
      maximum: 150000,
      interval: 50000
    };
    this.enableRotation = 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 3D Chart title

You can add a title using [title](#) property to the 3D Chart to provide quick information to the user about the data plotted in the 3D Chart.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts'
import { Component } from '@angular/core';
@Component({
  imports: [
    Chart3DAllModule
  ],
  providers: [Chart3DAllModule]
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the Chart component

```

```

    template: `<ejs-chart3d style='display:block;' align='center'
    [title]='title' [primaryXAxis]='primaryXAxis' [primaryYAxis]='primaryYAxis'
    rotation=7 tilt=10 depth=100 [enableRotation]='enableRotation'>
      <e-chart3d-series-collection>
        <e-chart3d-series [dataSource]='dataSource' type='Column' xName='x'
yName='y'>
          </e-chart3d-series>
        </e-chart3d-series-collection>
      </ejs-chart3d>`
  })
  export class AppComponent {
    public dataSource?: Object[];
    public primaryXAxis?: Object;
    public primaryYAxis?: Object;
    public enableRotation?: boolean;
    public title?: string;
    ngOnInit(): void {
      this.dataSource = [
        { x: 'Tesla', y: 137429 }, { x: 'Aion', y: 80308 },
        { x: 'Wuling', y: 76418 }, { x: 'Changan', y: 52849 },
        { x: 'Geely', y: 47234 }, { x: 'Nio', y: 31041 },
        { x: 'Neta', y: 22449 }, { x: 'BMW', y: 18733 }];
      this.primaryXAxis = {
        valueType: 'Category',
        labelRotation: -45,
        labelPlacement: 'BetweenTicks'
      };
      this.primaryYAxis = {
        maximum: 150000,
        interval: 50000
      };
      this.enableRotation = true;
      this.title = 'Top Selling Electric Cars in China';
    }
  }
}

```

MAIN.TS

```

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

```

Enable legend

You can use legend for the 3D Chart by setting the [visible](#) property to true in [LegendSettings](#) object and by injecting the [Legend3DService](#) into the [@NgModule.providers](#).

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts';
import { Component, OnInit } from '@angular/core';
@Component({
  imports: [

```

```

    Chart3DAllModule
  ],
  providers: [Chart3DAllModule]
  standalone: true,
  selector: 'app-container',
  template: `<ejs-chart3d style='display:block;' align='center'
[primaryXAxis]='primaryXAxis' [primaryYAxis]='primaryYAxis'
  rotation=7 tilt=10 depth=100 [legendSettings]="legendSettings"
[enableRotation]='enableRotation' [title]='title'>
    <e-chart3d-series-collection>
      <e-chart3d-series [dataSource]='dataSource' type='Column' xName='x'
yName='y' name='Sales'>
    </e-chart3d-series>
    </e-chart3d-series-collection>
  </ejs-chart3d>`
  })
  export class AppComponent implements OnInit {
    public dataSource?: Object[];
    public primaryXAxis?: Object;
    public primaryYAxis?: Object;
    public enableRotation?: boolean;
    public title?: string;
    public legendSettings?: Object;
    ngOnInit(): void {
      this.dataSource = [
        { x: 'Tesla', y: 137429 }, { x: 'Aion', y: 80308 },
        { x: 'Wuling', y: 76418 }, { x: 'Changan', y: 52849 },
        { x: 'Geely', y: 47234 }, { x: 'Nio', y: 31041 },
        { x: 'Neta', y: 22449 }, { x: 'BMW', y: 18733 }];
      this.primaryXAxis = {
        valueType: 'Category',
        labelRotation: -45,
        labelPlacement: 'BetweenTicks'
      };
      this.primaryYAxis = {
        maximum: 150000,
        interval: 50000
      };
      this.enableRotation = true;
      this.title = 'Top Selling Electric Cars in China';
      this.legendSettings = { visible: true };
    }
  }
}

```

MAIN.TS

```

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

```

Add data label

You can add data labels to improve the readability of the 3D Chart.

This can be achieved by setting the [visible](#) property to true in the [dataLabel](#) object and by injecting `DataLabel3DService` into the `@NgModule.providers`.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts'
import { Component, OnInit } from '@angular/core';
@Component({
  imports: [
    Chart3DAllModule
  ],
  providers: [Chart3DAllModule]
  standalone: true,
  selector: 'app-container',
  template: `<ejs-chart3d style='display:block;' align='center'
[primaryXAxis]='primaryXAxis' [primaryYAxis]='primaryYAxis'
rotation=7 tilt=10 depth=100 [enableRotation]='enableRotation'
[title]='title'>
    <e-chart3d-series-collection>
      <e-chart3d-series [dataSource]='dataSource' [dataLabel]="datalabel"
type='Column' xName='x' yName='y' name='Sales'>
        </e-chart3d-series>
      </e-chart3d-series-collection>
    </ejs-chart3d>`
})
export class AppComponent implements OnInit {
  public dataSource?: Object[];
  public primaryXAxis?: Object;
  public primaryYAxis?: Object;
  public enableRotation?: boolean;
  public title?: string;
  public legendSettings?: Object;
  public datalabel?: Object;
  ngOnInit(): void {
    this.dataSource = [
      { x: 'Tesla', y: 137429 }, { x: 'Aion', y: 80308 },
      { x: 'Wuling', y: 76418 }, { x: 'Changan', y: 52849 },
      { x: 'Geely', y: 47234 }, { x: 'Nio', y: 31041 },
      { x: 'Neta', y: 22449 }, { x: 'BMW', y: 18733 }];
    this.primaryXAxis = {
      valueType: 'Category',
      labelRotation: -45,
      labelPlacement: 'BetweenTicks'
    };
    this.primaryYAxis = {
      maximum: 150000,
      interval: 50000
    };
    this.enableRotation = true;
    this.title = 'Top Selling Electric Cars in China';
    this.legendSettings = { visible: true };
    this.datalabel = {
      visible: true
    };
  }
}
```



```
}

```

MAIN.TS

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

Enable tooltip

The tooltip is useful when you cannot display information by using the data labels due to space constraints. You can enable tooltip by setting the [enable](#) property as true in [tooltip](#) object and by injecting `Tooltip3DService` into the `@NgModule.providers`.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts'
import { Component, OnInit } from '@angular/core';
@Component({
  imports: [
    Chart3DAllModule
  ],
  providers: [Chart3DAllModule]
  standalone: true,
  selector: 'app-container',
  template: `<ejs-chart3d style='display:block;' align='center'
[tooltip]="tooltip" [primaryXAxis]='primaryXAxis'
[primaryYAxis]='primaryYAxis'
rotation=7 tilt=10 depth=100 [enableRotation]='enableRotation'>
<e-chart3d-series-collection>
  <e-chart3d-series [dataSource]='dataSource' [dataLabel]='datalabel'
type='Column' xName='x' yName='y' name='Sales'>
    </e-chart3d-series>
  </e-chart3d-series-collection>
</ejs-chart3d>`
})
export class AppComponent implements OnInit {
  public dataSource?: Object[];
  public primaryXAxis?: Object;
  public primaryYAxis?: Object;
  public enableRotation?: boolean;
  public title?: string;
  public legendSettings?: Object;
  public datalabel?: Object;
  public tooltip?: Object;
  ngOnInit(): void {
    this.dataSource = [
      { x: 'Tesla', y: 137429 }, { x: 'Aion', y: 80308 },
      { x: 'Wuling', y: 76418 }, { x: 'Changan', y: 52849 },
      { x: 'Geely', y: 47234 }, { x: 'Nio', y: 31041 },
      { x: 'Neta', y: 22449 }, { x: 'BMW', y: 18733 }];
    this.primaryXAxis = {
      valueType: 'Category',
```

```

        labelRotation: -45,
        labelPlacement: 'BetweenTicks'
    };
    this.primaryYAxis = {
        maximum: 150000,
        interval: 50000
    };
    this.enableRotation = true;
    this.title = 'Top Selling Electric Cars in China';
    this.legendSettings = { visible: true };
    this.datalabel = {
        visible: true
    };
    this.tooltip = { enable: true };
}
}

```

MAIN.TS

```

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

```

You can also explore our [Angular 3D Charts example](#) that shows various 3D Chart types and how to represent time-dependent data, showing trends in data at equal intervals.

<!-- markdownlint-disable MD036 -->

Working with data in Angular 3D Chart control

Local data

A simple JSON data can be bound to the 3D chart using [dataSource](#) property in series. Now map the fields in JSON to [xName](#) and [yName](#) properties.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts'
import { Component } from '@angular/core';
@Component({
  imports: [
    Chart3DAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the Chart component
  template: `<ejs-chart3d style='display:block;' align='center'
[primaryXAxis]='primaryXAxis'
rotation=7 tilt=10 depth=100 [enableRotation]='enableRotation'>
<e-chart3d-series-collection>
  <e-chart3d-series [dataSource]='dataSource' type='Column'
xName='month' yName='sales'>
  </e-chart3d-series>
</e-chart3d-series-collection>

```

```

</ejs-chart3d>`
  })
  export class AppComponent {
    public dataSource?: Object[];
    public primaryXAxis?: Object;
    public enableRotation?: boolean;
    ngOnInit(): void {
      this.dataSource = [
        { month: 'Jan', sales: 35 }, { month: 'Feb', sales: 28 },
        { month: 'Mar', sales: 34 }, { month: 'Apr', sales: 32 },
        { month: 'May', sales: 40 }, { month: 'Jun', sales: 32 },
        { month: 'Jul', sales: 35 }, { month: 'Aug', sales: 55 },
        { month: 'Sep', sales: 38 }, { month: 'Oct', sales: 30 },
        { month: 'Nov', sales: 25 }, { month: 'Dec', sales: 32 }
      ];
      this.primaryXAxis = {
        valueType: 'Category',
        labelRotation: -45,
        labelPlacement: 'BetweenTicks'
      };
      this.enableRotation = true;
    }
  }
}

```

MAIN.TS

```

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

```

Remote data

The remote data can be bound to the 3D chart using the [DataManager](#). The [DataManager](#) requires minimal information like web service URL, adaptor and cross domain to interact with service endpoint properly. Assign the instance of the [DataManager](#) to the [dataSource](#) property in series and map the fields of data to [xName](#) and [yName](#) properties. You can also use the [query](#) property of the series to filter the data.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts'
import { DataManager, Query } from '@syncfusion/ej2-data';
import { Component } from '@angular/core';
@Component({
  imports: [
    Chart3DAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the Chart component
  template: `<ejs-chart3d style='display:block;' align='center'
[primaryXAxis]='primaryXAxis'
rotation=7 tilt=10 depth=100 [enableRotation]='enableRotation'>

```

```

    <e-chart3d-series-collection>
      <e-chart3d-series [dataSource]='dataManager' type='Column'
xName='CustomerID' yName='Freight' [query]="query">
    </e-chart3d-series>
  </e-chart3d-series-collection>
</ejs-chart3d>`
  })
  export class AppComponent {
    public dataSource?: Object[];
    public primaryXAxis?: Object;
    public enableRotation?: boolean;
    public dataManager: DataManager = new DataManager({
      url: 'https://ej2services.syncfusion.com/production/web-
services/api/Orders'
    });
    public query: Query = new Query().take(5).where('Estimate',
'lessThan', 3, false);
    ngOnInit(): void {
      this.primaryXAxis = {
        valueType: 'Category',
        labelRotation: -45,
        labelPlacement: 'BetweenTicks'
      };
      this.enableRotation = true;
    }
  }
}

```

MAIN.TS

```

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

```

Binding data using ODataAdaptor

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

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts'
import { DataManager, Query, ODataAdaptor } from '@syncfusion/ej2-data';
import { Component } from '@angular/core';
@Component({
  imports: [
    Chart3DAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the Chart component
  template: `<ejs-chart3d style='display:block;' align='center'
[primaryXAxis]='primaryXAxis'

```

```

rotation=7 tilt=10 depth=100 [enableRotation]='enableRotation'>
<e-chart3d-series-collection>
  <e-chart3d-series [dataSource]='dataManager' type='Column'
xName='CustomerID' yName='Freight' [query]="query">
  </e-chart3d-series>
</e-chart3d-series-collection>
</ejs-chart3d>`
}))
export class AppComponent {
  public dataSource?: Object[];
  public primaryXAxis?: Object;
  public enableRotation?: boolean;
  public dataManager: DataManager = new DataManager({
    url: 'https://ej2services.syncfusion.com/production/web-
services/api/Orders',
    adaptor: new ODataAdaptor()
  });
  public query: Query = new Query();
  ngOnInit(): void {
    this.primaryXAxis = {
      valueType: 'Category',
      labelRotation: -45,
      labelPlacement: 'BetweenTicks'
    };
    this.enableRotation = true;
  }
}

```

MAIN.TS

```

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

```

Empty points

The data points that uses the **null** or **undefined** as value are considered as empty points. The empty data points are ignored and is not plotted in the chart. When the data is provided by using the points property, by using [emptyPointSettings](#) property in series, the empty can be customized. The default [mode](#) of the empty point is **Gap**.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts'
import { Component } from '@angular/core';
@Component({
  imports: [
    Chart3DAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the Chart component

```

```

    template: `<ejs-chart3d style='display:block;' align='center'
[primaryXAxis]='primaryXAxis'
    rotation=7 tilt=10 depth=100 [enableRotation]='enableRotation'>
    <e-chart3d-series-collection>
        <e-chart3d-series [dataSource]='dataSource' type='Column'
xName='month' yName='sales' [emptyPointSettings]="emptyPointSettings">
        </e-chart3d-series>
    </e-chart3d-series-collection>
</ejs-chart3d>`
    })
    export class AppComponent {
        public dataSource?: Object[];
        public primaryXAxis?: Object;
        public enableRotation?: boolean;
        public emptyPointSettings?: Object;
        ngOnInit(): void {
            this.dataSource = [
                { month: 'Jan', sales: 35 }, { month: 'Feb', sales: 28 },
                { month: 'Mar', sales: null }, { month: 'Apr', sales: 32 },
                { month: 'May', sales: 40 }, { month: 'Jun', sales: 32 },
                { month: 'Jul', sales: 35 }, { month: 'Aug', sales: undefined },
                { month: 'Sep', sales: 38 }, { month: 'Oct', sales: 30 },
                { month: 'Nov', sales: 25 }, { month: 'Dec', sales: 32 }
            ];
            this.primaryXAxis = {
                valueType: 'Category',
                labelRotation: -45,
                labelPlacement: 'BetweenTicks'
            };
            this.enableRotation = true;
            this.emptyPointSettings = {
                mode: 'Gap'
            };
        }
    }
}

```

MAIN.TS

```

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

```

Customizing empty point

The specific color for empty point can be set by the [fill](#) property in [emptyPointSettings](#).

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts'
import { DataManager, Query, ODataAdaptor } from '@syncfusion/ej2-data';
import { Component } from '@angular/core';
@Component({
    imports: [

```

```

    Chart3DAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the Chart component
  template: `<ejs-chart3d style='display:block;' align='center'
[primaryXAxis]='primaryXAxis'
  rotation=7 tilt=10 depth=100 [enableRotation]='enableRotation'>
    <e-chart3d-series-collection>
      <e-chart3d-series [dataSource]='dataSource' type='Column'
xName='month' yName='sales' [emptyPointSettings]="emptyPointSettings">
        </e-chart3d-series>
      </e-chart3d-series-collection>
    </ejs-chart3d>`
  })
  export class AppComponent {
    public dataSource?: Object[];
    public primaryXAxis?: Object;
    public enableRotation?: boolean;
    public emptyPointSettings: Object;
    ngOnInit(): void {
      this.dataSource = [
        { month: 'Jan', sales: 35 }, { month: 'Feb', sales: 28 },
        { month: 'Mar', sales: null }, { month: 'Apr', sales: 32 },
        { month: 'May', sales: 40 }, { month: 'Jun', sales: 32 },
        { month: 'Jul', sales: 35 }, { month: 'Aug', sales: undefined },
        { month: 'Sep', sales: 38 }, { month: 'Oct', sales: 30 },
        { month: 'Nov', sales: 25 }, { month: 'Dec', sales: 32 }
      ];
      this.primaryXAxis = {
        valueType: 'Category',
        labelRotation: -45,
        labelPlacement: 'BetweenTicks'
      };
      this.enableRotation = true;
      this.emptyPointSettings = {
        mode: 'Average',
        fill: 'green'
      };
    }
  }
}

```

MAIN.TS

```

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

```

Dimensions in Angular 3D Chart control

Size for container

The 3D chart can be rendered to its container size and it can be set via inline or CSS as demonstrated below.

```
`javascript
<div id='container'>

<div id='element' style="width:650px; height:350px;"></div>

</div>
`
```

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts'
import { Component } from '@angular/core';
@Component({
  imports: [
    Chart3DAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the Chart component
  template: `<ejs-chart3d width="650px" height="350px"
[primaryXAxis]='primaryXAxis' [primaryYAxis]="primaryYAxis"
rotation=7 tilt=10 depth=100 [enableRotation]='enableRotation'>
  <e-chart3d-series-collection>
    <e-chart3d-series [dataSource]='dataSource' type='Column'
xName='month' yName='sales'>
    </e-chart3d-series>
  </e-chart3d-series-collection>
</ejs-chart3d>`
})
export class AppComponent {
  public dataSource?: Object[];
  public primaryXAxis?: Object;
  public enableRotation?: boolean;
  public primaryYAxis?: Object;
  ngOnInit(): void {
    this.dataSource = [
      { month: 'Jan', sales: 35 }, { month: 'Feb', sales: 28 },
      { month: 'Mar', sales: 34 }, { month: 'Apr', sales: 32 },
      { month: 'May', sales: 40 }, { month: 'Jun', sales: 32 },
      { month: 'Jul', sales: 35 }, { month: 'Aug', sales: 55 },
      { month: 'Sep', sales: 38 }, { month: 'Oct', sales: 30 },
      { month: 'Nov', sales: 25 }, { month: 'Dec', sales: 32 }
    ];
    this.primaryXAxis = {
      valueType: "Category"
    };
    this.enableRotation = true;
    this.primaryYAxis = { };
  }
}
```

MAIN.TS

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



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

<!-- markdownlint-disable MD036 -->

Size for chart

<!-- markdownlint-disable MD036 -->

The size of the 3D chart can be set directly through [width](#) and [height](#) properties.

In Pixel

The size of the 3D chart can be set in pixel as demonstrated below.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts'
import { Component } from '@angular/core';
@Component({
  imports: [
    Chart3DAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the Chart component
  template: `<ejs-chart3d width="650" height="350"
[primaryXAxis]='primaryXAxis' [primaryYAxis]='primaryYAxis'
rotation=7 tilt=10 depth=100 [enableRotation]='enableRotation'>
<e-chart3d-series-collection>
  <e-chart3d-series [dataSource]='dataSource' type='Column'
xName='month' yName='sales'>
    </e-chart3d-series>
  </e-chart3d-series-collection>
</ejs-chart3d>`
})
export class AppComponent {
  public dataSource?: Object[];
  public primaryXAxis?: Object;
  public enableRotation?: boolean;
  public primaryYAxis?: Object;
  ngOnInit(): void {
    this.dataSource = [
      { month: 'Jan', sales: 35 }, { month: 'Feb', sales: 28 },
      { month: 'Mar', sales: 34 }, { month: 'Apr', sales: 32 },
      { month: 'May', sales: 40 }, { month: 'Jun', sales: 32 },
      { month: 'Jul', sales: 35 }, { month: 'Aug', sales: 55 },
      { month: 'Sep', sales: 38 }, { month: 'Oct', sales: 30 },
      { month: 'Nov', sales: 25 }, { month: 'Dec', sales: 32 }
    ];
    this.primaryXAxis = {
      valueType: "Category"
    };
    this.enableRotation = true;
    this.primaryYAxis = { };
  }
}
```

```
}
}
```

MAIN.TS

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

In Percentage

By setting the value in percentage, 3D chart gets its dimension with respect to its container. For example, when the height is **50%**, chart renders to half of the container height.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts'
import { Component } from '@angular/core';
@Component({
  imports: [
    Chart3DAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the Chart component
  template: `<ejs-chart3d width="80%" height="90%"
[primaryXAxis]='primaryXAxis' [primaryYAxis]='primaryYAxis'
rotation=7 tilt=10 depth=100 [enableRotation]='enableRotation'>
<e-chart3d-series-collection>
  <e-chart3d-series [dataSource]='dataSource' type='Column'
xName='month' yName='sales'>
    </e-chart3d-series>
  </e-chart3d-series-collection>
</ejs-chart3d>`
})
export class AppComponent {
  public dataSource?: Object[];
  public primaryXAxis?: Object;
  public enableRotation?: boolean;
  public primaryYAxis?: Object;
  ngOnInit(): void {
    this.dataSource = [
      { month: 'Jan', sales: 35 }, { month: 'Feb', sales: 28 },
      { month: 'Mar', sales: 34 }, { month: 'Apr', sales: 32 },
      { month: 'May', sales: 40 }, { month: 'Jun', sales: 32 },
      { month: 'Jul', sales: 35 }, { month: 'Aug', sales: 55 },
      { month: 'Sep', sales: 38 }, { month: 'Oct', sales: 30 },
      { month: 'Nov', sales: 25 }, { month: 'Dec', sales: 32 }
    ];
    this.primaryXAxis = {
      valueType: "Category"
    };
    this.enableRotation = true;
  }
}
```

```

        this.primaryYAxis = { };
    }
}

```

MAIN.TS

```

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

```

Note: When you do not specify the size, it takes 450px as the height and window size as its width.

Category axis in Angular 3D Chart control

The category axis is the horizontal axis of a 3D chart that shows text values rather than numerical values. Compared to the vertical axis, this axis has fewer labels. The following sample shows to render the 3D chart using category axis.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts';
import { Component } from '@angular/core';
@Component({
  imports: [
    Chart3DAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the Chart component
  template: `<ejs-chart3d style='display:block;' align='center'
[primaryXAxis]='primaryXAxis' [primaryYAxis]="primaryYAxis"
rotation=7 tilt=10 depth=100 [enableRotation]='enableRotation'>
<e-chart3d-series-collection>
  <e-chart3d-series [dataSource]='dataSource' type='Column'
xName='country' yName='gold'>
    </e-chart3d-series>
  </e-chart3d-series-collection>
</ejs-chart3d>`
})
export class AppComponent {
  public dataSource?: Object[];
  public primaryXAxis?: Object;
  public enableRotation?: boolean;
  public primaryYAxis?: Object;
  ngOnInit(): void {
    this.dataSource = [
      { country: "USA", gold: 50, silver: 70, bronze: 45 },
      { country: "China", gold: 40, silver: 60, bronze: 55 },
      { country: "Japan", gold: 70, silver: 60, bronze: 50 },
      { country: "Australia", gold: 60, silver: 56, bronze: 40 },
      { country: "France", gold: 50, silver: 45, bronze: 35 },
      { country: "Germany", gold: 40, silver: 30, bronze: 22 },
      { country: "Italy", gold: 40, silver: 35, bronze: 37 },
    ];
  }
}

```

```

        { country: "Sweden", gold: 30, silver: 25, bronze: 27 }
    ];
    this.primaryXAxis = {
        valueType: "Category",
        labelPlacement: 'OnTicks'
    };
    this.enableRotation = true;
    this.primaryYAxis = { };
}
}

```

MAIN.TS

```

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

```

Note: To use category axis, we need to inject `Category3DService` module into the `@NgModule.providers` and set the `valueType` of axis to `Category`.

Range

The range of the category axis can be customized using `minimum`, `maximum` and `interval` properties of the axis.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts'
import { Component } from '@angular/core';
@Component({
  imports: [
    Chart3DAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the Chart component
  template: `<ejs-chart3d style='display:block;' align='center'
[primaryXAxis]='primaryXAxis' [primaryYAxis]="primaryYAxis"
rotation=7 tilt=10 depth=100 [enableRotation]='enableRotation'>
<e-chart3d-series-collection>
  <e-chart3d-series [dataSource]='dataSource' type='Column'
xName='country' yName='gold'>
    </e-chart3d-series>
  </e-chart3d-series-collection>
</ejs-chart3d>`
})
export class AppComponent {
  public dataSource?: Object[];
  public primaryXAxis?: Object;
  public enableRotation?: boolean;
  public primaryYAxis?: Object;
  ngOnInit(): void {
    this.dataSource = [

```

```

        { country: "USA", gold: 50, silver: 70, bronze: 45 },
        { country: "China", gold: 40, silver: 60, bronze: 55 },
        { country: "Japan", gold: 70, silver: 60, bronze: 50 },
        { country: "Australia", gold: 60, silver: 56, bronze: 40 },
        { country: "France", gold: 50, silver: 45, bronze: 35 },
        { country: "Germany", gold: 40, silver: 30, bronze: 22 },
        { country: "Italy", gold: 40, silver: 35, bronze: 37 },
        { country: "Sweden", gold: 30, silver: 25, bronze: 27 }
    ];
    this.primaryXAxis = {
        valueType: "Category",
        interval: 2, minimum: 1, maximum: 5
    };
    this.enableRotation = true;
    this.primaryYAxis = { };
}
}

```

MAIN.TS

```

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

```

Indexed category axis

The category axis can also be rendered based on the index values of the data source. This can be achieved by defining the [isIndexed](#) property to **true** in the axis.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts'
import { Component } from '@angular/core';
@Component({
  imports: [
    Chart3DAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the Chart component
  template: `<ejs-chart3d style='display:block;' align='center'
    [primaryXAxis]='primaryXAxis' [primaryYAxis]="primaryYAxis"
    rotation=7 tilt=10 depth=100 [enableRotation]='enableRotation'>
    <e-chart3d-series-collection>
      <e-chart3d-series [dataSource]='dataSource' type='Column'
        xName='country' yName='gold'>
      </e-chart3d-series>
      <e-chart3d-series [dataSource]='dataSource' type='Column'
        xName='country' yName='silver'>
      </e-chart3d-series>
    </e-chart3d-series-collection>
  </ejs-chart3d>`
})

```

```

export class AppComponent {
  public dataSource?: Object[];
  public primaryXAxis?: Object;
  public enableRotation?: boolean;
  public primaryYAxis?: Object;
  ngOnInit(): void {
    this.dataSource = [
      { country: "USA", gold: 50, silver: 70, bronze: 45 },
      { country: "China", gold: 40, silver: 60, bronze: 55 },
      { country: "Japan", gold: 70, silver: 60, bronze: 50 },
      { country: "Australia", gold: 60, silver: 56, bronze: 40 },
      { country: "France", gold: 50, silver: 45, bronze: 35 },
      { country: "Germany", gold: 40, silver: 30, bronze: 22 },
      { country: "Italy", gold: 40, silver: 35, bronze: 37 },
      { country: "Sweden", gold: 30, silver: 25, bronze: 27 }
    ];
    this.primaryXAxis = {
      valueType: "Category",
      isIndexed: true
    };
    this.enableRotation = true;
    this.primaryYAxis = { };
  }
}

```

MAIN.TS

```

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

```

<!-- markdownlint-disable MD036 -->

Numeric axis in Angular 3D Chart control

The numeric axis can be used to represent the numeric values of data in 3D chart. By default, the **valueType** of an axis is **Double**.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts'
import { Component } from '@angular/core';
@Component({
  imports: [
    Chart3DAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the Chart component
  template: `<ejs-chart3d #chart style='display:block;' align='center'
[primaryXAxis]='primaryXAxis'
rotation=7 tilt=10 depth=100 [enableRotation]='enableRotation'>
<e-chart3d-series-collection>

```

```

    <e-chart3d-series [dataSource]='dataSource' type='Column' xName='x'
yName='y'>
    </e-chart3d-series>
  </e-chart3d-series-collection>
</ejs-chart3d>`
})
export class AppComponent {
  public dataSource?: Object[];
  public primaryXAxis?: Object;
  public enableRotation?: boolean;
  ngOnInit(): void {
    this.dataSource = [
      { x: 1, y: 7 }, { x: 2, y: 1 }, { x: 3, y: 1 }, { x: 4, y: 14 },
      { x: 5, y: 1 }, { x: 6, y: 10 },
      { x: 7, y: 8 }, { x: 8, y: 6 }, { x: 9, y: 10 }, { x: 10, y: 10 }, { x:
11, y: 16 }, { x: 12, y: 6 },
      { x: 13, y: 14 }, { x: 14, y: 7 }, { x: 15, y: 5 }, { x: 16, y: 2 }, { x:
17, y: 14 }, { x: 18, y: 7 },
      { x: 19, y: 7 }, { x: 20, y: 10 }
    ];
    this.enableRotation = true;
    this.primaryXAxis = {
      valueType: "Double"
    };
  }
}

```

MAIN.TS

```

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

```

Range

The range of an axis will be calculated automatically based on the provided data, and it can also be customized by using the [minimum](#), [maximum](#) and [interval](#) properties of the axis.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts'
import { Component } from '@angular/core';
@Component({
  imports: [
    Chart3DAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the Chart component
  template: `<ejs-chart3d #chart style='display:block;' align='center'
[primaryXAxis]='primaryXAxis'
rotation=7 tilt=10 depth=100 [enableRotation]='enableRotation'>
<e-chart3d-series-collection>

```

```

    <e-chart3d-series [dataSource]='dataSource' type='Column' xName='x'
yName='y'>
    </e-chart3d-series>
  </e-chart3d-series-collection>
</ejs-chart3d>`
  })
  export class AppComponent {
    public dataSource?: Object[];
    public primaryXAxis?: Object;
    public enableRotation?: boolean;
    ngOnInit(): void {
      this.dataSource = [
        { x: 1, y: 7 }, { x: 2, y: 1 }, { x: 3, y: 1 }, { x: 4, y: 14 },
        { x: 5, y: 1 }, { x: 6, y: 10 },
        { x: 7, y: 8 }, { x: 8, y: 6 }, { x: 9, y: 10 }, { x: 10, y: 10 },
        { x: 11, y: 16 }, { x: 12, y: 6 },
        { x: 13, y: 14 }, { x: 14, y: 7 }, { x: 15, y: 5 }, { x: 16, y: 2 },
        { x: 17, y: 14 }, { x: 18, y: 7 },
        { x: 19, y: 7 }, { x: 20, y: 10 }
      ];
      this.enableRotation = true;
      this.primaryXAxis = {
        valueType: 'Double',
        minimum: 1,
        maximum: 20,
        interval: 5
      }
    }
  }
}

```

MAIN.TS

```

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

```

Range padding

Padding can be applied to the minimum and maximum extremes of an axis range by using the [rangePadding](#) property. Numeric axis supports the following types of padding.

- None
- Round
- Additional
- Normal
- Auto

Numeric - None

When the [rangePadding](#) is set to **None**, minimum and maximum of the axis is based on the data.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'

```



```

import { BrowserModule } from '@angular/platform-browser'
import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts'
import { Component } from '@angular/core';
@Component({
  imports: [
    Chart3DAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the Chart component
  template: `<ejs-chart3d #chart style='display:block;' align='center'
[primaryXAxis]='primaryXAxis' [primaryYAxis]="primaryYAxis"
rotation=7 tilt=10 depth=100 [enableRotation]='enableRotation'>
<e-chart3d-series-collection>
  <e-chart3d-series [dataSource]='dataSource' type='Column' xName='x'
yName='y'>
  </e-chart3d-series>
</e-chart3d-series-collection>
</ejs-chart3d>`
})
export class AppComponent {
  public dataSource?: Object[];
  public primaryXAxis?: Object;
  public primaryYAxis?: Object;
  public enableRotation?: boolean;
  ngOnInit(): void {
    this.dataSource = [
      { x: 1, y: 7 }, { x: 2, y: 1 }, { x: 3, y: 1 }, { x: 4, y: 14 },
      { x: 5, y: 1 }, { x: 6, y: 10 },
      { x: 7, y: 8 }, { x: 8, y: 6 }, { x: 9, y: 10 }, { x: 10, y: 10 }, { x:
11, y: 16 }, { x: 12, y: 6 },
      { x: 13, y: 14 }, { x: 14, y: 7 }, { x: 15, y: 5 }, { x: 16, y: 2 }, { x:
17, y: 14 }, { x: 18, y: 7 },
      { x: 19, y: 7 }, { x: 20, y: 10 }
    ];
    this.enableRotation = true;
    this.primaryXAxis = {
      valueType: 'Double',
      minimum: 1,
      maximum: 20,
      interval: 5
    }
    this.primaryYAxis = {
      //RangePadding as none in Y Axis
      rangePadding: 'None'
    }
  }
}

```

MAIN.TS

```

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

```

Numeric - Round

When the [rangePadding](#) is set to **Round**, minimum and maximum will be rounded to the nearest possible value, which is divisible by interval. For example, when the [minimum](#) is **3.5** and the [interval](#) is **1**, then the minimum will be rounded to **3**.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts'
import { Component } from '@angular/core';
@Component({
  imports: [
    Chart3DAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the Chart component
  template: `<ejs-chart3d #chart style='display:block;' align='center'
[primaryXAxis]='primaryXAxis' [primaryYAxis]="primaryYAxis"
rotation=7 tilt=10 depth=100 [enableRotation]='enableRotation'>
  <e-chart3d-series-collection>
    <e-chart3d-series [dataSource]='dataSource' type='Column' xName='x'
yName='y'>
    </e-chart3d-series>
  </e-chart3d-series-collection>
</ejs-chart3d>`
})
export class AppComponent {
  public dataSource?: Object[];
  public primaryXAxis?: Object;
  public primaryYAxis?: Object;
  public enableRotation?: boolean;
  ngOnInit(): void {
    this.dataSource = [
      { x: 1, y: 7 }, { x: 2, y: 1 }, { x: 3, y: 1 }, { x: 4, y: 14 },
      { x: 5, y: 1 }, { x: 6, y: 10 },
      { x: 7, y: 8 }, { x: 8, y: 6 }, { x: 9, y: 10 }, { x: 10, y: 10 }, { x:
11, y: 16 }, { x: 12, y: 6 },
      { x: 13, y: 14 }, { x: 14, y: 7 }, { x: 15, y: 5 }, { x: 16, y: 2 }, { x:
17, y: 14 }, { x: 18, y: 7 },
      { x: 19, y: 7 }, { x: 20, y: 10 }
    ];
    this.enableRotation = true;
    this.primaryXAxis = {
      valueType: 'Double',
      minimum: 1,
      maximum: 20,
      interval: 5
    }
    this.primaryYAxis = {
      rangePadding: 'Round'
    }
  }
}
```

MAIN.TS

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

Numeric - Additional

When the [rangePadding](#) is set to **Additional**, interval of an axis will be added to the minimum and maximum of the axis.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts'
import { Component } from '@angular/core';
@Component({
  imports: [
    Chart3DAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the Chart component
  template: `<ejs-chart3d #chart style='display:block;' align='center'
[primaryXAxis]='primaryXAxis' [primaryYAxis]="primaryYAxis"
rotation=7 tilt=10 depth=100 [enableRotation]='enableRotation'>
<e-chart3d-series-collection>
  <e-chart3d-series [dataSource]='dataSource' type='Column' xName='x'
yName='y'>
  </e-chart3d-series>
</e-chart3d-series-collection>
</ejs-chart3d>`
})
export class AppComponent {
  public dataSource?: Object[];
  public primaryXAxis?: Object;
  public primaryYAxis?: Object;
  public enableRotation?: boolean;
  ngOnInit(): void {
    this.dataSource = [
      { x: 1, y: 7 }, { x: 2, y: 1 }, { x: 3, y: 1 }, { x: 4, y: 14 },
      { x: 5, y: 1 }, { x: 6, y: 10 },
      { x: 7, y: 8 }, { x: 8, y: 6 }, { x: 9, y: 10 }, { x: 10, y: 10 }, { x:
11, y: 16 }, { x: 12, y: 6 },
      { x: 13, y: 14 }, { x: 14, y: 7 }, { x: 15, y: 5 }, { x: 16, y: 2 }, { x:
17, y: 14 }, { x: 18, y: 7 },
      { x: 19, y: 7 }, { x: 20, y: 10 }
    ];
    this.enableRotation = true;
    this.primaryXAxis = {
      valueType: 'Double',
      minimum: 1,
      maximum: 20,
      interval: 5
    }
  }
}
```

```

        this.primaryYAxis = {
            rangePadding: 'Additional'
        }
    }
}

```

MAIN.TS

```

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

```

Numeric - Normal

When the [rangePadding](#) is set to **Normal**, padding is applied to the axis based on default range calculation.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts'
import { Component } from '@angular/core';
@Component({
  imports: [
    Chart3DAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the Chart component
  template: `<ejs-chart3d #chart style='display:block;' align='center'
[primaryXAxis]='primaryXAxis' [primaryYAxis]='primaryYAxis'
rotation=7 tilt=10 depth=100 [enableRotation]='enableRotation'>
<e-chart3d-series-collection>
    <e-chart3d-series [dataSource]='dataSource' type='Column' xName='x'
yName='y'>
    </e-chart3d-series>
  </e-chart3d-series-collection>
</ejs-chart3d>`
})
export class AppComponent {
  public dataSource?: Object[];
  public primaryXAxis?: Object;
  public primaryYAxis?: Object;
  public enableRotation?: boolean;
  ngOnInit(): void {
    this.dataSource = [
      { x: 1, y: 7 }, { x: 2, y: 1 }, { x: 3, y: 1 }, { x: 4, y: 14 },
      { x: 5, y: 1 }, { x: 6, y: 10 },
      { x: 7, y: 8 }, { x: 8, y: 6 }, { x: 9, y: 10 }, { x: 10, y: 10 }, { x:
11, y: 16 }, { x: 12, y: 6 },
      { x: 13, y: 14 }, { x: 14, y: 7 }, { x: 15, y: 5 }, { x: 16, y: 2 }, { x:
17, y: 14 }, { x: 18, y: 7 },
      { x: 19, y: 7 }, { x: 20, y: 10 }
    ];
  }
}

```

```

        this.enableRotation = true;
        this.primaryXAxis = {
            valueType: 'Double',
            minimum: 1,
            maximum: 20,
            interval: 5
        }
        this.primaryYAxis = {
            rangePadding: 'Normal'
        }
    }
}

```

MAIN.TS

```

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

```

Numeric - Auto

When the [rangePadding](#) is set to **Auto**, horizontal numeric axis takes **None** as padding calculation, while the vertical numeric axis takes **Normal** as padding calculation.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts';
import { Component } from '@angular/core';
@Component({
    imports: [
        Chart3DAllModule
    ],
    standalone: true,
    selector: 'app-container',
    // specifies the template string for the Chart component
    template: `<ejs-chart3d #chart style='display:block;' align='center'
[primaryXAxis]='primaryXAxis' [primaryYAxis]="primaryYAxis"
rotation=7 tilt=10 depth=100 [enableRotation]='enableRotation'>
<e-chart3d-series-collection>
    <e-chart3d-series [dataSource]='dataSource' type='Column' xName='x'
yName='y'>
    </e-chart3d-series>
</e-chart3d-series-collection>
</ejs-chart3d>`
})
export class AppComponent {
    public dataSource?: Object[];
    public primaryXAxis?: Object;
    public primaryYAxis?: Object;
    public enableRotation?: boolean;
    ngOnInit(): void {
        this.dataSource = [

```

```

        { x: 1, y: 7 }, { x: 2, y: 1 }, { x: 3, y: 1 }, { x: 4, y: 14 },
        { x: 5, y: 1 }, { x: 6, y: 10 },
        { x: 7, y: 8 }, { x: 8, y: 6 }, { x: 9, y: 10 }, { x: 10, y: 10 }, { x:
11, y: 16 }, { x: 12, y: 6 },
        { x: 13, y: 14 }, { x: 14, y: 7 }, { x: 15, y: 5 }, { x: 16, y: 2 }, { x:
17, y: 14 }, { x: 18, y: 7 },
        { x: 19, y: 7 }, { x: 20, y: 10 }
    ];
    this.enableRotation = true;
    this.primaryXAxis = {
        valueType: 'Double',
        // Set the rangePadding as auto in X Axis
        rangePadding: 'Auto'
    }
    this.primaryYAxis = {
        rangePadding: 'Auto'
    }
}
}

```

MAIN.TS

```

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

```

Label format

Numeric label format

Numeric labels can be formatted by using the [labelFormat](#) property. Also, it supports all globalize format.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts'
import { Component } from '@angular/core';
@Component({
  imports: [
    Chart3DAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the Chart component
  template: `<ejs-chart3d #chart style='display:block;' align='center'
[primaryXAxis]='primaryXAxis' [primaryYAxis]="primaryYAxis"
rotation=7 tilt=10 depth=100 [enableRotation]='enableRotation'>
<e-chart3d-series-collection>
    <e-chart3d-series [dataSource]='dataSource' type='Column' xName='x'
yName='y'>
    </e-chart3d-series>
  </e-chart3d-series-collection>
</ejs-chart3d>`
})

```

```

export class AppComponent {
  public dataSource?: Object[];
  public primaryXAxis?: Object;
  public primaryYAxis?: Object;
  public enableRotation?: boolean;
  ngOnInit(): void {
    this.dataSource = [
      { x: 1, y: 7 }, { x: 2, y: 1 }, { x: 3, y: 1 }, { x: 4, y: 14 },
      { x: 5, y: 1 }, { x: 6, y: 10 },
      { x: 7, y: 8 }, { x: 8, y: 6 }, { x: 9, y: 10 }, { x: 10, y: 10 }, { x:
11, y: 16 }, { x: 12, y: 6 },
      { x: 13, y: 14 }, { x: 14, y: 7 }, { x: 15, y: 5 }, { x: 16, y: 2 }, { x:
17, y: 14 }, { x: 18, y: 7 },
      { x: 19, y: 7 }, { x: 20, y: 10 }
    ];
    this.enableRotation = true;
    this.primaryXAxis = {
      valueType: 'Double'
    }
    this.primaryYAxis = {
      labelFormat: 'c'
    }
  }
}

```

MAIN.TS

```

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

```

The following table describes the result of applying some commonly used label formats on numeric values.

<!-- markdownlint-disable MD033 -->

Label Value	Label Format: Property Value	Result	Description
1000	n1	1000.0	The Number is rounded to 1 decimal place.
1000	n2	1000.00	The Number is rounded to 2 decimal place.
1000	n3	1000.000	The Number is rounded to 3 decimal place.
0.01	p1	1.0%	The Number is converted to percentage with 1 decimal place.
0.01	p2	1.00%	The Number is converted to percentage with 2 decimal place.
0.01	p3	1.000%	The Number is converted to percentage with 3 decimal place.

1000	c1	\$1000.0	The Currency symbol is appended to number and number is rounded to 1 decimal place.
1000	c2	\$1000.00	The Currency symbol is appended to number and number is rounded to 2 decimal place.

Grouping separator

To separate the y-axis labels to groups of thousands, set the [useGroupingSeparator](#) property to **true** in the 3D chart.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts'
import { Component } from '@angular/core';
@Component({
  imports: [
    Chart3DAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the Chart component
  template: `<ejs-chart3d #chart style='display:block;' align='center'
[primaryXAxis]='primaryXAxis' [useGroupingSeparator]=true
rotation=7 tilt=10 depth=100 [enableRotation]='enableRotation'>
  <e-chart3d-series-collection>
    <e-chart3d-series [dataSource]='dataSource' type='Column' xName='x'
yName='y'>
    </e-chart3d-series>
  </e-chart3d-series-collection>
</ejs-chart3d>`
})
export class AppComponent {
  public dataSource?: Object[];
  public primaryXAxis?: Object;
  public enableRotation?: boolean;
  ngOnInit(): void {
    this.dataSource = [
      { x: 10, y: 7000 }, { x: 20, y: 1000 }, { x: 30, y: 12000 }, { x:
40, y: 14000 }, { x: 50, y: 11000 }, { x: 60, y: 5000 },
      { x: 70, y: 7300 }, { x: 80, y: 9000 }, { x: 90, y: 12000 }, { x:
100, y: 14000 }, { x: 110, y: 11000 }, { x: 120, y: 5000 }
    ];
    this.enableRotation = true;
    this.primaryXAxis = {
      valueType: 'Double'
    }
  }
}
```

MAIN.TS

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



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

Custom label format

The axis supports custom label format using placeholder like **{value}°C**, in which the value represent the axis label e.g 20°C.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts'
import { Component } from '@angular/core';
@Component({
  imports: [
    Chart3DAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the Chart component
  template: `<ejs-chart3d #chart style='display:block;' align='center'
[primaryXAxis]='primaryXAxis' [primaryYAxis]="primaryYAxis"
rotation=7 tilt=10 depth=100 [enableRotation]='enableRotation'>
<e-chart3d-series-collection>
  <e-chart3d-series [dataSource]='dataSource' type='Column' xName='x'
yName='y'>
  </e-chart3d-series>
</e-chart3d-series-collection>
</ejs-chart3d>`
})
export class AppComponent {
  public dataSource?: Object[];
  public primaryXAxis?: Object;
  public primaryYAxis?: Object;
  public enableRotation?: boolean;
  ngOnInit(): void {
    this.dataSource = [
      { x: 1, y: 7 }, { x: 2, y: 1 }, { x: 3, y: 1 }, { x: 4, y: 14 },
      { x: 5, y: 1 }, { x: 6, y: 10 },
      { x: 7, y: 8 }, { x: 8, y: 6 }, { x: 9, y: 10 }, { x: 10, y: 10 },
      { x: 11, y: 16 }, { x: 12, y: 6 },
      { x: 13, y: 14 }, { x: 14, y: 7 }, { x: 15, y: 5 }, { x: 16, y: 2 },
      { x: 17, y: 14 }, { x: 18, y: 7 },
      { x: 19, y: 7 }, { x: 20, y: 10 }
    ];
    this.enableRotation = true;
    this.primaryXAxis = {
      valueType: 'Double'
    };
    this.primaryYAxis = {
      // Custom label format
      labelFormat: '`${value}K'
    }
  }
}
```

MAIN.TS

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

<!-- markdownlint-disable MD036 -->

DateTime axis in Angular 3D Chart control

DateTime axis

DateTime axis uses date time scale and displays the date time values as axis labels in the specified format.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts'
import { Component } from '@angular/core';
@Component({
  imports: [
    Chart3DAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the Chart component
  template: `<ejs-chart3d [primaryXAxis]='primaryXAxis'
[primaryYAxis]="primaryYAxis"
rotation=7 tilt=10 depth=100 [enableRotation]='enableRotation'>
<e-chart3d-series-collection>
  <e-chart3d-series [dataSource]='dataSource' type='Column' xName='x'
yName='y'>
    </e-chart3d-series>
  </e-chart3d-series-collection>
</ejs-chart3d>`
})
export class AppComponent {
  public dataSource?: Object[];
  public primaryXAxis?: Object;
  public enableRotation?: boolean;
  public primaryYAxis?: Object;
  ngOnInit(): void {
    this.dataSource = [
      { x: new Date(2000, 6, 11), y: 10 },
      { x: new Date(2002, 3, 7), y: 30 },
      { x: new Date(2004, 3, 6), y: 15 },
      { x: new Date(2006, 3, 30), y: 65 },
      { x: new Date(2008, 3, 8), y: 90 },
      { x: new Date(2010, 3, 8), y: 85 }
    ];
    this.primaryXAxis = {
      valueType: "DateTime"
    };
  }
}
```

```

        this.enableRotation = true;
        this.primaryYAxis = { };
    }
}

```

MAIN.TS

```

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

```

Note: To use datetime axis, we need to inject `DateTime3DService` module into the `@NgModule.providers` and set the `valueType` of axis to `DateTime`.

DateTime category axis

DateTime category axis is used to display the date time values with non-linear intervals. For example, the business days alone have been depicted in a week here.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts';
import { Component } from '@angular/core';
@Component({
  imports: [
    Chart3DAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the Chart component
  template: `<ejs-chart3d [primaryXAxis]='primaryXAxis'
[primaryYAxis]="primaryYAxis"
rotation=7 tilt=10 depth=100 [enableRotation]='enableRotation'>
<e-chart3d-series-collection>
  <e-chart3d-series [dataSource]='dataSource' type='Column' xName='x'
yName='y'>
    </e-chart3d-series>
  </e-chart3d-series-collection>
</ejs-chart3d>`
})
export class AppComponent {
  public dataSource?: Object[];
  public primaryXAxis?: Object;
  public enableRotation?: boolean;
  public primaryYAxis?: Object;
  ngOnInit(): void {
    this.dataSource = [
      { x: new Date(2017, 11, 20), y: 21 }, { x: new Date(2017, 11,
21), y: 24 },
      { x: new Date(2017, 11, 22), y: 24 }, { x: new Date(2017, 11,
26), y: 70 },
      { x: new Date(2017, 11, 27), y: 75 }, { x: new Date(2018, 0, 2),
y: 82 },

```

```

54     { x: new Date(2018, 0, 3), y: 53 }, { x: new Date(2018, 0, 4), y:
45     { x: new Date(2018, 0, 5), y: 53 }, { x: new Date(2018, 0, 8), y:
    };
    this.primaryXAxis = {
        valueType: 'DateTimeCategory',
        skeleton: 'Ed',
    };
    this.enableRotation = true;
    this.primaryYAxis = { };
}

```

MAIN.TS

```

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

```

Note: To use datetime category axis, we need to inject **DateTimeCategory3DService** module into the **@NgModule.providers** and set the [valueType](#) of axis to **DateTimeCategory**.

Range

Range of an axis will be calculated automatically based on the provided data. You can also customize the range of an axis using [minimum](#), [maximum](#) and [interval](#) properties.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts'
import { Component } from '@angular/core';
@Component({
  imports: [
    Chart3DAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the Chart component
  template: `<ejs-chart3d [primaryXAxis]='primaryXAxis'
[primaryYAxis]="primaryYAxis"
rotation=7 tilt=10 depth=100 [enableRotation]='enableRotation'>
<e-chart3d-series-collection>
  <e-chart3d-series [dataSource]='dataSource' type='Column' xName='x'
yName='y'>
  </e-chart3d-series>
</e-chart3d-series-collection>
</ejs-chart3d>`
})
export class AppComponent {
  public dataSource?: Object[];
  public primaryXAxis?: Object;
  public enableRotation?: boolean;

```

```

    public primaryYAxis?: Object;
    ngOnInit(): void {
        this.dataSource = [
            { x: new Date(2000, 6, 11), y: 10 }, { x: new Date(2002, 3, 7),
y: 30 },
            { x: new Date(2004, 3, 6), y: 15 }, { x: new Date(2006, 3, 30),
y: 65 },
            { x: new Date(2008, 3, 8), y: 90 }, { x: new Date(2010, 3, 8), y:
85 }
        ];
        this.primaryXAxis = {
            valueType: 'DateTime',
            minimum: new Date(2000, 6, 1),
            maximum: new Date(2010, 6, 1), interval: 1
        };
        this.enableRotation = true;
        this.primaryYAxis = { };
    }
}

```

MAIN.TS

```

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

```

Interval customization

Date time intervals can be customized by using the [interval](#) and [intervalType](#) properties of the axis. For example, when you set [interval](#) as **2** and [intervalType](#) as **Years**, it considers 2 years as interval.

DateTime axis supports following interval types,

- Auto
- Years
- Months
- Days
- Hours
- Minutes
- Seconds

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts';
import { Component } from '@angular/core';
@Component({
  imports: [
    Chart3DAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the Chart component

```

```

    template: `<ejs-chart3d [primaryXAxis]='primaryXAxis'
[primaryYAxis]="primaryYAxis"
    rotation=7 tilt=10 depth=100 [enableRotation]='enableRotation'>
    <e-chart3d-series-collection>
        <e-chart3d-series [dataSource]='dataSource' type='Column' xName='x'
yName='y'>
        </e-chart3d-series>
    </e-chart3d-series-collection>
</ejs-chart3d>`
    })
    export class AppComponent {
        public dataSource?: Object[];
        public primaryXAxis?: Object;
        public enableRotation?: boolean;
        public primaryYAxis?: Object;
        ngOnInit(): void {
            this.dataSource = [
                { x: new Date(2000, 6, 11), y: 10 }, { x: new Date(2002, 3, 7),
y: 30 },
                { x: new Date(2004, 3, 6), y: 15 }, { x: new Date(2006, 3, 30),
y: 65 },
                { x: new Date(2008, 3, 8), y: 90 }, { x: new Date(2010, 3, 8), y:
85 }
            ];
            this.primaryXAxis = {
                valueType: 'DateTime',
                intervalType: 'Years'
            };
            this.enableRotation = true;
            this.primaryYAxis = { };
        }
    }
}

```

MAIN.TS

```

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

```

Applying padding to the range

Padding can be applied to the minimum and maximum extremes of the range by using the [rangePadding](#) property. DateTime axis supports the following types of padding,

- None
- Round
- Additional

DateTime - None

When the [rangePadding](#) is set to **None**, minimum and maximum of an axis is based on the data.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts'
import { Component } from '@angular/core';
@Component({
  imports: [
    Chart3DAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the Chart component
  template: `<ejs-chart3d [primaryXAxis]='primaryXAxis'
[primaryYAxis]="primaryYAxis"
rotation=7 tilt=10 depth=100 [enableRotation]='enableRotation'>
<e-chart3d-series-collection>
  <e-chart3d-series [dataSource]='dataSource' type='Column' xName='x'
yName='y'>
  </e-chart3d-series>
</e-chart3d-series-collection>
</ejs-chart3d>`
})
export class AppComponent {
  public dataSource?: Object[];
  public primaryXAxis?: Object;
  public enableRotation?: boolean;
  public primaryYAxis?: Object;
  ngOnInit(): void {
    this.dataSource = [
      { x: new Date(2017, 11, 20), y: 21 }, { x: new Date(2017, 11,
21), y: 24 },
      { x: new Date(2017, 11, 22), y: 24 }, { x: new Date(2017, 11,
26), y: 70 },
      { x: new Date(2017, 11, 27), y: 75 }, { x: new Date(2018, 0, 2),
y: 82 },
      { x: new Date(2018, 0, 3), y: 53 }, { x: new Date(2018, 0, 4), y:
54 },
      { x: new Date(2018, 0, 5), y: 53 }, { x: new Date(2018, 0, 8), y:
45 }
    ];
    this.primaryXAxis = {
      valueType: 'DateTime',
      rangePadding: 'None'
    };
    this.enableRotation = true;
    this.primaryYAxis = { };
  }
}

```

MAIN.TS

```

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

```

DateTime - Round

When the [rangePadding](#) is set to **Round**, minimum and maximum will be rounded to the nearest possible value, which is divisible by interval. For example, when the minimum is **15th Jan**, interval is **1** and interval type is **Month**, then the axis minimum will be **Jan 1st**.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts'
import { Component } from '@angular/core';
@Component({
  imports: [
    Chart3DAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the Chart component
  template: `<ejs-chart3d [primaryXAxis]='primaryXAxis'
[primaryYAxis]='primaryYAxis'
  rotation=7 tilt=10 depth=100 [enableRotation]='enableRotation'>
    <e-chart3d-series-collection>
      <e-chart3d-series [dataSource]='dataSource' type='Column' xName='x'
yName='y'>
    </e-chart3d-series>
    </e-chart3d-series-collection>
  </ejs-chart3d>`
})
export class AppComponent {
  public dataSource?: Object[];
  public primaryXAxis?: Object;
  public enableRotation?: boolean;
  public primaryYAxis?: Object;
  ngOnInit(): void {
    this.dataSource = [
      { x: new Date(2017, 11, 20), y: 21 }, { x: new Date(2017, 11,
21), y: 24 },
      { x: new Date(2017, 11, 22), y: 24 }, { x: new Date(2017, 11,
26), y: 70 },
      { x: new Date(2017, 11, 27), y: 75 }, { x: new Date(2018, 0, 2),
y: 82 },
      { x: new Date(2018, 0, 3), y: 53 }, { x: new Date(2018, 0, 4), y:
54 },
      { x: new Date(2018, 0, 5), y: 53 }, { x: new Date(2018, 0, 8), y:
45 }
    ];
    this.primaryXAxis = {
      valueType: 'DateTime',
      rangePadding: 'Round'
    };
    this.enableRotation = true;
    this.primaryYAxis = { };
  }
}
```


MAIN.TS

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

DateTime - Additional

When the [rangePadding](#) is set to **Additional**, interval of an axis will be padded to the minimum and maximum of the axis.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts'
import { Component } from '@angular/core';
@Component({
  imports: [
    Chart3DAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the Chart component
  template: `<ejs-chart3d [primaryXAxis]='primaryXAxis'
[primaryYAxis]='primaryYAxis'
rotation=7 tilt=10 depth=100 [enableRotation]='enableRotation'>
<e-chart3d-series-collection>
  <e-chart3d-series [dataSource]='dataSource' type='Column' xName='x'
yName='y'>
    </e-chart3d-series>
  </e-chart3d-series-collection>
</ejs-chart3d>`
})
export class AppComponent {
  public dataSource?: Object[];
  public primaryXAxis?: Object;
  public enableRotation?: boolean;
  public primaryYAxis?: Object;
  ngOnInit(): void {
    this.dataSource = [
      { x: new Date(2017, 11, 20), y: 21 }, { x: new Date(2017, 11,
21), y: 24 },
      { x: new Date(2017, 11, 22), y: 24 }, { x: new Date(2017, 11,
26), y: 70 },
      { x: new Date(2017, 11, 27), y: 75 }, { x: new Date(2018, 0, 2),
y: 82 },
      { x: new Date(2018, 0, 3), y: 53 }, { x: new Date(2018, 0, 4), y:
54 },
      { x: new Date(2018, 0, 5), y: 53 }, { x: new Date(2018, 0, 8), y:
45 }
    ];
    this.primaryXAxis = {
      valueType: 'DateTime',
      rangePadding: 'Additional'
    };
  }
}
```

```

        this.enableRotation = true;
        this.primaryYAxis = { };
    }
}

```

MAIN.TS

```

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

```

Label format

The date can be formatted and parsed to all globalize format using the [labelFormat](#) property in an axis.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts';
import { Component } from '@angular/core';
@Component({
  imports: [
    Chart3DAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the Chart component
  template: `<ejs-chart3d [primaryXAxis]='primaryXAxis'
[primaryYAxis]="primaryYAxis"
rotation=7 tilt=10 depth=100 [enableRotation]='enableRotation'>
<e-chart3d-series-collection>
  <e-chart3d-series [dataSource]='dataSource' type='Column' xName='x'
yName='y'>
    </e-chart3d-series>
  </e-chart3d-series-collection>
</ejs-chart3d>`
})
export class AppComponent {
  public dataSource?: Object[];
  public primaryXAxis?: Object;
  public enableRotation?: boolean;
  public primaryYAxis?: Object;
  ngOnInit(): void {
    this.dataSource = [
      { x: new Date(2017, 11, 20), y: 21 }, { x: new Date(2017, 11,
21), y: 24 },
      { x: new Date(2017, 11, 22), y: 24 }, { x: new Date(2017, 11,
26), y: 70 },
      { x: new Date(2017, 11, 27), y: 75 }, { x: new Date(2018, 0, 2),
y: 82 },
      { x: new Date(2018, 0, 3), y: 53 }, { x: new Date(2018, 0, 4), y:
54 },
      { x: new Date(2018, 0, 5), y: 53 }, { x: new Date(2018, 0, 8), y:
45 }
    ];
  }
}

```

```

        this.primaryXAxis = {
            valueType: 'DateTime',
            labelFormat: 'yMd'
        };
        this.enableRotation = true;
        this.primaryYAxis = { };
    }
}

```

MAIN.TS

```

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

```

The following table describes the result of applying some common date time formats to the **labelFormat** property.

<!-- markdownlint-disable MD033 -->

Label Value	Label Format Property Value	Result	Description
new Date(2000, 03, 10)	EEEE	Monday	The Date is displayed in day format.
new Date(2000, 03, 10)	yMd	04/10/2000	The Date is displayed in month/date/year format.
new Date(2000, 03, 10)	MMM	Apr	The Shorthand month for the date is displayed.
new Date(2000, 03, 10)	hm	12:00 AM	Time of the date value is displayed as label.
new Date(2000, 03, 10)	hms	12:00:00 AM	The Label is displayed in hours:minutes:seconds format.

Custom label format

Axis also supports custom label format using placeholder like {value}°C, in which the value represent the axis label e.g 20°C.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts';
import { Component } from '@angular/core';
@Component({
    imports: [
        Chart3DAllModule
    ],
    standalone: true,

```

```

    selector: 'app-container',
    // specifies the template string for the Chart component
    template: `<ejs-chart3d [primaryXAxis]='primaryXAxis'
[primaryYAxis]='primaryYAxis'
    rotation=7 tilt=10 depth=100 [enableRotation]='enableRotation'>
    <e-chart3d-series-collection>
        <e-chart3d-series [dataSource]='dataSource' type='Column' xName='x'
yName='y'>
        </e-chart3d-series>
    </e-chart3d-series-collection>
</ejs-chart3d>`
  })
  export class AppComponent {
    public dataSource?: Object[];
    public primaryXAxis?: Object;
    public enableRotation?: boolean;
    public primaryYAxis?: Object;
    ngOnInit(): void {
      this.dataSource = [
        { x: new Date(2017, 11, 20), y: 21 }, { x: new Date(2017, 11,
21), y: 24 },
        { x: new Date(2017, 11, 22), y: 24 }, { x: new Date(2017, 11,
26), y: 70 },
        { x: new Date(2017, 11, 27), y: 75 }, { x: new Date(2018, 0, 2),
y: 82 },
        { x: new Date(2018, 0, 3), y: 53 }, { x: new Date(2018, 0, 4), y:
54 },
        { x: new Date(2018, 0, 5), y: 53 }, { x: new Date(2018, 0, 8), y:
45 }
      ];
      this.primaryXAxis = {
        valueType: 'DateTime'
      };
      this.enableRotation = true;
      this.primaryYAxis = {
        labelFormat: '${value}K'
      };
    }
  }
}

```

MAIN.TS

```

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

```

Logarithmic axis in Angular 3D Chart control

Logarithmic axis uses logarithmic scale and it is very useful in visualizing data, when it has numerical values in both lower order of magnitude (eg: 10⁻⁶) and higher order of magnitude (eg: 10⁶).

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'

```

```

import { BrowserModule } from '@angular/platform-browser'
import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts'
import { Component } from '@angular/core';
@Component({
  imports: [
    Chart3DAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the Chart component
  template: `<ejs-chart3d #chart style='display:block;' align='center'
[primaryXAxis]='primaryXAxis' [primaryYAxis]="primaryYAxis"
rotation=7 tilt=10 depth=100 [enableRotation]='enableRotation'>
<e-chart3d-series-collection>
  <e-chart3d-series [dataSource]='dataSource' type='Column' xName='x'
yName='y'>
  </e-chart3d-series>
</e-chart3d-series-collection>
</ejs-chart3d>`
})
export class AppComponent {
  public dataSource?: Object[];
  public primaryXAxis?: Object;
  public primaryYAxis?: Object;
  public enableRotation?: boolean;
  ngOnInit(): void {
    this.dataSource = [
      { x: new Date(1995, 0, 1), y: 80 }, { x: new Date(1996, 0, 1), y:
200 },
      { x: new Date(1997, 0, 1), y: 400 }, { x: new Date(1998, 0, 1),
y: 600 },
      { x: new Date(1999, 0, 1), y: 700 }, { x: new Date(2000, 0, 1),
y: 1400 },
      { x: new Date(2001, 0, 1), y: 2000 }, { x: new Date(2002, 0, 1),
y: 4000 },
      { x: new Date(2003, 0, 1), y: 6000 }, { x: new Date(2004, 0, 1),
y: 8000 },
      { x: new Date(2005, 0, 1), y: 11000 }
    ];
    this.enableRotation = true;
    this.primaryXAxis = {
      valueType: 'DateTime'
    };
    this.primaryYAxis = {
      valueType: 'Logarithmic'
    };
  }
}

```

MAIN.TS

```

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

```

Note: To use logarithmic axis, we need to inject `Logarithmic3DService` module into the `@NgModule.providers` and set the `valueType` of the axis to `Logarithmic`.

Range

The range of an axis will be calculated automatically based on the provided data and it can also be customized by using the `minimum`, `maximum` and `interval` properties of the axis.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts'
import { Component } from '@angular/core';
@Component({
  imports: [
    Chart3DAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the Chart component
  template: `<ejs-chart3d #chart style='display:block;' align='center'
[primaryXAxis]='primaryXAxis' [primaryYAxis]='primaryYAxis'
rotation=7 tilt=10 depth=100 [enableRotation]='enableRotation'>
<e-chart3d-series-collection>
  <e-chart3d-series [dataSource]='dataSource' type='Column' xName='x'
yName='y'>
  </e-chart3d-series>
</e-chart3d-series-collection>
</ejs-chart3d>`
})
export class AppComponent {
  public dataSource?: Object[];
  public primaryXAxis?: Object;
  public primaryYAxis?: Object;
  public enableRotation?: boolean;
  ngOnInit(): void {
    this.dataSource = [
      { x: new Date(1995, 0, 1), y: 80 }, { x: new Date(1996, 0, 1), y:
200 },
      { x: new Date(1997, 0, 1), y: 400 }, { x: new Date(1998, 0, 1),
y: 600 },
      { x: new Date(1999, 0, 1), y: 700 }, { x: new Date(2000, 0, 1),
y: 1400 },
      { x: new Date(2001, 0, 1), y: 2000 }, { x: new Date(2002, 0, 1),
y: 4000 },
      { x: new Date(2003, 0, 1), y: 6000 }, { x: new Date(2004, 0, 1),
y: 8000 },
      { x: new Date(2005, 0, 1), y: 11000 }
    ];
    this.enableRotation = true;
    this.primaryXAxis = {
      valueType: 'DateTime'
    };
    this.primaryYAxis = {
      valueType: 'Logarithmic',
      minimum: 100,
      maximum: 10000,

```

```

        interval: 1000
    };
}
}

```

MAIN.TS

```

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

```

Logarithmic base

Logarithmic base can be customized by using the [logBase](#) property of the axis. For example when the **logBase** is 5, the axis values follows 5^{-2} , 5^{-1} , 5^0 , 5^1 , 5^2 etc.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts';
import { Component } from '@angular/core';
@Component({
  imports: [
    Chart3DAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the Chart component
  template: `<ejs-chart3d #chart style='display:block;' align='center'
[primaryXAxis]='primaryXAxis' [primaryYAxis]="primaryYAxis"
rotation=7 tilt=10 depth=100 [enableRotation]='enableRotation'>
<e-chart3d-series-collection>
  <e-chart3d-series [dataSource]='dataSource' type='Column' xName='x'
yName='y'>
    </e-chart3d-series>
  </e-chart3d-series-collection>
</ejs-chart3d>`
})
export class AppComponent {
  public dataSource?: Object[];
  public primaryXAxis?: Object;
  public primaryYAxis?: Object;
  public enableRotation?: boolean;
  ngOnInit(): void {
    this.dataSource = [
      { x: new Date(1995, 0, 1), y: 80 }, { x: new Date(1996, 0, 1), y:
200 },
      { x: new Date(1997, 0, 1), y: 400 }, { x: new Date(1998, 0, 1),
y: 600 },
      { x: new Date(1999, 0, 1), y: 700 }, { x: new Date(2000, 0, 1),
y: 1400 },
      { x: new Date(2001, 0, 1), y: 2000 }, { x: new Date(2002, 0, 1),
y: 4000 },

```

```

        { x: new Date(2003, 0, 1), y: 6000 }, { x: new Date(2004, 0, 1),
y: 8000 },
        { x: new Date(2005, 0, 1), y: 11000 }
    ];
    this.enableRotation = true;
    this.primaryXAxis = {
        valueType: 'DateTime'
    };
    this.primaryYAxis = {
        valueType: 'Logarithmic',
        logBase: 2
    };
}
}

```

MAIN.TS

```

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

```

Logarithmic interval

The interval of the logarithmic axis can be customized by using the [interval](#) property in the axis. When the logarithmic base is 10 and logarithmic **interval** is 2, then the axis labels are placed at an interval of 10^2 . The default value of the [interval](#) is 1.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts'
import { Component } from '@angular/core';
@Component({
  imports: [
    Chart3DAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the Chart component
  template: `<ejs-chart3d #chart style='display:block;' align='center'
[primaryXAxis]='primaryXAxis' [primaryYAxis]="primaryYAxis"
rotation=7 tilt=10 depth=100 [enableRotation]='enableRotation'>
<e-chart3d-series-collection>
  <e-chart3d-series [dataSource]='dataSource' type='Column' xName='x'
yName='y'>
    </e-chart3d-series>
  </e-chart3d-series-collection>
</ejs-chart3d>`
})
export class AppComponent {
  public dataSource?: Object[];
  public primaryXAxis?: Object;
  public primaryYAxis?: Object;
  public enableRotation?: boolean;
}

```



```

ngOnInit(): void {
    this.dataSource = [
        { x: new Date(1995, 0, 1), y: 80 }, { x: new Date(1996, 0, 1), y:
200 },
        { x: new Date(1997, 0, 1), y: 400 }, { x: new Date(1998, 0, 1),
y: 600 },
        { x: new Date(1999, 0, 1), y: 700 }, { x: new Date(2000, 0, 1),
y: 1400 },
        { x: new Date(2001, 0, 1), y: 2000 }, { x: new Date(2002, 0, 1),
y: 4000 },
        { x: new Date(2003, 0, 1), y: 6000 }, { x: new Date(2004, 0, 1),
y: 8000 },
        { x: new Date(2005, 0, 1), y: 11000 }
    ];
    this.enableRotation = true;
    this.primaryXAxis = {
        valueType: 'DateTime'
    };
    this.primaryYAxis = {
        valueType: 'Logarithmic',
        interval: 2
    };
}
}

```

MAIN.TS

```

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

```

Axis labels in Angular 3D Chart control

Axis labels are the labels that are positioned adjacent to the y-axis and beneath the x-axis. It provides descriptive information about the axis.

Smart axis labels

When the axis labels overlap with each other, [labelIntersectAction](#) property in the axis can be used to place them smartly.

Case 1: When setting `labelIntersectAction` as `Hide`.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts'
import { Component } from '@angular/core';
@Component({
  imports: [
    Chart3DAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the Chart component

```

```

    template: `<ejs-chart3d style='display:block;' align='center'
[primaryXAxis]='primaryXAxis' [primaryYAxis]="primaryYAxis"
    rotation=7 tilt=10 depth=100 [enableRotation]='enableRotation'>
    <e-chart3d-series-collection>
        <e-chart3d-series [dataSource]='dataSource' type='Column' xName='x'
yName='y'>
        </e-chart3d-series>
    </e-chart3d-series-collection>
</ejs-chart3d>`
    })
    export class AppComponent {
        public dataSource?: Object[];
        public primaryXAxis?: Object;
        public enableRotation?: boolean;
        public primaryYAxis?: Object;
        ngOnInit(): void {
            this.dataSource = [
                { x: "South Korea", y: 39.4 },
                { x: "India", y: 61.3 },
                { x: "Pakistan", y: 20.4 },
                { x: "Germany", y: 65.1 },
                { x: "Australia", y: 15.8 },
                { x: "Italy", y: 29.2 },
                { x: "United Kingdom", y: 44.6 },
                { x: "Saudi Arabia", y: 9.7 },
                { x: "Russia", y: 40.8 },
                { x: "Mexico", y: 31 },
                { x: "Brazil", y: 75.9 },
                { x: "China", y: 51.4 }
            ];
            this.primaryXAxis = {
                valueType: "Category",
                labelIntersectAction: 'Hide'
            };
            this.enableRotation = true;
            this.primaryYAxis = { };
        }
    }
}

```

MAIN.TS

```

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

```

Case 2: When setting `labelIntersectAction` as `Rotate45`.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts'
import { Component } from '@angular/core';
@Component({
  imports: [

```

```

    Chart3DAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the Chart component
  template: `<ejs-chart3d style='display:block;' align='center'
[primaryXAxis]='primaryXAxis' [primaryYAxis]="primaryYAxis"
rotation=7 tilt=10 depth=100 [enableRotation]='enableRotation'>
  <e-chart3d-series-collection>
    <e-chart3d-series [dataSource]='dataSource' type='Column' xName='x'
yName='y'>
    </e-chart3d-series>
  </e-chart3d-series-collection>
</ejs-chart3d>`
  })
export class AppComponent {
  public dataSource?: Object[];
  public primaryXAxis?: Object;
  public enableRotation?: boolean;
  public primaryYAxis?: Object;
  ngOnInit(): void {
    this.dataSource = [
      { x: "South Korea", y: 39.4 },
      { x: "India", y: 61.3 },
      { x: "Pakistan", y: 20.4 },
      { x: "Germany", y: 65.1 },
      { x: "Australia", y: 15.8 },
      { x: "Italy", y: 29.2 },
      { x: "United Kingdom", y: 44.6 },
      { x: "Saudi Arabia", y: 9.7 },
      { x: "Russia", y: 40.8 },
      { x: "Mexico", y: 31 },
      { x: "Brazil", y: 75.9 },
      { x: "China", y: 51.4 }
    ];
    this.primaryXAxis = {
      valueType: "Category",
      labelIntersectAction: 'Rotate45'
    };
    this.enableRotation = true;
    this.primaryYAxis = { };
  }
}

```

MAIN.TS

```

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

```

Case 3: When setting `labelIntersectAction` as `Rotate90`.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'

```

```

import { BrowserModule } from '@angular/platform-browser'
import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts'
import { Component } from '@angular/core';
@Component({
  imports: [
    Chart3DAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the Chart component
  template: `<ejs-chart3d style='display:block;' align='center'
[primaryXAxis]='primaryXAxis' [primaryYAxis]="primaryYAxis"
rotation=7 tilt=10 depth=100 [enableRotation]='enableRotation'>
<e-chart3d-series-collection>
  <e-chart3d-series [dataSource]='dataSource' type='Column' xName='x'
yName='y'>
  </e-chart3d-series>
</e-chart3d-series-collection>
</ejs-chart3d>`
})
export class AppComponent {
  public dataSource?: Object[];
  public primaryXAxis?: Object;
  public enableRotation?: boolean;
  public primaryYAxis?: Object;
  ngOnInit(): void {
    this.dataSource = [
      { x: "South Korea", y: 39.4 },
      { x: "India", y: 61.3 },
      { x: "Pakistan", y: 20.4 },
      { x: "Germany", y: 65.1 },
      { x: "Australia", y: 15.8 },
      { x: "Italy", y: 29.2 },
      { x: "United Kingdom", y: 44.6 },
      { x: "Saudi Arabia", y: 9.7 },
      { x: "Russia", y: 40.8 },
      { x: "Mexico", y: 31 },
      { x: "Brazil", y: 75.9 },
      { x: "China", y: 51.4 }
    ];
    this.primaryXAxis = {
      valueType: "Category",
      labelIntersectAction: 'Rotate90'
    };
    this.enableRotation = true;
    this.primaryYAxis = { };
  }
}

```

MAIN.TS

```

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

```

Edge label placement

Labels with long text at the edges of an axis may appear partially in the 3D chart. To avoid this, use the [edgeLabelPlacement](#) property in axis, which moves the label inside the chart area for better appearance or hides it.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts'
import { Component } from '@angular/core';
@Component({
  imports: [
    Chart3DAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the Chart component
  template: `<ejs-chart3d style='display:block;' align='center'
[primaryXAxis]='primaryXAxis' [primaryYAxis]="primaryYAxis"
rotation=7 tilt=10 depth=100 [enableRotation]='enableRotation'>
<e-chart3d-series-collection>
  <e-chart3d-series [dataSource]='dataSource' type='Column' xName='x'
yName='y'>
  </e-chart3d-series>
</e-chart3d-series-collection>
</ejs-chart3d>`
})
export class AppComponent {
  public dataSource?: Object[];
  public primaryXAxis?: Object;
  public enableRotation?: boolean;
  public primaryYAxis?: Object;
  ngOnInit(): void {
    this.dataSource = [
      { x: "South Korea", y: 39.4 },
      { x: "India", y: 61.3 },
      { x: "Pakistan", y: 20.4 },
      { x: "Germany", y: 65.1 },
      { x: "Australia", y: 15.8 },
      { x: "Italy", y: 29.2 },
      { x: "United Kingdom", y: 44.6 },
      { x: "Saudi Arabia", y: 9.7 },
      { x: "Russia", y: 40.8 },
      { x: "Mexico", y: 31 },
      { x: "Brazil", y: 75.9 },
      { x: "China", y: 51.4 }
    ];
    this.primaryXAxis = {
      valueType: "Category",
      edgeLabelPlacement: 'Shift'
    };
    this.enableRotation = true;
    this.primaryYAxis = { };
  }
}
```

MAIN.TS

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

Maximum labels

The labels will be rendered based on the count in the [maximumLabels](#) property per 100 pixel. If the range (minimum, maximum, interval) and [maximumLabels](#) are set, then the priority goes to range. If the range is not set, then the priority goes to [maximumLabels](#) property.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts'
import { Component } from '@angular/core';
@Component({
  imports: [
    Chart3DAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the Chart component
  template: `<ejs-chart3d style='display:block;' align='center'
[primaryXAxis]='primaryXAxis' [primaryYAxis]="primaryYAxis"
rotation=7 tilt=10 depth=100 [enableRotation]='enableRotation'>
<e-chart3d-series-collection>
  <e-chart3d-series [dataSource]='dataSource' type='Column' xName='x'
yName='y'>
  </e-chart3d-series>
</e-chart3d-series-collection>
</ejs-chart3d>`
})
export class AppComponent {
  public dataSource?: Object[];
  public primaryXAxis?: Object;
  public enableRotation?: boolean;
  public primaryYAxis?: Object;
  ngOnInit(): void {
    let series1: Object[] = [];
    let point1: Object;
    let value: number = 80;
    let i: number;
    for (i = 1; i < 50; i++) {
      if (Math.random() > .5) {
        value += Math.random();
      } else {
        value -= Math.random();
      }
      point1 = { x: i, y: value.toFixed(1) };
      series1.push(point1);
    }
  }
}
```

```

        this.dataSource = series1;
        this.primaryXAxis = {
            title: 'Years',
            edgeLabelPlacement: 'Shift',
            majorGridLines: { width: 0 },
            maximumLabels: 1,
        };
        this.enableRotation = true;
        this.primaryYAxis = { };
    }
}

```

MAIN.TS

```

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

```

Axis customization in Angular 3D Chart control

Title

The title for the axis can be added by using the [title](#) property. It helps to provide quick information to the user about the data plotted in the axis. Title style can be customized using [titleStyle](#) property of the axis.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts';
import { Component } from '@angular/core';
@Component({
  imports: [
    Chart3DAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the Chart component
  template: `<ejs-chart3d style='display:block;' align='center'
[primaryXAxis]='primaryXAxis'
rotation=7 tilt=10 depth=100 [enableRotation]='enableRotation'>
<e-chart3d-series-collection>
  <e-chart3d-series [dataSource]='dataSource' type='Column'
xName='country' yName='gold'>
    </e-chart3d-series>
  </e-chart3d-series-collection>
</ejs-chart3d>`
})
export class AppComponent {
  public dataSource?: Object[];
  public primaryXAxis?: Object;
  public enableRotation?: boolean;
  ngOnInit(): void {
    this.dataSource = [
      { country: "USA", gold: 50, silver: 70, bronze: 45 },

```

```

        { country: "China", gold: 40, silver: 60, bronze: 55 },
        { country: "Japan", gold: 70, silver: 60, bronze: 50 },
        { country: "Australia", gold: 60, silver: 56, bronze: 40 },
        { country: "France", gold: 50, silver: 45, bronze: 35 },
        { country: "Germany", gold: 40, silver: 30, bronze: 22 },
        { country: "Italy", gold: 40, silver: 35, bronze: 37 },
        { country: "Sweden", gold: 30, silver: 25, bronze: 27 }
    ];
    this.primaryXAxis = {
        valueType: "Category",
        title: 'Countries',
        //Axis title text style
        titleStyle: {
            size: '16px', color: 'grey',
            fontFamily : 'Segoe UI', fontWeight : 'bold'
        }
    };
    this.enableRotation = true;
}
}

```

MAIN.TS

```

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

```

Title rotation

The title can be rotated from 0 to 360 degree by using the [titleRotation](#) property.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts'
import { Component } from '@angular/core';
@Component({
  imports: [
    Chart3DAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the Chart component
  template: `<ejs-chart3d style='display:block;' align='center'
[primaryXAxis]='primaryXAxis'
rotation=7 tilt=10 depth=100 [enableRotation]='enableRotation'>
<e-chart3d-series-collection>
  <e-chart3d-series [dataSource]='dataSource' type='Column'
xName='country' yName='gold'>
    </e-chart3d-series>
  </e-chart3d-series-collection>
</ejs-chart3d>`
})
export class AppComponent {
  public dataSource?: Object[];
}

```



```

public primaryXAxis?: Object;
public enableRotation?: boolean;
ngOnInit(): void {
    this.dataSource = [
        { country: "USA", gold: 50, silver: 70, bronze: 45 },
        { country: "China", gold: 40, silver: 60, bronze: 55 },
        { country: "Japan", gold: 70, silver: 60, bronze: 50 },
        { country: "Australia", gold: 60, silver: 56, bronze: 40 },
        { country: "France", gold: 50, silver: 45, bronze: 35 },
        { country: "Germany", gold: 40, silver: 30, bronze: 22 },
        { country: "Italy", gold: 40, silver: 35, bronze: 37 },
        { country: "Sweden", gold: 30, silver: 25, bronze: 27 }
    ];
    this.primaryXAxis = {
        valueType: "Category",
        title: 'Countries',
        titleRotation: 90,
        //Axis title text style
        titleStyle: {
            size: '16px', color: 'grey',
            fontFamily : 'Segoe UI', fontWeight : 'bold'
        }
    };
    this.enableRotation = true;
}
}

```

MAIN.TS

```

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

```

Tick lines customization

The [width](#), [color](#) and [height](#) of the minor and major tick lines can be customized by using the [majorTickLines](#) and [minorTickLines](#) properties in the axis.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts'
import { Component } from '@angular/core';
@Component({
  imports: [
    Chart3DAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the Chart component
  template: `<ejs-chart3d style='display:block;' align='center'
[primaryXAxis]='primaryXAxis'
rotation=7 tilt=10 depth=100 [enableRotation]='enableRotation'>
<e-chart3d-series-collection>

```

```

    <e-chart3d-series [dataSource]='dataSource' type='Column'
    xName='country' yName='gold'>
    </e-chart3d-series>
  </e-chart3d-series-collection>
</ejs-chart3d>`
})
export class AppComponent {
  public dataSource?: Object[];
  public primaryXAxis?: Object;
  public enableRotation?: boolean;
  ngOnInit(): void {
    this.dataSource = [
      { country: "USA", gold: 50, silver: 70, bronze: 45 },
      { country: "China", gold: 40, silver: 60, bronze: 55 },
      { country: "Japan", gold: 70, silver: 60, bronze: 50 },
      { country: "Australia", gold: 60, silver: 56, bronze: 40 },
      { country: "France", gold: 50, silver: 45, bronze: 35 },
      { country: "Germany", gold: 40, silver: 30, bronze: 22 },
      { country: "Italy", gold: 40, silver: 35, bronze: 37 },
      { country: "Sweden", gold: 30, silver: 25, bronze: 27 }
    ];
    this.primaryXAxis = {
      valueType: "Category",
      majorTickLines : {
        color : 'blue',
        width : 5
      }
    };
    this.enableRotation = true;
  }
}

```

MAIN.TS

```

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

```

Grid lines customization

The [width](#) and [color](#) of the minor and major grid lines can be customized by using the [majorGridLines](#) and [minorGridLines](#) properties in the axis.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts'
import { Component } from '@angular/core';
@Component({
  imports: [
    Chart3DAllModule
  ],
  standalone: true,
  selector: 'app-container',

```

```
// specifies the template string for the Chart component
template: `<ejs-chart3d style='display:block;' align='center'
[primaryXAxis]='primaryXAxis'
rotation=7 tilt=10 depth=100 [enableRotation]='enableRotation'>
  <e-chart3d-series-collection>
    <e-chart3d-series [dataSource]='dataSource' type='Column'
xName='country' yName='gold'>
    </e-chart3d-series>
  </e-chart3d-series-collection>
</ejs-chart3d>`
})
export class AppComponent {
  public dataSource?: Object[];
  public primaryXAxis?: Object;
  public enableRotation?: boolean;
  ngOnInit(): void {
    this.dataSource = [
      { country: "USA", gold: 50, silver: 70, bronze: 45 },
      { country: "China", gold: 40, silver: 60, bronze: 55 },
      { country: "Japan", gold: 70, silver: 60, bronze: 50 },
      { country: "Australia", gold: 60, silver: 56, bronze: 40 },
      { country: "France", gold: 50, silver: 45, bronze: 35 },
      { country: "Germany", gold: 40, silver: 30, bronze: 22 },
      { country: "Italy", gold: 40, silver: 35, bronze: 37 },
      { country: "Sweden", gold: 30, silver: 25, bronze: 27 }
    ];
    this.primaryXAxis = {
      valueType: "Category",
      majorGridLines : {
        color : 'blue',
        width : 1
      }
    };
    this.enableRotation = true;
  }
}
```

MAIN.TS

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

Multiple axis

In addition to primary X and Y axis, n number of axis can be added to the chart. Series can be associated with this axis, by mapping with axis's unique name.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts'
import { Component } from '@angular/core';
@Component({
```

```

imports: [
    Chart3DAllModule
],
standalone: true,
selector: 'app-container',
// specifies the template string for the Chart component
template: `<ejs-chart3d style='display:block;' align='center'
[primaryXAxis]='primaryXAxis' [axes]="axes"
rotation=7 tilt=10 depth=100 [enableRotation]='enableRotation'>
<e-chart3d-series-collection>
    <e-chart3d-series [dataSource]='dataSource' type='Column'
xName='country' yName='gold'>
    </e-chart3d-series>
    <e-chart3d-series [dataSource]='dataSource' type='Column'
xName='country' yName='silver' yAxisName="yAxis">
    </e-chart3d-series>
</e-chart3d-series-collection>
</ejs-chart3d>`
))
export class AppComponent {
    public dataSource?: Object[];
    public primaryXAxis?: Object;
    public enableRotation?: boolean;
    public axes?: Object[];
    ngOnInit(): void {
        this.dataSource = [
            { country: "USA", gold: 50, silver: 70, bronze: 45 },
            { country: "China", gold: 40, silver: 60, bronze: 55 },
            { country: "Japan", gold: 70, silver: 60, bronze: 50 },
            { country: "Australia", gold: 60, silver: 56, bronze: 40 },
            { country: "France", gold: 50, silver: 45, bronze: 35 },
            { country: "Germany", gold: 40, silver: 30, bronze: 22 },
            { country: "Italy", gold: 40, silver: 35, bronze: 37 },
            { country: "Sweden", gold: 30, silver: 25, bronze: 27 }
        ];
        this.primaryXAxis = {
            valueType: "Category"
        };
        this.enableRotation = true;
        this.axes = [
            {
                rowIndex: 0,
                name: 'yAxis'
            }
        ]
    }
}

```

MAIN.TS

```

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

```

Inversed axis

```
<!-- markdownlint-disable MD033 -->
```

When an axis is inversed, highest value of the axis comes closer to origin and vice versa. To place an axis in inversed manner, set the [isInversed](#) property to **true**.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts'
import { Component } from '@angular/core';
@Component({
  imports: [
    Chart3DAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the Chart component
  template: `<ejs-chart3d style='display:block;' align='center'
[primaryXAxis]='primaryXAxis' [primaryYAxis]="primaryYAxis"
rotation=7 tilt=10 depth=100 [enableRotation]='enableRotation'>
<e-chart3d-series-collection>
  <e-chart3d-series [dataSource]='dataSource' type='Column'
xName='country' yName='gold'>
    </e-chart3d-series>
  </e-chart3d-series-collection>
</ejs-chart3d>`
})
export class AppComponent {
  public dataSource?: Object[];
  public primaryXAxis?: Object;
  public enableRotation?: boolean;
  public primaryYAxis?: Object;
  ngOnInit(): void {
    this.dataSource = [
      { country: "USA", gold: 50, silver: 70, bronze: 45 },
      { country: "China", gold: 40, silver: 60, bronze: 55 },
      { country: "Japan", gold: 70, silver: 60, bronze: 50 },
      { country: "Australia", gold: 60, silver: 56, bronze: 40 },
      { country: "France", gold: 50, silver: 45, bronze: 35 },
      { country: "Germany", gold: 40, silver: 30, bronze: 22 },
      { country: "Italy", gold: 40, silver: 35, bronze: 37 },
      { country: "Sweden", gold: 30, silver: 25, bronze: 27 }
    ];
    this.primaryXAxis = {
      valueType: "Category",
      isInversed: true
    };
    this.enableRotation = true;
    this.primaryYAxis = {
      isInversed: true
    };
  }
}
```

MAIN.TS

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

Opposed position

To place an axis opposite from its original position, set the [opposedPosition](#) property to **true**.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts'
import { Component } from '@angular/core';
@Component({
  imports: [
    Chart3DAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the Chart component
  template: `<ejs-chart3d style='display:block;' align='center'
[primaryXAxis]='primaryXAxis' [primaryYAxis]="primaryYAxis"
rotation=7 tilt=10 depth=100 [enableRotation]='enableRotation'>
<e-chart3d-series-collection>
  <e-chart3d-series [dataSource]='dataSource' type='Column'
xName='country' yName='gold'>
    </e-chart3d-series>
  </e-chart3d-series-collection>
</ejs-chart3d>`
})
export class AppComponent {
  public dataSource?: Object[];
  public primaryXAxis?: Object;
  public enableRotation?: boolean;
  public primaryYAxis?: Object;
  ngOnInit(): void {
    this.dataSource = [
      { country: "USA", gold: 50, silver: 70, bronze: 45 },
      { country: "China", gold: 40, silver: 60, bronze: 55 },
      { country: "Japan", gold: 70, silver: 60, bronze: 50 },
      { country: "Australia", gold: 60, silver: 56, bronze: 40 },
      { country: "France", gold: 50, silver: 45, bronze: 35 },
      { country: "Germany", gold: 40, silver: 30, bronze: 22 },
      { country: "Italy", gold: 40, silver: 35, bronze: 37 },
      { country: "Sweden", gold: 30, silver: 25, bronze: 27 }
    ];
    this.primaryXAxis = {
      valueType: "Category"
    };
    this.enableRotation = true;
    this.primaryYAxis = {
      //Axis position as opposite
      opposedPosition: true
    };
  }
}
```

```
    };
  }
}
```

MAIN.TS

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

Multiple panes in Angular 3D Chart control

The chart area can be divided into multiple panes using [rows](#) and [columns](#).

Rows

To split the chart area vertically into number of rows, use [rows](#) property of the 3D chart.

- The space for each row can be allocated by using the [height](#) property. The value can be either in percentage or in pixel.
- To associate a vertical axis to a particular row, specify its index to [rowIndex](#) property of the axis.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts';
import { Component } from '@angular/core';
@Component({
  imports: [
    Chart3DAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the Chart component
  template: `<ejs-chart3d #chart style='display:block;' align='center'
[primaryXAxis]='primaryXAxis' [primaryYAxis]="primaryYAxis" [rows]="rows"
rotation=7 tilt=10 depth=100 [axes]="axes"
[enableRotation]='enableRotation'>
<e-chart3d-series-collection>
<e-chart3d-series [dataSource]='dataSource' type='Column' xName='x'
yName='y'>
</e-chart3d-series>
<e-chart3d-series [dataSource]='dataSource' yAxisName="yAxis"
type='Column' xName='x' yName='y1'>
</e-chart3d-series>
</e-chart3d-series-collection>
</ejs-chart3d>`
})
export class AppComponent {
  public dataSource?: Object[];
  public primaryXAxis?: Object;
  public primaryYAxis?: Object;
  public axes?: Object[];
  public rows?: Object[];
```

```

public enableRotation?: boolean;
ngOnInit(): void {
    this.dataSource = [
        { x: 'Jan', y: 15, y1: 33 }, { x: 'Feb', y: 20, y1: 31 }, { x:
'Mar', y: 35, y1: 30 },
        { x: 'Apr', y: 40, y1: 28 }, { x: 'May', y: 80, y1: 29 }, { x:
'Jun', y: 70, y1: 30 },
        { x: 'Jul', y: 65, y1: 33 }, { x: 'Aug', y: 55, y1: 32 }, { x:
'Sep', y: 50, y1: 34 },
        { x: 'Oct', y: 30, y1: 32 }, { x: 'Nov', y: 35, y1: 32 }, { x:
'Dec', y: 35, y1: 31 }
    ];
    this.enableRotation = true;
    this.primaryXAxis = {
        title: 'Months',
        valueType: 'Category',
        interval: 1
    };
    this.primaryYAxis = {
        minimum: 0, maximum: 90, interval: 20,
        title: 'Temperature (Fahrenheit)',
        labelFormat: '{value}°F'
    }
    this.axes = [
        {
            majorGridLines: { width: 0 },
            rowIndex: 1, opposedPosition: true,
            minimum: 24, maximum: 36, interval: 4,
            name: 'yAxis', title: 'Temperature (Celsius)',
            labelFormat: '{value}°C'
        }
    ];
    this.rows = [
        {
            height: '50%'
        }, {
            height: '50%'
        }
    ]
}
}

```

MAIN.TS

```

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

```

For spanning the vertical axis along multiple rows, use [span](#) property of an axis.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts'

```



```

import { Component } from '@angular/core';
@Component({
  imports: [
    Chart3DAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the Chart component
  template: `<ejs-chart3d #chart style='display:block;' align='center'
[primaryXAxis]='primaryXAxis' [primaryYAxis]="primaryYAxis" [rows]="rows"
rotation=7 tilt=10 depth=100 [axes]="axes"
[enableRotation]='enableRotation'>
    <e-chart3d-series-collection>
      <e-chart3d-series [dataSource]='dataSource' type='Column' xName='x'
yName='y'>
        </e-chart3d-series>
      <e-chart3d-series [dataSource]='dataSource' yAxisName="yAxis"
type='Column' xName='x' yName='y1'>
        </e-chart3d-series>
      </e-chart3d-series-collection>
    </ejs-chart3d>`
})
export class AppComponent {
  public dataSource?: Object[];
  public primaryXAxis?: Object;
  public primaryYAxis?: Object;
  public axes?: Object[];
  public rows?: Object[];
  public enableRotation?: boolean;
  ngOnInit(): void {
    this.dataSource = [
      { x: 'Jan', y: 15, y1: 33 }, { x: 'Feb', y: 20, y1: 31 }, { x:
'Mar', y: 35, y1: 30 },
      { x: 'Apr', y: 40, y1: 28 }, { x: 'May', y: 80, y1: 29 }, { x:
'Jun', y: 70, y1: 30 },
      { x: 'Jul', y: 65, y1: 33 }, { x: 'Aug', y: 55, y1: 32 }, { x:
'Sep', y: 50, y1: 34 },
      { x: 'Oct', y: 30, y1: 32 }, { x: 'Nov', y: 35, y1: 32 }, { x:
'Dec', y: 35, y1: 31 }
    ];
    this.enableRotation = true;
    this.primaryXAxis = {
      title: 'Months',
      valueType: 'Category',
      interval: 1
    };
    this.primaryYAxis = {
      minimum: 0, maximum: 90, interval: 20,
      title: 'Temperature (Fahrenheit)',
      labelFormat: '{value}°F',
      //Span for chart axis
      span: 1
    };
    this.axes = [
      {
        majorGridLines: { width: 0 },
        rowIndex: 1, opposedPosition: true,

```

```

        minimum: 24, maximum: 36, interval: 4,
        name: 'yAxis', title: 'Temperature (Celsius)',
        labelFormat: '{value}°C'
    }
];
this.rows = [
    {
        height: '50%'
    }, {
        height: '50%'
    }
]
}
}

```

MAIN.TS

```

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

```

Columns

To split the chart area horizontally into number of columns, use [columns](#) property of the 3D chart.

- The space for each column can be allocated by using the [width](#) property. The given width can be either in percentage or in pixel.
- To associate a horizontal axis to a particular column, specify its index to [columnIndex](#) property of the axis.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts'
import { Component } from '@angular/core';
@Component({
  imports: [
    Chart3DAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the Chart component
  template: `<ejs-chart3d #chart style='display:block;' align='center'
[primaryXAxis]='primaryXAxis' [primaryYAxis]="primaryYAxis"
[columns]="columns"
rotation=7 tilt=10 depth=100 [axes]="axes"
[enableRotation]='enableRotation'>
<e-chart3d-series-collection>
  <e-chart3d-series [dataSource]='dataSource' type='Column' xName='x'
yName='y'>
  </e-chart3d-series>
  <e-chart3d-series [dataSource]='dataSource' xAxisName="xAxis"
type='Column' xName='x' yName='y1'>

```

```

        </e-chart3d-series>
    </e-chart3d-series-collection>
</ejs-chart3d>`
    })
    export class AppComponent {
        public dataSource?: Object[];
        public primaryXAxis?: Object;
        public primaryYAxis?: Object;
        public axes?: Object[];
        public columns?: Object[];
        public enableRotation?: boolean;
        ngOnInit(): void {
            this.dataSource = [
                { x: 'Jan', y: 15, y1: 33 }, { x: 'Feb', y: 20, y1: 31 }, { x:
'Mar', y: 35, y1: 30 },
                { x: 'Apr', y: 40, y1: 28 }, { x: 'May', y: 80, y1: 29 }, { x:
'Jun', y: 70, y1: 30 },
                { x: 'Jul', y: 65, y1: 33 }, { x: 'Aug', y: 55, y1: 32 }, { x:
'Sep', y: 50, y1: 34 },
                { x: 'Oct', y: 30, y1: 32 }, { x: 'Nov', y: 35, y1: 32 }, { x:
'Dec', y: 35, y1: 31 }
            ];
            this.enableRotation = true;
            this.primaryXAxis = {
                valueType: 'Category',
                interval: 1
            };
            this.primaryYAxis = {
                minimum: 0, maximum: 90, interval: 10,
                title: 'Temperature (Fahrenheit)',
                labelFormat: '{value}°F'
            };
            this.axes = [
                {
                    majorGridLines: { width: 0 },
                    columnIndex: 1,
                    valueType: 'Category',
                    name: 'xAxis'
                }
            ];
            this.columns = [
                {
                    width: '50%'
                }, {
                    width: '50%'
                }
            ];
        }
    }
}

```

MAIN.TS

```

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

```

For spanning the vertical axis along multiple column, you can use [span](#) property of an axis.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts'
import { Component } from '@angular/core';
@Component({
  imports: [
    Chart3DAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the Chart component
  template: `<ejs-chart3d #chart style='display:block;' align='center'
[primaryXAxis]='primaryXAxis' [primaryYAxis]='primaryYAxis'
[columns]="columns"
rotation=7 tilt=10 depth=100 [axes]="axes"
[enableRotation]='enableRotation'>
  <e-chart3d-series-collection>
    <e-chart3d-series [dataSource]='dataSource' type='Column' xName='x'
yName='y'>
      </e-chart3d-series>
    <e-chart3d-series [dataSource]='dataSource' xAxisName="xAxis"
type='Column' xName='x' yName='y1'>
      </e-chart3d-series>
    </e-chart3d-series-collection>
  </ejs-chart3d>`
})
export class AppComponent {
  public dataSource?: Object[];
  public primaryXAxis?: Object;
  public primaryYAxis?: Object;
  public axes?: Object[];
  public columns?: Object[];
  public enableRotation?: boolean;
  ngOnInit(): void {
    this.dataSource = [
      { x: 'Jan', y: 15, y1: 33 }, { x: 'Feb', y: 20, y1: 31 }, { x:
'Mar', y: 35, y1: 30 },
      { x: 'Apr', y: 40, y1: 28 }, { x: 'May', y: 80, y1: 29 }, { x:
'Jun', y: 70, y1: 30 },
      { x: 'Jul', y: 65, y1: 33 }, { x: 'Aug', y: 55, y1: 32 }, { x:
'Sep', y: 50, y1: 34 },
      { x: 'Oct', y: 30, y1: 32 }, { x: 'Nov', y: 35, y1: 32 }, { x:
'Dec', y: 35, y1: 31 }
    ];
    this.enableRotation = true;
    this.primaryXAxis = {
      valueType: 'Category',
      interval: 1,
      // Span for chart axis
      span: 1
    };
  }
}
```

```

    this.primaryYAxis = {
      minimum: 0, maximum: 90, interval: 10,
      title: 'Temperature (Fahrenheit)',
      labelFormat: '{value}°F'
    };
    this.axes = [
      {
        majorGridLines: { width: 0 },
        columnIndex: 1,
        valueType: 'Category',
        name: 'xAxis'
      }
    ];
    this.columns = [
      {
        width: '50%'
      }, {
        width: '50%'
      }
    ];
  }
}

```

MAIN.TS

```

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

```

Chart Types

Column Chart in Angular 3D Chart control

Column chart

To render a column series, use series [type](#) as `Column` and inject `ColumnSeries3DService` module into the `@NgModule.providers`.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts';
import { Component } from '@angular/core';
@Component({
  imports: [
    Chart3DAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the Chart component
  template: `<ejs-chart3d style='display:block;' align='center'
[primaryXAxis]='primaryXAxis' [primaryYAxis]="primaryYAxis"
rotation=7 tilt=10 depth=100 [enableRotation]='enableRotation'>
<e-chart3d-series-collection>

```

```

    <e-chart3d-series [dataSource]='dataSource' type='Column'
    xName='country' yName='gold'>
    </e-chart3d-series>
  </e-chart3d-series-collection>
</ejs-chart3d>`
  })
  export class AppComponent {
    public dataSource?: Object[];
    public primaryXAxis?: Object;
    public enableRotation?: boolean;
    public primaryYAxis?: Object;
    ngOnInit(): void {
      this.dataSource = [
        { country: "USA", gold: 50, silver: 70, bronze: 45 },
        { country: "China", gold: 40, silver: 60, bronze: 55 },
        { country: "Japan", gold: 70, silver: 60, bronze: 50 },
        { country: "Australia", gold: 60, silver: 56, bronze: 40 },
        { country: "France", gold: 50, silver: 45, bronze: 35 },
        { country: "Germany", gold: 40, silver: 30, bronze: 22 },
        { country: "Italy", gold: 40, silver: 35, bronze: 37 },
        { country: "Sweden", gold: 30, silver: 25, bronze: 27 }
      ];
      this.primaryXAxis = {
        valueType: "Category"
      };
      this.enableRotation = true;
      this.primaryYAxis = { };
    }
  }
}

```

MAIN.TS

```

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

```

Column space and width

The [columnSpacing](#) and [columnWidth](#) properties are used to customize the space between columns.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts'
import { Component } from '@angular/core';
@Component({
  imports: [
    Chart3DAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the Chart component
  template: `<ejs-chart3d [primaryXAxis]='primaryXAxis'
[primaryYAxis]='primaryYAxis'
rotation=7 tilt=10 depth=100 [enableRotation]='enableRotation'>

```

```

    <e-chart3d-series-collection>
      <e-chart3d-series [dataSource]='dataSource' type='Column'
xName='country' yName='gold'>
      </e-chart3d-series>
      <e-chart3d-series [dataSource]='dataSource' type='Column'
xName='country' yName='silver' columnSpacing=0.75 columnWidth=0.5>
      </e-chart3d-series>
    </e-chart3d-series-collection>
  </ejs-chart3d>`
  })
  export class AppComponent {
    public dataSource?: Object[];
    public primaryXAxis?: Object;
    public enableRotation?: boolean;
    public primaryYAxis?: Object;
    ngOnInit(): void {
      this.dataSource = [
        { country: "USA", gold: 50, silver: 70, bronze: 45 },
        { country: "China", gold: 40, silver: 60, bronze: 55 },
        { country: "Japan", gold: 70, silver: 60, bronze: 50 },
        { country: "Australia", gold: 60, silver: 56, bronze: 40 },
        { country: "France", gold: 50, silver: 45, bronze: 35 },
        { country: "Germany", gold: 40, silver: 30, bronze: 22 },
        { country: "Italy", gold: 40, silver: 35, bronze: 37 },
        { country: "Sweden", gold: 30, silver: 25, bronze: 27 }
      ];
      this.primaryXAxis = {
        valueType: "Category"
      };
      this.enableRotation = true;
      this.primaryYAxis = { };
    }
  }
}

```

MAIN.TS

```

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

```

Grouped column

The data points can be grouped in the column type charts by using the [groupName](#) property. Data points with same group name are grouped together.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts'
import { Component } from '@angular/core';
@Component({
  imports: [
    Chart3DAllModule
  ],

```

```

standalone: true,
  selector: 'app-container',
  // specifies the template string for the Chart component
  template: `<ejs-chart3d [primaryXAxis]='primaryXAxis'
[primaryYAxis]="primaryYAxis"
  rotation=7 tilt=10 depth=100 [enableRotation]='enableRotation'>
    <e-chart3d-series-collection>
      <e-chart3d-series [dataSource]='dataSource' type='Column' xName='x'
yName='y' groupName="USA">
    </e-chart3d-series>
      <e-chart3d-series [dataSource]='dataSource' type='Column' xName='x'
yName='y1' groupName="USA">
    </e-chart3d-series>
      <e-chart3d-series [dataSource]='dataSource' type='Column' xName='x'
yName='y2' groupName="UK">
    </e-chart3d-series>
      <e-chart3d-series [dataSource]='dataSource' type='Column' xName='x'
yName='y3' groupName="UK">
    </e-chart3d-series>
    </e-chart3d-series-collection>
  </ejs-chart3d>`
})
export class AppComponent {
  public dataSource?: Object[];
  public primaryXAxis?: Object;
  public enableRotation?: boolean;
  public primaryYAxis?: Object;
  ngOnInit(): void {
    this.dataSource = [
      { x: '2012', y: 104, y1: 46, y2: 65, y3: 29 },
      { x: '2016', y: 121, y1: 46, y2: 67, y3: 27 },
      { x: '2020', y: 113, y1: 39, y2: 65, y3: 22 }
    ];
    this.primaryXAxis = {
      valueType: "Category"
    };
    this.enableRotation = true;
    this.primaryYAxis = { };
  }
}

```

MAIN.TS

```

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

```

Cylindrical column chart

To render a cylindrical column chart, set the [columnFacet](#) property to `Cylinder` in the chart series.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts'

```



```

import { Component } from '@angular/core';
@Component({
  imports: [
    Chart3DAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the Chart component
  template: `<ejs-chart3d [primaryXAxis]='primaryXAxis'
[primaryYAxis]="primaryYAxis"
rotation=7 tilt=10 depth=100 [enableRotation]='enableRotation'>
  <e-chart3d-series-collection>
    <e-chart3d-series [dataSource]='dataSource' type='Column' xName='x'
yName='y' columnFacet="Cylinder">
    </e-chart3d-series>
  </e-chart3d-series-collection>
</ejs-chart3d>`
})
export class AppComponent {
  public dataSource?: Object[];
  public primaryXAxis?: Object;
  public enableRotation?: boolean;
  public primaryYAxis?: Object;
  ngOnInit(): void {
    this.dataSource = [
      { x: 'Czechia', y: 1.11 },
      { x: 'Spain', y: 1.66 },
      { x: 'USA', y: 1.56 },
      { x: 'Germany', y: 3.1 },
      { x: 'Russia', y: 1.35 },
      { x: 'Slovakia', y: 1 },
      { x: 'South Korea', y: 3.16 },
      { x: 'France', y: 0.92 }
    ];
    this.primaryXAxis = {
      valueType: "Category"
    };
    this.enableRotation = true;
    this.primaryYAxis = { };
  }
}

```

MAIN.TS

```

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

```

Series customization

The following properties can be used to customize the **column** series.

- [fill](#) – Specifies the color of the series.
- [opacity](#) – Specifies the opacity of the [fill](#) color.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts'
import { Component } from '@angular/core';
@Component({
  imports: [
    Chart3DAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the Chart component
  template: `<ejs-chart3d [primaryXAxis]='primaryXAxis'
[primaryYAxis]="primaryYAxis"
rotation=7 tilt=10 depth=100 [enableRotation]='enableRotation'>
<e-chart3d-series-collection>
  <e-chart3d-series [dataSource]='dataSource' type='Column'
xName='country' yName='gold' fill="red">
    </e-chart3d-series>
  </e-chart3d-series-collection>
</ejs-chart3d>`
})
export class AppComponent {
  public dataSource?: Object[];
  public primaryXAxis?: Object;
  public enableRotation?: boolean;
  public primaryYAxis?: Object;
  ngOnInit(): void {
    this.dataSource = [
      { country: "USA", gold: 50, silver: 70, bronze: 45 },
      { country: "China", gold: 40, silver: 60, bronze: 55 },
      { country: "Japan", gold: 70, silver: 60, bronze: 50 },
      { country: "Australia", gold: 60, silver: 56, bronze: 40 },
      { country: "France", gold: 50, silver: 45, bronze: 35 },
      { country: "Germany", gold: 40, silver: 30, bronze: 22 },
      { country: "Italy", gold: 40, silver: 35, bronze: 37 },
      { country: "Sweden", gold: 30, silver: 25, bronze: 27 }
    ];
    this.primaryXAxis = {
      valueType: "Category"
    };
    this.enableRotation = true;
    this.primaryYAxis = { };
  }
}

```

MAIN.TS

```

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

```

Stacked column chart in Angular 3D Chart control

Stacked column chart

To render a stacked column series, use series [type](#) as `StackingColumn` and inject `StackingColumnSeries3DService` module into the `@NgModule.providers`.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts'
import { Component } from '@angular/core';
@Component({
  imports: [
    Chart3DAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the Chart component
  template: `<ejs-chart3d style='display:block;' align='center'
[primaryXAxis]='primaryXAxis'
rotation=7 tilt=10 depth=100 [enableRotation]='enableRotation'>
<e-chart3d-series-collection>
  <e-chart3d-series [dataSource]='dataSource' type='StackingColumn'
xName='x' yName='y' name="General Motors" >
  </e-chart3d-series>
  <e-chart3d-series [dataSource]='dataSource' type='StackingColumn'
xName='x' yName='y1' name="Honda">
  </e-chart3d-series>
  <e-chart3d-series [dataSource]='dataSource' type='StackingColumn'
xName='x' yName='y2' name="Suzuki">
  </e-chart3d-series>
</e-chart3d-series-collection>
</ejs-chart3d>`
})
export class AppComponent {
  public dataSource?: Object[];
  public primaryXAxis?: Object;
  public enableRotation?: boolean;
  ngOnInit(): void {
    this.dataSource = [
      { x: 2000, y: 0.61, y1: 0.03, y2: 0.48 }, { x: 2001, y: 0.81, y1:
0.05, y2: 0.53 },
      { x: 2002, y: 0.91, y1: 0.06, y2: 0.57 }, { x: 2003, y: 1, y1:
0.09, y2: 0.61 }, { x: 2004, y: 1.19, y1: 0.14, y2: 0.63 },
      { x: 2005, y: 1.47, y1: 0.20, y2: 0.64 }, { x: 2006, y: 1.74, y1:
0.29, y2: 0.66 }, { x: 2007, y: 1.98, y1: 0.46, y2: 0.76 },
      { x: 2008, y: 1.99, y1: 0.64, y2: 0.77 }, { x: 2009, y: 1.70, y1:
0.75, y2: 0.55 }
    ];
    this.primaryXAxis = {
      valueType: 'Category',
      labelRotation: -45,
      labelPlacement: 'BetweenTicks'
    };
    this.enableRotation = true;
  }
}
```

```
}

```

MAIN.TS

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

Stacking group

To group the stacked column, the [stackingGroup](#) property can be used. The columns with same group name are stacked on top of each other.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts'
import { Component } from '@angular/core';
@Component({
  imports: [
    Chart3DAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the Chart component
  template: `<ejs-chart3d style='display:block;' align='center'
[primaryXAxis]='primaryXAxis'
rotation=7 tilt=10 depth=100 [enableRotation]='enableRotation'>
<e-chart3d-series-collection>
  <e-chart3d-series [dataSource]='dataSource' type='StackingColumn'
xName='x' yName='y' name="General Motors" stackingGroup="A">
  </e-chart3d-series>
  <e-chart3d-series [dataSource]='dataSource' type='StackingColumn'
xName='x' yName='y1' name="Honda" stackingGroup="A">
  </e-chart3d-series>
  <e-chart3d-series [dataSource]='dataSource' type='StackingColumn'
xName='x' yName='y2' name="Suzuki" stackingGroup="B">
  </e-chart3d-series>
</e-chart3d-series-collection>
</ejs-chart3d>`
})
export class AppComponent {
  public dataSource?: Object[];
  public primaryXAxis?: Object;
  public enableRotation?: boolean;
  ngOnInit(): void {
    this.dataSource = [
      { x: 2000, y: 0.61, y1: 0.03, y2: 0.48 }, { x: 2001, y: 0.81, y1:
0.05, y2: 0.53 },
      { x: 2002, y: 0.91, y1: 0.06, y2: 0.57 }, { x: 2003, y: 1, y1:
0.09, y2: 0.61 }, { x: 2004, y: 1.19, y1: 0.14, y2: 0.63 },
      { x: 2005, y: 1.47, y1: 0.20, y2: 0.64 }, { x: 2006, y: 1.74, y1:
0.29, y2: 0.66 }, { x: 2007, y: 1.98, y1: 0.46, y2: 0.76 },
```

```

        { x: 2008, y: 1.99, y1: 0.64, y2: 0.77 }, { x: 2009, y: 1.70, y1:
0.75, y2: 0.55 }
    ];
    this.primaryXAxis = {
        valueType: 'Category',
        labelRotation: -45,
        labelPlacement: 'BetweenTicks'
    };
    this.enableRotation = true;
}
}

```

MAIN.TS

```

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

```

Cylindrical stacked column chart

To render a cylindrical stacked column chart, set the [columnFacet](#) property to **Cylinder** in the chart series.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts'
import { Component } from '@angular/core';
@Component({
  imports: [
    Chart3DAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the Chart component
  template: `<ejs-chart3d style='display:block;' align='center'
[primaryXAxis]='primaryXAxis'
rotation=7 tilt=10 depth=100 [enableRotation]='enableRotation'>
<e-chart3d-series-collection>
  <e-chart3d-series [dataSource]='dataSource' type='StackingColumn'
xName='x' yName='y' name="General Motors" columnFacet="Cylinder">
  </e-chart3d-series>
  <e-chart3d-series [dataSource]='dataSource' type='StackingColumn'
xName='x' yName='y1' name="Honda" columnFacet="Cylinder">
  </e-chart3d-series>
  <e-chart3d-series [dataSource]='dataSource' type='StackingColumn'
xName='x' yName='y2' name="Suzuki" columnFacet="Cylinder">
  </e-chart3d-series>
</e-chart3d-series-collection>
</ejs-chart3d>`
})
export class AppComponent {
  public dataSource?: Object[];
  public primaryXAxis?: Object;
}

```

```

public enableRotation?: boolean;
ngOnInit(): void {
    this.dataSource = [
        { x: 2000, y: 0.61, y1: 0.03, y2: 0.48 }, { x: 2001, y: 0.81, y1:
0.05, y2: 0.53 },
        { x: 2002, y: 0.91, y1: 0.06, y2: 0.57 }, { x: 2003, y: 1, y1:
0.09, y2: 0.61 }, { x: 2004, y: 1.19, y1: 0.14, y2: 0.63 },
        { x: 2005, y: 1.47, y1: 0.20, y2: 0.64 }, { x: 2006, y: 1.74, y1:
0.29, y2: 0.66 }, { x: 2007, y: 1.98, y1: 0.46, y2: 0.76 },
        { x: 2008, y: 1.99, y1: 0.64, y2: 0.77 }, { x: 2009, y: 1.70, y1:
0.75, y2: 0.55 }
    ];
    this.primaryXAxis = {
        valueType: 'Category',
        labelRotation: -45,
        labelPlacement: 'BetweenTicks'
    };
    this.enableRotation = true;
}
}

```

MAIN.TS

```

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

```

Series customization

The following properties can be used to customize the **stacked column** series.

- [fill](#) – Specifies the color of the series.
- [opacity](#) – Specifies the opacity of the [fill](#) color.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts';
import { Component } from '@angular/core';
@Component({
    imports: [
        Chart3DAllModule
    ],
    standalone: true,
    selector: 'app-container',
    // specifies the template string for the Chart component
    template: `<ejs-chart3d style='display:block;' align='center'
[primaryXAxis]='primaryXAxis'
rotation=7 tilt=10 depth=100 [enableRotation]='enableRotation'>
<e-chart3d-series-collection>
    <e-chart3d-series [dataSource]='dataSource' type='StackingColumn'
xName='x' yName='y' name="General Motors" fill="red">
    </e-chart3d-series>

```

```

    <e-chart3d-series [dataSource]='dataSource' type='StackingColumn'
    xName='x' yName='y1' name="Honda" fill="green">
    </e-chart3d-series>
    <e-chart3d-series [dataSource]='dataSource' type='StackingColumn'
    xName='x' yName='y2' name="Suzuki" fill="yellow">
    </e-chart3d-series>
  </e-chart3d-series-collection>
</ejs-chart3d>`
  })
  export class AppComponent {
    public dataSource?: Object[];
    public primaryXAxis?: Object;
    public enableRotation?: boolean;
    ngOnInit(): void {
      this.dataSource = [
        { x: 2000, y: 0.61, y1: 0.03, y2: 0.48 }, { x: 2001, y: 0.81, y1:
0.05, y2: 0.53 },
        { x: 2002, y: 0.91, y1: 0.06, y2: 0.57 }, { x: 2003, y: 1, y1:
0.09, y2: 0.61 }, { x: 2004, y: 1.19, y1: 0.14, y2: 0.63 },
        { x: 2005, y: 1.47, y1: 0.20, y2: 0.64 }, { x: 2006, y: 1.74, y1:
0.29, y2: 0.66 }, { x: 2007, y: 1.98, y1: 0.46, y2: 0.76 },
        { x: 2008, y: 1.99, y1: 0.64, y2: 0.77 }, { x: 2009, y: 1.70, y1:
0.75, y2: 0.55 }
      ];
      this.primaryXAxis = {
        valueType: 'Category',
        labelRotation: -45,
        labelPlacement: 'BetweenTicks'
      };
      this.enableRotation = true;
    }
  }
}

```

MAIN.TS

```

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

```

100% Stacked column chart in Angular 3D Chart control

100% Stacked column chart

To render a 100% stacked column series, use series [type](#) as `StackingColumn100` and inject `StackingColumnSeries3DService` module into the `@NgModule.providers`.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts'
import { Component } from '@angular/core';
@Component({
  imports: [
    Chart3DAllModule
  ],

```

```

standalone: true,
  selector: 'app-container',
  // specifies the template string for the Chart component
  template: `<ejs-chart3d style='display:block;' align='center'
[primaryXAxis]='primaryXAxis' [primaryYAxis]="primaryYAxis"
rotation=7 tilt=10 depth=100 [enableRotation]='enableRotation'>
    <e-chart3d-series-collection>
      <e-chart3d-series [dataSource]='dataSource' type='StackingColumn100'
xName='x' yName='y' name="General Motors">
      </e-chart3d-series>
      <e-chart3d-series [dataSource]='dataSource' type='StackingColumn100'
xName='x' yName='y1' name="Honda">
      </e-chart3d-series>
      <e-chart3d-series [dataSource]='dataSource' type='StackingColumn100'
xName='x' yName='y2' name="Suzuki">
      </e-chart3d-series>
    </e-chart3d-series-collection>
  </ejs-chart3d>`
  })
export class AppComponent {
  public dataSource?: Object[];
  public primaryXAxis?: Object;
  public primaryYAxis?: Object;
  public enableRotation?: boolean;
  ngOnInit(): void {
    this.dataSource = [
      { x: '2013', y: 9628912, y1: 4298390, y2: 2842133, y3: 2006366 },
      { x: '2014', y: 9609326, y1: 4513769, y2: 3016710, y3: 2165566 },
      { x: '2015', y: 7485587, y1: 4543838, y2: 3034081, y3: 2279503 },
      { x: '2016', y: 7793066, y1: 4999266, y2: 2945295, y3: 2359756 },
      { x: '2017', y: 6856880, y1: 5235842, y2: 3302336, y3: 2505741 }
    ];
    this.primaryXAxis = {
      valueType: 'Category',
      labelRotation: -45,
      labelPlacement: 'BetweenTicks'
    };
    this.primaryYAxis = {
      interval: 25
    };
    this.enableRotation = true;
  }
}

```

MAIN.TS

```

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

```

100% Cylindrical stacked column chart

To render a 100% cylindrical stacked column chart, set the [columnFacet](#) property to `Cylinder` in the chart series.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts'
import { Component } from '@angular/core';
@Component({
  imports: [
    Chart3DAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the Chart component
  template: `<ejs-chart3d style='display:block;' align='center'
[primaryXAxis]='primaryXAxis' [primaryYAxis]='primaryYAxis'
rotation=7 tilt=10 depth=100 [enableRotation]='enableRotation'>
<e-chart3d-series-collection>
  <e-chart3d-series [dataSource]='dataSource' type='StackingColumn100'
xName='x' yName='y' name="General Motors" columnFacet="Cylinder">
  </e-chart3d-series>
  <e-chart3d-series [dataSource]='dataSource' type='StackingColumn100'
xName='x' yName='y1' name="Honda" columnFacet="Cylinder">
  </e-chart3d-series>
  <e-chart3d-series [dataSource]='dataSource' type='StackingColumn100'
xName='x' yName='y2' name="Suzuki" columnFacet="Cylinder">
  </e-chart3d-series>
</e-chart3d-series-collection>
</ejs-chart3d>`
})
export class AppComponent {
  public dataSource?: Object[];
  public primaryXAxis?: Object;
  public primaryYAxis?: Object;
  public enableRotation?: boolean;
  ngOnInit(): void {
    this.dataSource = [
      { x: '2013', y: 9628912, y1: 4298390, y2: 2842133, y3: 2006366 },
      { x: '2014', y: 9609326, y1: 4513769, y2: 3016710, y3: 2165566 },
      { x: '2015', y: 7485587, y1: 4543838, y2: 3034081, y3: 2279503 },
      { x: '2016', y: 7793066, y1: 4999266, y2: 2945295, y3: 2359756 },
      { x: '2017', y: 6856880, y1: 5235842, y2: 3302336, y3: 2505741 }
    ];
    this.primaryXAxis = {
      valueType: 'Category',
      labelRotation: -45,
      labelPlacement: 'BetweenTicks'
    };
    this.primaryYAxis = {
      interval: 25
    };
    this.enableRotation = true;
  }
}

```

MAIN.TS

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

Series customization

The following properties can be used to customize the 100% stacked column series.

- [fill](#) – Specifies the color of the series.
- [opacity](#) – Specifies the opacity of the [fill](#) color.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts'
import { Component } from '@angular/core';
@Component({
  imports: [
    Chart3DAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the Chart component
  template: `<ejs-chart3d style='display:block;' align='center'
[primaryXAxis]='primaryXAxis' [primaryYAxis]='primaryYAxis'
rotation=7 tilt=10 depth=100 [enableRotation]='enableRotation'>
<e-chart3d-series-collection>
  <e-chart3d-series [dataSource]='dataSource' type='StackingColumn100'
xName='x' yName='y' name="General Motors" fill="red">
  </e-chart3d-series>
  <e-chart3d-series [dataSource]='dataSource' type='StackingColumn100'
xName='x' yName='y1' name="Honda" fill="green">
  </e-chart3d-series>
  <e-chart3d-series [dataSource]='dataSource' type='StackingColumn100'
xName='x' yName='y2' name="Suzuki" fill="yellow">
  </e-chart3d-series>
</e-chart3d-series-collection>
</ejs-chart3d>`
})
export class AppComponent {
  public dataSource?: Object[];
  public primaryXAxis?: Object;
  public primaryYAxis?: Object;
  public enableRotation?: boolean;
  ngOnInit(): void {
    this.dataSource = [
      { x: '2013', y: 9628912, y1: 4298390, y2: 2842133, y3: 2006366 },
      { x: '2014', y: 9609326, y1: 4513769, y2: 3016710, y3: 2165566 },
      { x: '2015', y: 7485587, y1: 4543838, y2: 3034081, y3: 2279503 },
      { x: '2016', y: 7793066, y1: 4999266, y2: 2945295, y3: 2359756 },
      { x: '2017', y: 6856880, y1: 5235842, y2: 3302336, y3: 2505741 }
    ];
    this.primaryXAxis = {
      valueType: 'Category',
```

```

        labelRotation: -45,
        labelPlacement: 'BetweenTicks'
    };
    this.primaryYAxis = {
        interval: 25
    };
    this.enableRotation = true;
}
}

```

MAIN.TS

```

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

```

Bar Chart in Angular 3D Chart control

Bar chart

To render a bar series, use series [type](#) as `Bar` and inject `BarSeries3DService` module into the `@NgModule.providers`.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts';
import { Component } from '@angular/core';
@Component({
  imports: [
    Chart3DAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the Chart component
  template: `<ejs-chart3d style='display:block;' align='center'
[primaryXAxis]='primaryXAxis' [primaryYAxis]="primaryYAxis"
rotation=7 tilt=10 depth=100 [enableRotation]='enableRotation'>
<e-chart3d-series-collection>
  <e-chart3d-series [dataSource]='dataSource' type='Bar'
xName='country' yName='gold'>
    </e-chart3d-series>
  </e-chart3d-series-collection>
</ejs-chart3d>`
})
export class AppComponent {
  public dataSource?: Object[];
  public primaryXAxis?: Object;
  public enableRotation?: boolean;
  public primaryYAxis?: Object;
  ngOnInit(): void {
    this.dataSource = [
      { country: "USA", gold: 50, silver: 70, bronze: 45 },
      { country: "China", gold: 40, silver: 60, bronze: 55 },
      { country: "Japan", gold: 70, silver: 60, bronze: 50 },
    ];
  }
}

```

```

        { country: "Australia", gold: 60, silver: 56, bronze: 40 },
        { country: "France", gold: 50, silver: 45, bronze: 35 },
        { country: "Germany", gold: 40, silver: 30, bronze: 22 },
        { country: "Italy", gold: 40, silver: 35, bronze: 37 },
        { country: "Sweden", gold: 30, silver: 25, bronze: 27 }
    ];
    this.primaryXAxis = {
        valueType: "Category"
    };
    this.enableRotation = true;
    this.primaryYAxis = { };
}
}

```

MAIN.TS

```

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

```

Bar space and width

The [columnSpacing](#) and [columnWidth](#) properties are used to customize the space between bars.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts'
import { Component } from '@angular/core';
@Component({
  imports: [
    Chart3DAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the Chart component
  template: `<ejs-chart3d style='display:block;' align='center'
[primaryXAxis]='primaryXAxis' [primaryYAxis]="primaryYAxis"
rotation=7 tilt=10 depth=100 [enableRotation]='enableRotation'>
<e-chart3d-series-collection>
  <e-chart3d-series [dataSource]='dataSource' type='Bar'
xName='country' yName='gold'>
  </e-chart3d-series>
  <e-chart3d-series [dataSource]='dataSource' type='Bar'
xName='country' yName='silver' columnSpacing=0.75 columnWidth=0.5>
  </e-chart3d-series>
</e-chart3d-series-collection>
</ejs-chart3d>`
})
export class AppComponent {
  public dataSource?: Object[];
  public primaryXAxis?: Object;
  public enableRotation?: boolean;
  public primaryYAxis?: Object;
  ngOnInit(): void {

```

```

    this.dataSource = [
      { country: "USA", gold: 50, silver: 70, bronze: 45 },
      { country: "China", gold: 40, silver: 60, bronze: 55 },
      { country: "Japan", gold: 70, silver: 60, bronze: 50 },
      { country: "Australia", gold: 60, silver: 56, bronze: 40 },
      { country: "France", gold: 50, silver: 45, bronze: 35 },
      { country: "Germany", gold: 40, silver: 30, bronze: 22 },
      { country: "Italy", gold: 40, silver: 35, bronze: 37 },
      { country: "Sweden", gold: 30, silver: 25, bronze: 27 }
    ];
    this.primaryXAxis = {
      valueType: "Category"
    };
    this.enableRotation = true;
    this.primaryYAxis = { };
  }
}

```

MAIN.TS

```

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

```

Grouped bar

The data points can be grouped in the bar type charts by using the [groupName](#) property. Data points with same group name are grouped together.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts'
import { Component } from '@angular/core';
@Component({
  imports: [
    Chart3DAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the Chart component
  template: `<ejs-chart3d style='display:block;' align='center'
[primaryXAxis]='primaryXAxis' [primaryYAxis]="primaryYAxis"
rotation=7 tilt=10 depth=100 [enableRotation]='enableRotation'>
<e-chart3d-series-collection>
  <e-chart3d-series [dataSource]='dataSource' type='Bar' xName='x'
yName='y' groupName="USA" columnSpacing=0.7 columnWidth=0.1>
</e-chart3d-series>
  <e-chart3d-series [dataSource]='dataSource' type='Bar' xName='x'
yName='y1' columnSpacing=0.7 columnWidth=0.1 groupName="USA">
</e-chart3d-series>
  <e-chart3d-series [dataSource]='dataSource' type='Bar' xName='x'
yName='y2' columnSpacing=0.7 columnWidth=0.5 groupName="UK">
</e-chart3d-series>

```

```

    <e-chart3d-series [dataSource]='dataSource' type='Bar' xName='x'
yName='y3' columnSpacing=0.7 columnWidth=0.1 groupName="UK">
    </e-chart3d-series>
  </e-chart3d-series-collection>
</ejs-chart3d>`
  })
  export class AppComponent {
    public dataSource?: Object[];
    public primaryXAxis?: Object;
    public enableRotation?: boolean;
    public primaryYAxis?: Object;
    ngOnInit(): void {
      this.dataSource = [
        { x: '2012', y: 104, y1: 46, y2: 65, y3: 29 },
        { x: '2016', y: 121, y1: 46, y2: 67, y3: 27 },
        { x: '2020', y: 113, y1: 39, y2: 65, y3: 22 }
      ];
      this.primaryXAxis = {
        valueType: "Category"
      };
      this.enableRotation = true;
      this.primaryYAxis = { };
    }
  }
}

```

MAIN.TS

```

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

```

Cylindrical bar chart

To render a cylindrical bar chart, set the [columnFacet](#) property to **Cylinder** in the chart series.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts'
import { Component } from '@angular/core';
@Component({
  imports: [
    Chart3DAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the Chart component
  template: `<ejs-chart3d style='display:block;' align='center'
[primaryXAxis]='primaryXAxis' [primaryYAxis]="primaryYAxis"
rotation=7 tilt=10 depth=100 [enableRotation]='enableRotation'>
    <e-chart3d-series-collection>
      <e-chart3d-series [dataSource]='dataSource' type='Bar'
xName='country' yName='gold' columnFacet="Cylinder">
        </e-chart3d-series>
      </e-chart3d-series-collection>
    </e-chart3d-series-collection>
  `
})

```

```

</ejs-chart3d>`
})
export class AppComponent {
  public dataSource?: Object[];
  public primaryXAxis?: Object;
  public enableRotation?: boolean;
  public primaryYAxis?: Object;
  ngOnInit(): void {
    this.dataSource = [
      { country: "USA", gold: 50, silver: 70, bronze: 45 },
      { country: "China", gold: 40, silver: 60, bronze: 55 },
      { country: "Japan", gold: 70, silver: 60, bronze: 50 },
      { country: "Australia", gold: 60, silver: 56, bronze: 40 },
      { country: "France", gold: 50, silver: 45, bronze: 35 },
      { country: "Germany", gold: 40, silver: 30, bronze: 22 },
      { country: "Italy", gold: 40, silver: 35, bronze: 37 },
      { country: "Sweden", gold: 30, silver: 25, bronze: 27 }
    ];
    this.primaryXAxis = {
      valueType: "Category"
    };
    this.enableRotation = true;
    this.primaryYAxis = { };
  }
}

```

MAIN.TS

```

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

```

Series customization

The following properties can be used to customize the **bar** series.

- [fill](#) – Specifies the color of the series.
- [opacity](#) – Specifies the opacity of the [fill](#) color.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts'
import { Component } from '@angular/core';
@Component({
  imports: [
    Chart3DAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the Chart component
  template: `<ejs-chart3d style='display:block;' align='center'
[primaryXAxis]='primaryXAxis' [primaryYAxis]='primaryYAxis'

```

```

rotation=7 tilt=10 depth=100 [enableRotation]='enableRotation'>
<e-chart3d-series-collection>
  <e-chart3d-series [dataSource]='dataSource' type='Bar' xName='x'
yName='y' fill="blue" opacity=0.5>
  </e-chart3d-series>
</e-chart3d-series-collection>
</ejs-chart3d>`
}))
export class AppComponent {
  public dataSource?: Object[];
  public primaryXAxis?: Object;
  public enableRotation?: boolean;
  public primaryYAxis?: Object;
  ngOnInit(): void {
    this.dataSource = [
      { x: 2005, y: 28 }, { x: 2006, y: 25 },
      { x: 2007, y: 26 }, { x: 2008, y: 27 },
      { x: 2009, y: 32 }, { x: 2010, y: 35 },
      { x: 2011, y: 30 }
    ];
    this.primaryXAxis = {
      valueType: "Category"
    };
    this.enableRotation = true;
    this.primaryYAxis = { };
  }
}

```

MAIN.TS

```

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

```

Stacked bar chart in Angular 3D Chart control

Stacked bar chart

To render a stacked bar series, use series `type` as `StackingBar` and inject `StackingBarSeries3DService` module into the `@NgModule.providers`.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts'
import { Component } from '@angular/core';
@Component({
  imports: [
    Chart3DAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the Chart component
  template: `<ejs-chart3d style='display:block;' align='center'
[primaryXAxis]='primaryXAxis'

```



```

rotation=7 tilt=10 depth=100 [enableRotation]='enableRotation'>
<e-chart3d-series-collection>
  <e-chart3d-series [dataSource]='dataSource' type='StackingBar'
xName='x' yName='y' name="America">
  </e-chart3d-series>
  <e-chart3d-series [dataSource]='dataSource' type='StackingBar'
xName='x' yName='y1' name="Canada">
  </e-chart3d-series>
  <e-chart3d-series [dataSource]='dataSource' type='StackingBar'
xName='x' yName='y2' name="France">
  </e-chart3d-series>
</e-chart3d-series-collection>
</ejs-chart3d>`
})
export class AppComponent {
  public dataSource?: Object[];
  public primaryXAxis?: Object;
  public enableRotation?: boolean;
  ngOnInit(): void {
    this.dataSource = [
      { x: 'Sochi', y: 9, y1: 10, y2: 4 },
      { x: 'Rio', y: 46, y1: 4, y2: 10 },
      { x: 'Pyeongchang', y: 9, y1: 11, y2: 5 },
      { x: 'Tokyo', y: 39, y1: 7, y2: 10 },
      { x: 'Beijing', y: 8, y1: 4, y2: 5 }
    ];
    this.primaryXAxis = {
      valueType: 'Category',
      labelRotation: -45,
      labelPlacement: 'BetweenTicks'
    };
    this.enableRotation = true;
  }
}

```

MAIN.TS

```

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

```

Stacking group

To group the stacked bar, the [stackingGroup](#) property can be used. The columns with same group name are stacked on top of each other.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts'
import { Component } from '@angular/core';
@Component({
  imports: [
    Chart3DAllModule

```

```

    ],
    standalone: true,
    selector: 'app-container',
    // specifies the template string for the Chart component
    template: `<ejs-chart3d style='display:block;' align='center'
[primaryXAxis]='primaryXAxis'
    rotation=7 tilt=10 depth=100 [enableRotation]='enableRotation'>
    <e-chart3d-series-collection>
      <e-chart3d-series [dataSource]='dataSource' type='StackingBar'
xName='x' yName='y' name="America" stackingGroup="JohnAndAndrew">
    </e-chart3d-series>
      <e-chart3d-series [dataSource]='dataSource' type='StackingBar'
xName='x' yName='y1' name="Canada" stackingGroup="JohnAndAndrew">
    </e-chart3d-series>
      <e-chart3d-series [dataSource]='dataSource' type='StackingBar'
xName='x' yName='y2' name="France" stackingGroup="ThomasAndMichael">
    </e-chart3d-series>
    </e-chart3d-series-collection>
  </ejs-chart3d>`
  })
  export class AppComponent {
    public dataSource?: Object[];
    public primaryXAxis?: Object;
    public enableRotation?: boolean;
    ngOnInit(): void {
      this.dataSource = [
        { x: 'Sochi', y: 9, y1: 10, y2: 4 },
        { x: 'Rio', y: 46, y1: 4, y2: 10 },
        { x: 'Pyeongchang', y: 9, y1: 11, y2: 5 },
        { x: 'Tokyo', y: 39, y1: 7, y2: 10 },
        { x: 'Beijing', y: 8, y1: 4, y2: 5 }
      ];
      this.primaryXAxis = {
        valueType: 'Category',
        labelRotation: -45,
        labelPlacement: 'BetweenTicks'
      };
      this.enableRotation = true;
    }
  }
}

```

MAIN.TS

```

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

```

Cylindrical stacked bar chart

To render a cylindrical stacked bar chart, set the [columnFacet](#) property to `Cylinder` in the chart series.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts'

```

```

import { Component } from '@angular/core';
@Component({
  imports: [
    Chart3DAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the Chart component
  template: `<ejs-chart3d style='display:block;' align='center'
[primaryXAxis]='primaryXAxis'
rotation=7 tilt=10 depth=100 [enableRotation]='enableRotation'>
  <e-chart3d-series-collection>
    <e-chart3d-series [dataSource]='dataSource' type='StackingBar'
xName='x' yName='y' name="America" columnFacet="Cylinder">
    </e-chart3d-series>
    <e-chart3d-series [dataSource]='dataSource' type='StackingBar'
xName='x' yName='y1' name="Canada" columnFacet="Cylinder">
    </e-chart3d-series>
    <e-chart3d-series [dataSource]='dataSource' type='StackingBar'
xName='x' yName='y2' name="France" columnFacet="Cylinder">
    </e-chart3d-series>
  </e-chart3d-series-collection>
</ejs-chart3d>`
})
export class AppComponent {
  public dataSource?: Object[];
  public primaryXAxis?: Object;
  public enableRotation?: boolean;
  ngOnInit(): void {
    this.dataSource = [
      { x: 'Sochi', y: 9, y1: 10, y2: 4 },
      { x: 'Rio', y: 46, y1: 4, y2: 10 },
      { x: 'Pyeongchang', y: 9, y1: 11, y2: 5 },
      { x: 'Tokyo', y: 39, y1: 7, y2: 10 },
      { x: 'Beijing', y: 8, y1: 4, y2: 5 }
    ];
    this.primaryXAxis = {
      valueType: 'Category',
      labelRotation: -45,
      labelPlacement: 'BetweenTicks'
    };
    this.enableRotation = true;
  }
}

```

MAIN.TS

```

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

```

Series customization

The following properties can be used to customize the **stacked bar** series.

- [fill](#) – Specifies the color of the series.
- [opacity](#) – Specifies the opacity of the [fill](#) color.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts'
import { Component } from '@angular/core';
@Component({
  imports: [
    Chart3DAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the Chart component
  template: `<ejs-chart3d style='display:block;' align='center'
[primaryXAxis]='primaryXAxis'
rotation=7 tilt=10 depth=100 [enableRotation]='enableRotation'>
<e-chart3d-series-collection>
  <e-chart3d-series [dataSource]='dataSource' type='StackingBar'
xName='x' yName='y' name="America" fill="red">
  </e-chart3d-series>
  <e-chart3d-series [dataSource]='dataSource' type='StackingBar'
xName='x' yName='y1' name="Canada" fill="green">
  </e-chart3d-series>
  <e-chart3d-series [dataSource]='dataSource' type='StackingBar'
xName='x' yName='y2' name="France" fill="yellow">
  </e-chart3d-series>
</e-chart3d-series-collection>
</ejs-chart3d>`
})
export class AppComponent {
  public dataSource?: Object[];
  public primaryXAxis?: Object;
  public enableRotation?: boolean;
  ngOnInit(): void {
    this.dataSource = [
      { x: 'Sochi', y: 9, y1: 10, y2: 4 },
      { x: 'Rio', y: 46, y1: 4, y2: 10 },
      { x: 'Pyeongchang', y: 9, y1: 11, y2: 5 },
      { x: 'Tokyo', y: 39, y1: 7, y2: 10 },
      { x: 'Beijing', y: 8, y1: 4, y2: 5 }
    ];
    this.primaryXAxis = {
      valueType: 'Category',
      labelRotation: -45,
      labelPlacement: 'BetweenTicks'
    };
    this.enableRotation = true;
  }
}
```

MAIN.TS

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

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

100% Stacked bar chart in Angular 3D Chart control

100% Stacked bar chart

To render a 100% stacked bar series, use series [type](#) as `StackingBar100` and inject `StackingBarSeries3DService` module into the `@NgModule.providers`.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts'
import { Component } from '@angular/core';
@Component({
  imports: [
    Chart3DAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the Chart component
  template: `<ejs-chart3d style='display:block;' align='center'
    [primaryXAxis]='primaryXAxis' [primaryYAxis]="primaryYAxis"
    rotation=7 tilt=10 depth=100 [enableRotation]='enableRotation'>
    <e-chart3d-series-collection>
      <e-chart3d-series [dataSource]='dataSource' type='StackingBar100'
        xName='x' yName='y' name="General Motors">
      </e-chart3d-series>
      <e-chart3d-series [dataSource]='dataSource' type='StackingBar100'
        xName='x' yName='y1' name="Honda">
      </e-chart3d-series>
      <e-chart3d-series [dataSource]='dataSource' type='StackingBar100'
        xName='x' yName='y2' name="Suzuki">
      </e-chart3d-series>
    </e-chart3d-series-collection>
  </ejs-chart3d>`
})
export class AppComponent {
  public dataSource?: Object[];
  public primaryXAxis?: Object;
  public primaryYAxis?: Object;
  public enableRotation?: boolean;
  ngOnInit(): void {
    this.dataSource = [
      { x: '2013', y: 9628912, y1: 4298390, y2: 2842133, y3: 2006366 },
      { x: '2014', y: 9609326, y1: 4513769, y2: 3016710, y3: 2165566 },
      { x: '2015', y: 7485587, y1: 4543838, y2: 3034081, y3: 2279503 },
      { x: '2016', y: 7793066, y1: 4999266, y2: 2945295, y3: 2359756 },
      { x: '2017', y: 6856880, y1: 5235842, y2: 3302336, y3: 2505741 }
    ];
    this.primaryXAxis = {
      valueType: 'Category',
      labelRotation: -45,
      labelPlacement: 'BetweenTicks'
    }
  }
}
```

```

    };
    this.primaryYAxis = {
      interval: 25
    };
    this.enableRotation = true;
  }
}

```

MAIN.TS

```

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

```

100% Cylindrical stacked bar chart

To render a cylindrical 100% stacked bar chart, set the [columnFacet](#) property to **Cylinder** in the chart series.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts';
import { Component } from '@angular/core';
@Component({
  imports: [
    Chart3DAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the Chart component
  template: `<ejs-chart3d style='display:block;' align='center'
[primaryXAxis]='primaryXAxis' [primaryYAxis]="primaryYAxis"
rotation=7 tilt=10 depth=100 [enableRotation]='enableRotation'>
<e-chart3d-series-collection>
  <e-chart3d-series [dataSource]='dataSource' type='StackingBar100'
xName='x' yName='y' name="General Motors" columnFacet="Cylinder">
  </e-chart3d-series>
  <e-chart3d-series [dataSource]='dataSource' type='StackingBar100'
xName='x' yName='y1' name="Honda" columnFacet="Cylinder">
  </e-chart3d-series>
  <e-chart3d-series [dataSource]='dataSource' type='StackingBar100'
xName='x' yName='y2' name="Suzuki" columnFacet="Cylinder">
  </e-chart3d-series>
</e-chart3d-series-collection>
</ejs-chart3d>`
})
export class AppComponent {
  public dataSource?: Object[];
  public primaryXAxis?: Object;
  public primaryYAxis?: Object;
  public enableRotation?: boolean;
  ngOnInit(): void {
    this.dataSource = [

```

```

        { x: '2013', y: 9628912, y1: 4298390, y2: 2842133, y3: 2006366 },
        { x: '2014', y: 9609326, y1: 4513769, y2: 3016710, y3: 2165566 },
        { x: '2015', y: 7485587, y1: 4543838, y2: 3034081, y3: 2279503 },
        { x: '2016', y: 7793066, y1: 4999266, y2: 2945295, y3: 2359756 },
        { x: '2017', y: 6856880, y1: 5235842, y2: 3302336, y3: 2505741 }
    ];
    this.primaryXAxis = {
        valueType: 'Category',
        labelRotation: -45,
        labelPlacement: 'BetweenTicks'
    };
    this.primaryYAxis = {
        interval: 25
    };
    this.enableRotation = true;
}
}

```

MAIN.TS

```

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

```

Series customization

The following properties can be used to customize the **100% stacked bar** series.

- [fill](#) – Specifies the color of the series.
- [opacity](#) – Specifies the opacity of the [fill](#) color.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts'
import { Component } from '@angular/core';
@Component({
  imports: [
    Chart3DAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the Chart component
  template: `<ejs-chart3d style='display:block;' align='center'
[primaryXAxis]='primaryXAxis' [primaryYAxis]="primaryYAxis"
rotation=7 tilt=10 depth=100 [enableRotation]='enableRotation'>
<e-chart3d-series-collection>
  <e-chart3d-series [dataSource]='dataSource' type='StackingBar100'
xName='x' yName='y' name="General Motors" fill="red">
</e-chart3d-series>
  <e-chart3d-series [dataSource]='dataSource' type='StackingBar100'
xName='x' yName='y1' name="Honda" fill="green">
</e-chart3d-series>

```

```

    <e-chart3d-series [dataSource]='dataSource' type='StackingBar100'
    xName='x' yName='y2' name="Suzuki" fill="yellow">
    </e-chart3d-series>
  </e-chart3d-series-collection>
</ejs-chart3d>`
}))
export class AppComponent {
  public dataSource?: Object[];
  public primaryXAxis?: Object;
  public primaryYAxis?: Object;
  public enableRotation?: boolean;
  ngOnInit(): void {
    this.dataSource = [
      { x: '2013', y: 9628912, y1: 4298390, y2: 2842133, y3: 2006366 },
      { x: '2014', y: 9609326, y1: 4513769, y2: 3016710, y3: 2165566 },
      { x: '2015', y: 7485587, y1: 4543838, y2: 3034081, y3: 2279503 },
      { x: '2016', y: 7793066, y1: 4999266, y2: 2945295, y3: 2359756 },
      { x: '2017', y: 6856880, y1: 5235842, y2: 3302336, y3: 2505741 }
    ];
    this.primaryXAxis = {
      valueType: 'Category',
      labelRotation: -45,
      labelPlacement: 'BetweenTicks'
    };
    this.primaryYAxis = {
      interval: 25
    };
    this.enableRotation = true;
  }
}

```

MAIN.TS

```

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

```

Data labels in Angular 3D Chart control

Data labels are fields that includes information about the sample point connected to an output. It can be added to a chart series by enabling the [visible](#) property in the [dataLabel](#). By default, the labels will arrange smartly without overlapping.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts';
import { Component } from '@angular/core';
@Component({
  imports: [
    Chart3DAllModule
  ],
  standalone: true,
  selector: 'app-container',
})

```



```
// specifies the template string for the Chart component
template: `<ejs-chart3d [primaryXAxis]='primaryXAxis'
[primaryYAxis]='primaryYAxis'
rotation=7 tilt=10 depth=100 [enableRotation]='enableRotation'>
  <e-chart3d-series-collection>
    <e-chart3d-series [dataSource]='dataSource' type='Column'
xName='country' yName='gold' [dataLabel]='dataLabel'>
    </e-chart3d-series>
  </e-chart3d-series-collection>
</ejs-chart3d>`
})
export class AppComponent {
  public dataSource?: Object[];
  public primaryXAxis?: Object;
  public enableRotation?: boolean;
  public primaryYAxis?: Object;
  public dataLabel?: Object;
  ngOnInit(): void {
    this.dataSource = [
      { country: "USA", gold: 50, silver: 70, bronze: 45 },
      { country: "China", gold: 40, silver: 60, bronze: 55 },
      { country: "Japan", gold: 70, silver: 60, bronze: 50 },
      { country: "Australia", gold: 60, silver: 56, bronze: 40 },
      { country: "France", gold: 50, silver: 45, bronze: 35 },
      { country: "Germany", gold: 40, silver: 30, bronze: 22 },
      { country: "Italy", gold: 40, silver: 35, bronze: 37 },
      { country: "Sweden", gold: 30, silver: 25, bronze: 27 }
    ];
    this.primaryXAxis = {
      valueType: "Category"
    };
    this.enableRotation = true;
    this.primaryYAxis = { };
    this.dataLabel = {
      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));
```

Note: To use data label feature, we need to inject `DataLabel3DService` module into the `@NgModule.providers`.

Position

The `position` property is used to place the label either on `Top`, `Middle`, or `Bottom`.

APP.COMPONENT.TS

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

```

import { BrowserModule } from '@angular/platform-browser'
import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts'
import { Component } from '@angular/core';
@Component({
  imports: [
    Chart3DAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the Chart component
  template: `<ejs-chart3d [primaryXAxis]='primaryXAxis'
[primaryYAxis]="primaryYAxis"
rotation=7 tilt=10 depth=100 [enableRotation]='enableRotation'>
<e-chart3d-series-collection>
  <e-chart3d-series [dataSource]='dataSource' type='Column'
xName='country' yName='gold' [dataLabel]="dataLabel">
    </e-chart3d-series>
  </e-chart3d-series-collection>
</ejs-chart3d>`
})
export class AppComponent {
  public dataSource?: Object[];
  public primaryXAxis?: Object;
  public enableRotation?: boolean;
  public primaryYAxis?: Object;
  public dataLabel?: Object;
  ngOnInit(): void {
    this.dataSource = [
      { country: "USA", gold: 50, silver: 70, bronze: 45 },
      { country: "China", gold: 40, silver: 60, bronze: 55 },
      { country: "Japan", gold: 70, silver: 60, bronze: 50 },
      { country: "Australia", gold: 60, silver: 56, bronze: 40 },
      { country: "France", gold: 50, silver: 45, bronze: 35 },
      { country: "Germany", gold: 40, silver: 30, bronze: 22 },
      { country: "Italy", gold: 40, silver: 35, bronze: 37 },
      { country: "Sweden", gold: 30, silver: 25, bronze: 27 }
    ];
    this.primaryXAxis = {
      valueType: "Category"
    };
    this.enableRotation = true;
    this.primaryYAxis = { };
    this.dataLabel = {
      visible: true,
      position: 'Top'
    }
  }
}

```

MAIN.TS

```

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

```

Template

Label content can be formatted by using the template option. Inside the template, the placeholder text `${point.x}` and `${point.y}` can be added to display corresponding data points x & y value. Using [template](#) property, the data label template can be set.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts'
import { Component } from '@angular/core';
@Component({
  imports: [
    Chart3DAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the Chart component
  template: `<ejs-chart3d [primaryXAxis]='primaryXAxis'
[primaryYAxis]="primaryYAxis"
rotation=7 tilt=10 depth=100 [enableRotation]='enableRotation'>
<e-chart3d-series-collection>
  <e-chart3d-series [dataSource]='dataSource' type='Column'
xName='country' yName='gold' [dataLabel]="dataLabel">
    </e-chart3d-series>
  </e-chart3d-series-collection>
</ejs-chart3d>`
})
export class AppComponent {
  public dataSource?: Object[];
  public primaryXAxis?: Object;
  public enableRotation?: boolean;
  public primaryYAxis?: Object;
  public dataLabel?: Object;
  ngOnInit(): void {
    this.dataSource = [
      { country: "USA", gold: 50, silver: 70, bronze: 45 },
      { country: "China", gold: 40, silver: 60, bronze: 55 },
      { country: "Japan", gold: 70, silver: 60, bronze: 50 },
      { country: "Australia", gold: 60, silver: 56, bronze: 40 },
      { country: "France", gold: 50, silver: 45, bronze: 35 },
      { country: "Germany", gold: 40, silver: 30, bronze: 22 },
      { country: "Italy", gold: 40, silver: 35, bronze: 37 },
      { country: "Sweden", gold: 30, silver: 25, bronze: 27 }
    ];
    this.primaryXAxis = {
      valueType: "Category"
    };
    this.enableRotation = true;
    this.primaryYAxis = { };
    this.dataLabel = {
      visible: true,
      template: '<div style="border: 1px solid black; padding: 3px 3px 3px 3px"><div>${point.x}</div><div>${point.y}</div></div>'
    }
  }
}
```

```
}

```

MAIN.TS

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

Text mapping

Text from the data source can be mapped using the [name](#) property.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts'
import { Component } from '@angular/core';
@Component({
  imports: [
    Chart3DAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the Chart component
  template: `<ejs-chart3d [primaryXAxis]='primaryXAxis'
[primaryYAxis]="primaryYAxis"
rotation=7 tilt=10 depth=100 [enableRotation]='enableRotation'>
<e-chart3d-series-collection>
  <e-chart3d-series [dataSource]='dataSource' type='Column' xName='x'
yName='y' [dataLabel]="dataLabel">
    </e-chart3d-series>
  </e-chart3d-series-collection>
</ejs-chart3d>`
})
export class AppComponent {
  public dataSource?: Object[];
  public primaryXAxis?: Object;
  public enableRotation?: boolean;
  public primaryYAxis?: Object;
  public dataLabel?: Object;
  ngOnInit(): void {
    this.dataSource = [
      { x: 'Jan', y: 12, text: 'January : 12°C' },
      { x: 'Feb', y: 8, text: 'February : 8°C' },
      { x: 'Mar', y: 11, text: 'March : 11°C' },
      { x: 'Apr', y: 6, text: 'April : 6°C' }
    ];
    this.primaryXAxis = {
      valueType: "Category"
    };
    this.enableRotation = true;
    this.primaryYAxis = { };
    this.dataLabel = {
      visible: true,
      name: "text"
    }
  }
}
```

```

    }
  }
}

```

MAIN.TS

```

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

```

Format

Data label for the chart can be formatted using the [format](#) property. The global formatting options can be used as 'n', 'p', and 'c'.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts';
import { Component } from '@angular/core';
@Component({
  imports: [
    Chart3DAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the Chart component
  template: `<ejs-chart3d [primaryXAxis]='primaryXAxis'
[primaryYAxis]="primaryYAxis"
rotation=7 tilt=10 depth=100 [enableRotation]='enableRotation'>
<e-chart3d-series-collection>
  <e-chart3d-series [dataSource]='dataSource' type='Column'
xName='country' yName='gold' [dataLabel]="dataLabel">
    </e-chart3d-series>
  </e-chart3d-series-collection>
</ejs-chart3d>`
})
export class AppComponent {
  public dataSource?: Object[];
  public primaryXAxis?: Object;
  public enableRotation?: boolean;
  public primaryYAxis?: Object;
  public dataLabel?: Object;
  ngOnInit(): void {
    this.dataSource = [
      { country: "USA", gold: 50, silver: 70, bronze: 45 },
      { country: "China", gold: 40, silver: 60, bronze: 55 },
      { country: "Japan", gold: 70, silver: 60, bronze: 50 },
      { country: "Australia", gold: 60, silver: 56, bronze: 40 }
    ];
    this.primaryXAxis = {
      valueType: "Category"
    };
    this.enableRotation = true;
  }
}

```

```

        this.primaryYAxis = { };
        this.dataLabel = {
            visible: true,
            format: 'n2'
        }
    }
}

```

MAIN.TS

```

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

```

Value	Format	Resultant Value	Description
1000	n1	1000.0	The number is rounded to 1 decimal place.
1000	n2	1000.00	The number is rounded to 2 decimal places.
1000	n3	1000.000	The number is rounded to 3 decimal place.
0.01	p1	1.0%	The number is converted to percentage with 1 decimal place.
0.01	p2	1.00%	The number is converted to percentage with 2 decimal place.
0.01	p3	1.000%	The number is converted to percentage with 3 decimal place.
1000	c1	\$1000.0	The currency symbol is appended to number and number is rounded to 1 decimal place.
1000	c2	\$1000.00	The currency symbol is appended to number and number is rounded to 2 decimal place.

Margin

The [margin](#) for data label can be applied by using [left](#), [right](#), [bottom](#) and [top](#) properties.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts';
import { Component } from '@angular/core';
@Component({
    imports: [
        Chart3DAllModule
    ],
    standalone: true,
    selector: 'app-container',
    // specifies the template string for the Chart component
    template: `<ejs-chart3d [primaryXAxis]='primaryXAxis'
[primaryYAxis]="primaryYAxis"
rotation=7 tilt=10 depth=100 [enableRotation]='enableRotation'>

```

```

    <e-chart3d-series-collection>
      <e-chart3d-series [dataSource]='dataSource' type='Column'
xName='country' yName='gold' [dataLabel]="dataLabel">
      </e-chart3d-series>
    </e-chart3d-series-collection>
  </ejs-chart3d>`
  })
  export class AppComponent {
    public dataSource?: Object[];
    public primaryXAxis?: Object;
    public enableRotation?: boolean;
    public primaryYAxis?: Object;
    public dataLabel?: Object;
    ngOnInit(): void {
      this.dataSource = [
        { country: "USA", gold: 50, silver: 70, bronze: 45 },
        { country: "China", gold: 40, silver: 60, bronze: 55 },
        { country: "Japan", gold: 70, silver: 60, bronze: 50 },
        { country: "Australia", gold: 60, silver: 56, bronze: 40 }
      ];
      this.primaryXAxis = {
        valueType: "Category"
      };
      this.enableRotation = true;
      this.primaryYAxis = { };
      this.dataLabel = {
        visible: true,
        border:{
          width: 1,
          color : 'red'
        },
        margin:{
          left:5,
          right:5,
          top:5,
          bottom:5
        }
      }
    }
  }
}

```

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 **stroke** and **border** of data label can be customized using [fill](#) and [border](#) properties.

APP.COMPONENT.TS

```

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

```

```

import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts'
import { Component } from '@angular/core';
@Component({
  imports: [
    Chart3DAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the Chart component
  template: `<ejs-chart3d [primaryXAxis]='primaryXAxis'
[primaryYAxis]="primaryYAxis"
rotation=7 tilt=10 depth=100 [enableRotation]='enableRotation'>
  <e-chart3d-series-collection>
    <e-chart3d-series [dataSource]='dataSource' type='Column'
xName='country' yName='gold' [dataLabel]="dataLabel">
      </e-chart3d-series>
    </e-chart3d-series-collection>
  </ejs-chart3d>`
})
export class AppComponent {
  public dataSource?: Object[];
  public primaryXAxis?: Object;
  public enableRotation?: boolean;
  public primaryYAxis?: Object;
  public dataLabel?: Object;
  ngOnInit(): void {
    this.dataSource = [
      { country: "USA", gold: 50, silver: 70, bronze: 45 },
      { country: "China", gold: 40, silver: 60, bronze: 55 },
      { country: "Japan", gold: 70, silver: 60, bronze: 50 },
      { country: "Australia", gold: 60, silver: 56, bronze: 40 }
    ];
    this.primaryXAxis = {
      valueType: "Category"
    };
    this.enableRotation = true;
    this.primaryYAxis = { };
    this.dataLabel = {
      visible: true,
      border:{ width: 2, color : 'red'}
    }
  }
}

```

MAIN.TS

```

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

```

Customizing specific label

A specific label can be customized by using the [textRender](#) event. The `textRender` event allows you to change the label text for the point.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts'
import { Chart3DTextRenderEventArgs } from '@syncfusion/ej2-angular-charts';
import { Component } from '@angular/core';
@Component({
  imports: [
    Chart3DAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the Chart component
  template: `<ejs-chart3d [primaryXAxis]='primaryXAxis'
(textRender)='textRender($event)' [primaryYAxis]='primaryYAxis'
rotation=7 tilt=10 depth=100 [enableRotation]='enableRotation'>
<e-chart3d-series-collection>
  <e-chart3d-series [dataSource]='dataSource' type='Column'
xName='country' yName='gold' [dataLabel]='dataLabel">
  </e-chart3d-series>
</e-chart3d-series-collection>
</ejs-chart3d>`
})
export class AppComponent {
  public dataSource?: Object[];
  public primaryXAxis?: Object;
  public enableRotation?: boolean;
  public primaryYAxis?: Object;
  public dataLabel?: Object;
  public textRender(args: Chart3DTextRenderEventArgs): void {
    if (args.point.index === 2) {
      args.text = 'Label';
    }
    else {
      args.cancel = true;
    }
  };
  ngOnInit(): void {
    this.dataSource = [
      { country: "USA", gold: 50, silver: 70, bronze: 45 },
      { country: "China", gold: 40, silver: 60, bronze: 55 },
      { country: "Japan", gold: 70, silver: 60, bronze: 50 },
      { country: "Australia", gold: 60, silver: 56, bronze: 40 }
    ];
    this.primaryXAxis = {
      valueType: "Category"
    };
    this.enableRotation = true;
    this.primaryYAxis = { };
    this.dataLabel = {
      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));
```

```
<!-- markdownlint-disable MD036 -->
```

Legend in Angular 3D Chart control

```
<!-- markdownlint-disable MD036 -->
```

Legend provides information about the series rendered in the 3D chart.

Position and alignment

By using the [position](#) property, the legend can be positioned at left, right, top or bottom of the 3D chart. The legend is positioned at the bottom of the 3D chart, by default.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts'
import { Component } from '@angular/core';
@Component({
  imports: [
    Chart3DAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the Chart component
  template: `<ejs-chart3d [primaryXAxis]='primaryXAxis'
[primaryYAxis]="primaryYAxis" [legendSettings]="legend"
rotation=7 tilt=10 depth=100 [enableRotation]='enableRotation'>
<e-chart3d-series-collection>
  <e-chart3d-series [dataSource]='dataSource' type='Column'
xName='country' yName='gold' name="Gold">
  </e-chart3d-series>
  <e-chart3d-series [dataSource]='dataSource' type='Column'
xName='country' yName='silver' name="Silver">
  </e-chart3d-series>
  <e-chart3d-series [dataSource]='dataSource' type='Column'
xName='country' yName='bronze' name="Bronze">
  </e-chart3d-series>
</e-chart3d-series-collection>
</ejs-chart3d>`
})
export class AppComponent {
  public dataSource?: Object[];
  public primaryXAxis?: Object;
  public enableRotation?: boolean;
  public legend?: Object;
  public primaryYAxis?: Object;
  ngOnInit(): void {
    this.dataSource = [
      { country: "USA", gold: 50, silver: 70, bronze: 45 },
      { country: "China", gold: 40, silver: 60, bronze: 55 },
```

```

        { country: "Japan", gold: 70, silver: 60, bronze: 50 },
        { country: "Australia", gold: 60, silver: 56, bronze: 40 },
        { country: "France", gold: 50, silver: 45, bronze: 35 },
        { country: "Germany", gold: 40, silver: 30, bronze: 22 },
        { country: "Italy", gold: 40, silver: 35, bronze: 37 },
        { country: "Sweden", gold: 30, silver: 25, bronze: 27 }
    ];
    this.primaryXAxis = {
        valueType: 'Category',
        title: 'Countries'
    };
    this.enableRotation = true;
    this.primaryYAxis = {
        minimum: 0, maximum: 80,
        interval: 20, title: 'Medals'
    };
    this.legend = {
        visible: true,
        //Legend position as top
        position: 'Top'
    };
}
}

```

MAIN.TS

```

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

```

The custom position helps you to position the legend anywhere in the 3D chart using x and y coordinates.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts';
import { Component } from '@angular/core';
@Component({
    imports: [
        Chart3DAllModule
    ],
    standalone: true,
    selector: 'app-container',
    // specifies the template string for the Chart component
    template: `<ejs-chart3d [primaryXAxis]='primaryXAxis'
[primaryYAxis]="primaryYAxis" [legendSettings]="legend"
rotation=7 tilt=10 depth=100 [enableRotation]='enableRotation'>
<e-chart3d-series-collection>
    <e-chart3d-series [dataSource]='dataSource' type='Column'
xName='country' yName='gold' name="Gold">
    </e-chart3d-series>
    <e-chart3d-series [dataSource]='dataSource' type='Column'
xName='country' yName='silver' name="Silver">

```

```

        </e-chart3d-series>
        <e-chart3d-series [dataSource]='dataSource' type='Column'
        xName='country' yName='bronze' name="Bronze">
        </e-chart3d-series>
    </e-chart3d-series-collection>
</ejs-chart3d>`
    })
    export class AppComponent {
        public dataSource?: Object[];
        public primaryXAxis?: Object;
        public enableRotation?: boolean;
        public legend?: Object;
        public primaryYAxis?: Object;
        ngOnInit(): void {
            this.dataSource = [
                { country: "USA", gold: 50, silver: 70, bronze: 45 },
                { country: "China", gold: 40, silver: 60, bronze: 55 },
                { country: "Japan", gold: 70, silver: 60, bronze: 50 },
                { country: "Australia", gold: 60, silver: 56, bronze: 40 },
                { country: "France", gold: 50, silver: 45, bronze: 35 },
                { country: "Germany", gold: 40, silver: 30, bronze: 22 },
                { country: "Italy", gold: 40, silver: 35, bronze: 37 },
                { country: "Sweden", gold: 30, silver: 25, bronze: 27 }
            ];
            this.primaryXAxis = {
                valueType: 'Category',
                title: 'Countries'
            };
            this.enableRotation = true;
            this.primaryYAxis = {
                minimum: 0, maximum: 80,
                interval: 20, title: 'Medals'
            };
            this.legend = {
                visible: true,
                position: 'Custom',
                location: { x: 200, y: 20 }
            };
        }
    }
}

```

MAIN.TS

```

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

```

Legend reverse

The order of the legend items can be reversed by using the [reverse](#) property. By default, legend for the first series in the collection will be placed first.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'

```

```

import { BrowserModule } from '@angular/platform-browser'
import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts'
import { Component } from '@angular/core';
@Component({
  imports: [
    Chart3DAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the Chart component
  template: `<ejs-chart3d [primaryXAxis]='primaryXAxis'
[primaryYAxis]="primaryYAxis" [legendSettings]="legend"
rotation=7 tilt=10 depth=100 [enableRotation]='enableRotation'>
<e-chart3d-series-collection>
  <e-chart3d-series [dataSource]='dataSource' type='Column'
xName='country' yName='gold' name="Gold">
  </e-chart3d-series>
  <e-chart3d-series [dataSource]='dataSource' type='Column'
xName='country' yName='silver' name="Silver">
  </e-chart3d-series>
  <e-chart3d-series [dataSource]='dataSource' type='Column'
xName='country' yName='bronze' name="Bronze">
  </e-chart3d-series>
</e-chart3d-series-collection>
</ejs-chart3d>`
})
export class AppComponent {
  public dataSource?: Object[];
  public primaryXAxis?: Object;
  public enableRotation?: boolean;
  public legend?: Object;
  public primaryYAxis?: Object;
  ngOnInit(): void {
    this.dataSource = [
      { country: "USA", gold: 50, silver: 70, bronze: 45 },
      { country: "China", gold: 40, silver: 60, bronze: 55 },
      { country: "Japan", gold: 70, silver: 60, bronze: 50 },
      { country: "Australia", gold: 60, silver: 56, bronze: 40 },
      { country: "France", gold: 50, silver: 45, bronze: 35 },
      { country: "Germany", gold: 40, silver: 30, bronze: 22 },
      { country: "Italy", gold: 40, silver: 35, bronze: 37 },
      { country: "Sweden", gold: 30, silver: 25, bronze: 27 }
    ];
    this.primaryXAxis = {
      valueType: 'Category',
      title: 'Countries'
    };
    this.enableRotation = true;
    this.primaryYAxis = {
      minimum: 0, maximum: 80,
      interval: 20, title: 'Medals'
    };
    this.legend = {
      visible: true,
      reverse: true
    };
  }
}

```

```
}

```

MAIN.TS

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

Legend alignment

The legend can be aligned at near, far or center to the 3D chart using the [alignment](#) property.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts'
import { Component } from '@angular/core';
@Component({
  imports: [
    Chart3DAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the Chart component
  template: `<ejs-chart3d [primaryXAxis]='primaryXAxis'
[primaryYAxis]="primaryYAxis" [legendSettings]="legend"
rotation=7 tilt=10 depth=100 [enableRotation]='enableRotation'>
  <e-chart3d-series-collection>
    <e-chart3d-series [dataSource]='dataSource' type='Column'
xName='country' yName='gold' name="Gold">
    </e-chart3d-series>
    <e-chart3d-series [dataSource]='dataSource' type='Column'
xName='country' yName='silver' name="Silver">
    </e-chart3d-series>
    <e-chart3d-series [dataSource]='dataSource' type='Column'
xName='country' yName='bronze' name="Bronze">
    </e-chart3d-series>
  </e-chart3d-series-collection>
</ejs-chart3d>`
})
export class AppComponent {
  public dataSource?: Object[];
  public primaryXAxis?: Object;
  public enableRotation?: boolean;
  public legend?: Object;
  public primaryYAxis?: Object;
  ngOnInit(): void {
    this.dataSource = [
      { country: "USA", gold: 50, silver: 70, bronze: 45 },
      { country: "China", gold: 40, silver: 60, bronze: 55 },
      { country: "Japan", gold: 70, silver: 60, bronze: 50 },
      { country: "Australia", gold: 60, silver: 56, bronze: 40 },
      { country: "France", gold: 50, silver: 45, bronze: 35 },
      { country: "Germany", gold: 40, silver: 30, bronze: 22 },
      { country: "Italy", gold: 40, silver: 35, bronze: 37 },
```

```

        { country: "Sweden", gold: 30, silver: 25, bronze: 27 }
    ];
    this.primaryXAxis = {
        valueType: 'Category',
        title: 'Countries'
    };
    this.enableRotation = true;
    this.primaryYAxis = {
        minimum: 0, maximum: 80,
        interval: 20, title: 'Medals'
    };
    this.legend = {
        visible: true,
        position: 'Top',
        alignment: 'Near'
    };
}
}

```

MAIN.TS

```

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

```

Legend customization

To change the legend icon shape, [legendShape](#) property in the [series](#) can be used. By default, the legend icon shape is `seriesType`.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts'
import { Component } from '@angular/core';
@Component({
  imports: [
    Chart3DAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the Chart component
  template: `<ejs-chart3d [primaryXAxis]='primaryXAxis'
[primaryYAxis]="primaryYAxis" [legendSettings]="legend"
rotation=7 tilt=10 depth=100 [enableRotation]='enableRotation'>
<e-chart3d-series-collection>
    <e-chart3d-series [dataSource]='dataSource' type='Column'
xName='country' yName='gold' name="Gold" legendShape="Circle">
    </e-chart3d-series>
    <e-chart3d-series [dataSource]='dataSource' type='Column'
xName='country' yName='silver' name="Silver" legendShape="SeriesType">
    </e-chart3d-series>
    <e-chart3d-series [dataSource]='dataSource' type='Column'
xName='country' yName='bronze' name="Bronze" legendShape="Rectangle">

```

```

        </e-chart3d-series>
    </e-chart3d-series-collection>
</ejs-chart3d>`
    })
    export class AppComponent {
        public dataSource?: Object[];
        public primaryXAxis?: Object;
        public enableRotation?: boolean;
        public legend?: Object;
        public primaryYAxis?: Object;
        ngOnInit(): void {
            this.dataSource = [
                { country: "USA", gold: 50, silver: 70, bronze: 45 },
                { country: "China", gold: 40, silver: 60, bronze: 55 },
                { country: "Japan", gold: 70, silver: 60, bronze: 50 },
                { country: "Australia", gold: 60, silver: 56, bronze: 40 },
                { country: "France", gold: 50, silver: 45, bronze: 35 },
                { country: "Germany", gold: 40, silver: 30, bronze: 22 },
                { country: "Italy", gold: 40, silver: 35, bronze: 37 },
                { country: "Sweden", gold: 30, silver: 25, bronze: 27 }
            ];
            this.primaryXAxis = {
                valueType: 'Category',
                title: 'Countries'
            };
            this.enableRotation = true;
            this.primaryYAxis = {
                minimum: 0, maximum: 80,
                interval: 20, title: 'Medals'
            };
            this.legend = {
                visible: true
            };
        }
    }
}

```

MAIN.TS

```

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

```

Legend size

By default, legend takes 20% - 25% of the 3D chart's height horizontally, when it is placed on top or bottom position and 20% - 25% of the 3D chart's width vertically, when it is placed on left or right position. You can change this default legend size by using the [height](#) and [width](#) properties of the [legendSettings](#).

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts'
import { Component } from '@angular/core';

```



```

@Component({
  imports: [
    Chart3DAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the Chart component
  template: `<ejs-chart3d [primaryXAxis]='primaryXAxis'
[primaryYAxis]="primaryYAxis" [legendSettings]="legend"
rotation=7 tilt=10 depth=100 [enableRotation]='enableRotation'>
  <e-chart3d-series-collection>
    <e-chart3d-series [dataSource]='dataSource' type='Column'
xName='country' yName='gold' name="Gold" legendShape="Circle">
    </e-chart3d-series>
    <e-chart3d-series [dataSource]='dataSource' type='Column'
xName='country' yName='silver' name="Silver" legendShape="Circle">
    </e-chart3d-series>
    <e-chart3d-series [dataSource]='dataSource' type='Column'
xName='country' yName='bronze' name="Bronze" legendShape="Circle">
    </e-chart3d-series>
  </e-chart3d-series-collection>
</ejs-chart3d>`
})
export class AppComponent {
  public dataSource?: Object[];
  public primaryXAxis?: Object;
  public enableRotation?: boolean;
  public legend?: Object;
  public primaryYAxis?: Object;
  ngOnInit(): void {
    this.dataSource = [
      { country: "USA", gold: 50, silver: 70, bronze: 45 },
      { country: "China", gold: 40, silver: 60, bronze: 55 },
      { country: "Japan", gold: 70, silver: 60, bronze: 50 },
      { country: "Australia", gold: 60, silver: 56, bronze: 40 },
      { country: "France", gold: 50, silver: 45, bronze: 35 },
      { country: "Germany", gold: 40, silver: 30, bronze: 22 },
      { country: "Italy", gold: 40, silver: 35, bronze: 37 },
      { country: "Sweden", gold: 30, silver: 25, bronze: 27 }
    ];
    this.primaryXAxis = {
      valueType: 'Category',
      title: 'Countries'
    };
    this.enableRotation = true;
    this.primaryYAxis = {
      minimum: 0, maximum: 80,
      interval: 20, title: 'Medals'
    };
    this.legend = {
      visible: true,
      width: '500', height: '100',
      border: { width: 1, color: 'pink' }
    };
  }
}

```

MAIN.TS

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

Legend item size

The size of the legend items can be customised by using the [shapeHeight](#) and [shapeWidth](#) properties.\

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts'
import { Component } from '@angular/core';
@Component({
  imports: [
    Chart3DAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the Chart component
  template: `<ejs-chart3d [primaryXAxis]='primaryXAxis'
[primaryYAxis]="primaryYAxis" [legendSettings]="legend"
rotation=7 tilt=10 depth=100 [enableRotation]='enableRotation'>
<e-chart3d-series-collection>
  <e-chart3d-series [dataSource]='dataSource' type='Column'
xName='country' yName='gold' name="Gold" legendShape="Circle">
  </e-chart3d-series>
  <e-chart3d-series [dataSource]='dataSource' type='Column'
xName='country' yName='silver' name="Silver" legendShape="Circle">
  </e-chart3d-series>
  <e-chart3d-series [dataSource]='dataSource' type='Column'
xName='country' yName='bronze' name="Bronze" legendShape="Circle">
  </e-chart3d-series>
</e-chart3d-series-collection>
</ejs-chart3d>`
})
export class AppComponent {
  public dataSource?: Object[];
  public primaryXAxis?: Object;
  public enableRotation?: boolean;
  public legend?: Object;
  public primaryYAxis?: Object;
  ngOnInit(): void {
    this.dataSource = [
      { country: "USA", gold: 50, silver: 70, bronze: 45 },
      { country: "China", gold: 40, silver: 60, bronze: 55 },
      { country: "Japan", gold: 70, silver: 60, bronze: 50 },
      { country: "Australia", gold: 60, silver: 56, bronze: 40 },
      { country: "France", gold: 50, silver: 45, bronze: 35 },
      { country: "Germany", gold: 40, silver: 30, bronze: 22 },
      { country: "Italy", gold: 40, silver: 35, bronze: 37 },
      { country: "Sweden", gold: 30, silver: 25, bronze: 27 }
```

```

    ];
    this.primaryXAxis = {
      valueType: 'Category',
      title: 'Countries'
    };
    this.enableRotation = true;
    this.primaryYAxis = {
      minimum: 0, maximum: 80,
      interval: 20, title: 'Medals'
    };
    this.legend = {
      visible: true,
      shapeHeight: 10, shapeWidth: 10
    };
  }
}

```

MAIN.TS

```

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

```

Paging for legend

Paging will be enabled by default, when the legend items exceeds the legend bounds. Each legend items can be viewed by navigating between the pages using navigation buttons.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts'
import { Component } from '@angular/core';
@Component({
  imports: [
    Chart3DAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the Chart component
  template: `<ejs-chart3d [primaryXAxis]='primaryXAxis'
[primaryYAxis]="primaryYAxis" [legendSettings]="legend"
rotation=7 tilt=10 depth=100 [enableRotation]='enableRotation'>
<e-chart3d-series-collection>
  <e-chart3d-series [dataSource]='dataSource' type='Column' xName='x'
yName='y' name="December 2007">
  </e-chart3d-series>
  <e-chart3d-series [dataSource]='dataSource' type='Column' xName='x'
yName='y1' name="December 2008" >
  </e-chart3d-series>
  <e-chart3d-series [dataSource]='dataSource' type='Column' xName='x'
yName='y2' name="December 2009">
  </e-chart3d-series>

```

```

    <e-chart3d-series [dataSource]='dataSource' type='Column' xName='x'
yName='y3' name="December 2010">
    </e-chart3d-series>
  </e-chart3d-series-collection>
</ejs-chart3d>`
  })
  export class AppComponent {
    public dataSource?: Object[];
    public primaryXAxis?: Object;
    public enableRotation?: boolean;
    public legend?: Object;
    public primaryYAxis?: Object;
    ngOnInit(): void {
      this.dataSource = [
        { x: 'WW', y: 12, y1: 22, y2: 38.3, y3: 50 },
        { x: 'EU', y: 9.9, y1: 26, y2: 45.2, y3: 63.6 },
        { x: 'APAC', y: 4.4, y1: 9.3, y2: 18.2, y3: 20.9 },
        { x: 'LATAM', y: 6.4, y1: 28, y2: 46.7, y3: 65.1 },
        { x: 'MEA', y: 30, y1: 45.7, y2: 61.5, y3: 73 },
        { x: 'NA', y: 25.3, y1: 35.9, y2: 64, y3: 81.4 }
      ];
      this.primaryXAxis = {
        valueType: 'Category',
        title: 'Countries'
      };
      this.enableRotation = true;
      this.primaryYAxis = {
        minimum: 0, maximum: 80,
        interval: 20, title: 'Medals'
      };
      this.legend = {
        padding: 10, shapePadding: 10,
        visible: true, border: {
          width: 2, color: 'grey'
        },
        width: '200'
      };
    }
  }
}

```

MAIN.TS

```

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

```

Legend text wrap

When the legend text exceeds the container, the text can be wrapped by using the [textWrap](#) property. End user can also wrap the legend text based on the [maximumLabelWidth](#) property.

APP.COMPONENT.TS

```

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

```

```

import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts'
import { Component } from '@angular/core';
@Component({
  imports: [
    Chart3DAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the Chart component
  template: `<ejs-chart3d [primaryXAxis]='primaryXAxis'
[primaryYAxis]="primaryYAxis" [legendSettings]="legend"
rotation=7 tilt=10 depth=100 [enableRotation]='enableRotation'>
  <e-chart3d-series-collection>
    <e-chart3d-series [dataSource]='dataSource' type='Column'
xName='country' yName='gold' name="Gold">
    </e-chart3d-series>
    <e-chart3d-series [dataSource]='dataSource' type='Column'
xName='country' yName='silver' name="Silver" >
    </e-chart3d-series>
    <e-chart3d-series [dataSource]='dataSource' type='Column'
xName='country' yName='bronze' name="Bronze">
    </e-chart3d-series>
  </e-chart3d-series-collection>
</ejs-chart3d>`
})
export class AppComponent {
  public dataSource?: Object[];
  public primaryXAxis?: Object;
  public enableRotation?: boolean;
  public legend?: Object;
  public primaryYAxis?: Object;
  ngOnInit(): void {
    this.dataSource = [
      { country: "USA", gold: 50, silver: 70, bronze: 45 },
      { country: "China", gold: 40, silver: 60, bronze: 55 },
      { country: "Japan", gold: 70, silver: 60, bronze: 50 },
      { country: "Australia", gold: 60, silver: 56, bronze: 40 },
      { country: "France", gold: 50, silver: 45, bronze: 35 },
      { country: "Germany", gold: 40, silver: 30, bronze: 22 },
      { country: "Italy", gold: 40, silver: 35, bronze: 37 },
      { country: "Sweden", gold: 30, silver: 25, bronze: 27 }
    ];
    this.primaryXAxis = {
      valueType: 'Category',
      title: 'Countries'
    };
    this.enableRotation = true;
    this.primaryYAxis = {
      minimum: 0, maximum: 80,
      interval: 20, title: 'Medals'
    };
    this.legend = {
      visible: true,
      position: 'Right',
      textWrap: 'Wrap',
      maximumLabelWidth: 50,
    };
  }
}

```

```
}
}
```

MAIN.TS

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

Series selection through legend

By default, you can collapse the series visibility by clicking the legend. On the other hand, turn off the [toggleVisibility](#) property if you must use a legend click to choose a series.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts'
import { Component } from '@angular/core';
@Component({
  imports: [
    Chart3DAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the Chart component
  template: `<ejs-chart3d [primaryXAxis]='primaryXAxis'
[primaryYAxis]="primaryYAxis" [legendSettings]="legend"
rotation=7 tilt=10 depth=100 [enableRotation]='enableRotation'>
<e-chart3d-series-collection>
  <e-chart3d-series [dataSource]='dataSource' type='Column'
xName='country' yName='gold' name="Gold">
    </e-chart3d-series>
  <e-chart3d-series [dataSource]='dataSource' type='Column'
xName='country' yName='silver' name="Silver" >
    </e-chart3d-series>
  <e-chart3d-series [dataSource]='dataSource' type='Column'
xName='country' yName='bronze' name="Bronze">
    </e-chart3d-series>
</e-chart3d-series-collection>
</ejs-chart3d>`
})
export class AppComponent {
  public dataSource?: Object[];
  public primaryXAxis?: Object;
  public enableRotation?: boolean;
  public legend?: Object;
  public primaryYAxis?: Object;
  ngOnInit(): void {
    this.dataSource = [
      { country: "USA", gold: 50, silver: 70, bronze: 45 },
      { country: "China", gold: 40, silver: 60, bronze: 55 },
      { country: "Japan", gold: 70, silver: 60, bronze: 50 },
      { country: "Australia", gold: 60, silver: 56, bronze: 40 },
```

```

        { country: "France", gold: 50, silver: 45, bronze: 35 },
        { country: "Germany", gold: 40, silver: 30, bronze: 22 },
        { country: "Italy", gold: 40, silver: 35, bronze: 37 },
        { country: "Sweden", gold: 30, silver: 25, bronze: 27 }
    ];
    this.primaryXAxis = {
        valueType: 'Category',
        title: 'Countries'
    };
    this.enableRotation = true;
    this.primaryYAxis = {
        minimum: 0, maximum: 80,
        interval: 20, title: 'Medals'
    };
    this.legend = {
        visible: true,
        toggleVisibility: false,
        enableHighlight: true
    };
}
}

```

MAIN.TS

```

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

```

Collapsing legend item

By default, series name will be displayed as legend. To skip the legend for a particular series, you can give empty string to the series name.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts';
import { Component } from '@angular/core';
@Component({
    imports: [
        Chart3DAllModule
    ],
    standalone: true,
    selector: 'app-container',
    // specifies the template string for the Chart component
    template: `<ejs-chart3d [primaryXAxis]='primaryXAxis'
[primaryYAxis]="primaryYAxis" [legendSettings]="legend"
rotation=7 tilt=10 depth=100 [enableRotation]='enableRotation'>
<e-chart3d-series-collection>
    <e-chart3d-series [dataSource]='dataSource' type='Column'
xName='country' yName='gold' name="Gold">
    </e-chart3d-series>
    <e-chart3d-series [dataSource]='dataSource' type='Column'
xName='country' yName='silver' name="Silver" >

```

```

        </e-chart3d-series>
        <e-chart3d-series [dataSource]='dataSource' type='Column'
        xName='country' yName='bronze' name="Bronze">
        </e-chart3d-series>
    </e-chart3d-series-collection>
</ejs-chart3d>`
    })
    export class AppComponent {
        public dataSource?: Object[];
        public primaryXAxis?: Object;
        public enableRotation?: boolean;
        public legend?: Object;
        public primaryYAxis?: Object;
        ngOnInit(): void {
            this.dataSource = [
                { country: "USA", gold: 50, silver: 70, bronze: 45 },
                { country: "China", gold: 40, silver: 60, bronze: 55 },
                { country: "Japan", gold: 70, silver: 60, bronze: 50 },
                { country: "Australia", gold: 60, silver: 56, bronze: 40 },
                { country: "France", gold: 50, silver: 45, bronze: 35 },
                { country: "Germany", gold: 40, silver: 30, bronze: 22 },
                { country: "Italy", gold: 40, silver: 35, bronze: 37 },
                { country: "Sweden", gold: 30, silver: 25, bronze: 27 }
            ];
            this.primaryXAxis = {
                valueType: 'Category',
                title: 'Countries'
            };
            this.enableRotation = true;
            this.primaryYAxis = {
                minimum: 0, maximum: 80,
                interval: 20, title: 'Medals'
            };
            this.legend = {
                visible: true,
                toggleVisibility: true
            };
        }
    }
}

```

MAIN.TS

```

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

```

Legend title

You can set title for legend using [title](#) property in [legendSettings](#). The [size](#), [color](#), [opacity](#), [fontStyle](#), [fontWeight](#), [fontFamily](#), [textAlignment](#), and [textOverflow](#) of legend title can be customized by using the [titleStyle](#) property in [legendSettings](#). The [titlePosition](#) is used to set the legend position in Top, Left and Right position. The [maximumTitleWidth](#) is used to set the width of the legend title. By default, it will be 100px.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts'
import { Component } from '@angular/core';
@Component({
  imports: [
    Chart3DAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the Chart component
  template: `<ejs-chart3d [primaryXAxis]='primaryXAxis'
[primaryYAxis]="primaryYAxis" [legendSettings]="legend"
rotation=7 tilt=10 depth=100 [enableRotation]='enableRotation'>
<e-chart3d-series-collection>
  <e-chart3d-series [dataSource]='dataSource' type='Column'
xName='country' yName='gold' name="Gold">
  </e-chart3d-series>
  <e-chart3d-series [dataSource]='dataSource' type='Column'
xName='country' yName='silver' name="Silver" >
  </e-chart3d-series>
  <e-chart3d-series [dataSource]='dataSource' type='Column'
xName='country' yName='bronze' name="Bronze">
  </e-chart3d-series>
</e-chart3d-series-collection>
</ejs-chart3d>`
})
export class AppComponent {
  public dataSource?: Object[];
  public primaryXAxis?: Object;
  public enableRotation?: boolean;
  public legend?: Object;
  public primaryYAxis?: Object;
  ngOnInit(): void {
    this.dataSource = [
      { country: "USA", gold: 50, silver: 70, bronze: 45 },
      { country: "China", gold: 40, silver: 60, bronze: 55 },
      { country: "Japan", gold: 70, silver: 60, bronze: 50 },
      { country: "Australia", gold: 60, silver: 56, bronze: 40 },
      { country: "France", gold: 50, silver: 45, bronze: 35 },
      { country: "Germany", gold: 40, silver: 30, bronze: 22 },
      { country: "Italy", gold: 40, silver: 35, bronze: 37 },
      { country: "Sweden", gold: 30, silver: 25, bronze: 27 }
    ];
    this.primaryXAxis = {
      valueType: 'Category',
    };
    this.enableRotation = true;
    this.primaryYAxis = {
      minimum: 0, maximum: 80,
      interval: 20, title: 'Medals'
    };
    this.legend = {
      visible: true,
      title: 'Countries'
    }
  }
}

```

```

    };
  }
}

```

MAIN.TS

```

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

```

Arrow page navigation

The page number will always be visible while using legend paging. It is now possible to disable the page number and enable page navigation with the left and right arrows. The [enablePages](#) property needs to be set to **false** in order to render the arrow page navigation.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts'
import { Component } from '@angular/core';
@Component({
  imports: [
    Chart3DAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the Chart component
  template: `<ejs-chart3d [primaryXAxis]='primaryXAxis'
[primaryYAxis]="primaryYAxis" [legendSettings]="legend"
rotation=7 tilt=10 depth=100 [enableRotation]='enableRotation'>
<e-chart3d-series-collection>
  <e-chart3d-series [dataSource]='dataSource' type='Column'
xName='country' yName='gold' name="Gold">
    </e-chart3d-series>
  <e-chart3d-series [dataSource]='dataSource' type='Column'
xName='country' yName='silver' name="Silver" >
    </e-chart3d-series>
  <e-chart3d-series [dataSource]='dataSource' type='Column'
xName='country' yName='bronze' name="Bronze">
    </e-chart3d-series>
</e-chart3d-series-collection>
</ejs-chart3d>`
})
export class AppComponent {
  public dataSource?: Object[];
  public primaryXAxis?: Object;
  public enableRotation?: boolean;
  public legend?: Object;
  public primaryYAxis?: Object;
  ngOnInit(): void {
    this.dataSource = [
      { country: "USA", gold: 50, silver: 70, bronze: 45 },
      { country: "China", gold: 40, silver: 60, bronze: 55 },
    ]
  }
}

```

```

        { country: "Japan", gold: 70, silver: 60, bronze: 50 },
        { country: "Australia", gold: 60, silver: 56, bronze: 40 },
        { country: "France", gold: 50, silver: 45, bronze: 35 },
        { country: "Germany", gold: 40, silver: 30, bronze: 22 },
        { country: "Italy", gold: 40, silver: 35, bronze: 37 },
        { country: "Sweden", gold: 30, silver: 25, bronze: 27 }
    ];
    this.primaryXAxis = {
        valueType: 'Category',
    };
    this.enableRotation = true;
    this.primaryYAxis = {
        minimum: 0, maximum: 80,
        interval: 20, title: 'Medals'
    };
    this.legend = {
        visible: true,
        width: '180',
        height: '20',
        enablePages: false
    };
}

```

MAIN.TS

```

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

```

Legend item padding

The [itemPadding](#) property can be used to adjust the space between the legend items.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts';
import { Component } from '@angular/core';
@Component({
    imports: [
        Chart3DAllModule
    ],
    standalone: true,
    selector: 'app-container',
    // specifies the template string for the Chart component
    template: `<ejs-chart3d [primaryXAxis]='primaryXAxis'
[primaryYAxis]="primaryYAxis" [legendSettings]="legend"
rotation=7 tilt=10 depth=100 [enableRotation]='enableRotation'>
<e-chart3d-series-collection>
    <e-chart3d-series [dataSource]='dataSource' type='Column'
xName='country' yName='gold' name="Gold">
    </e-chart3d-series>
    <e-chart3d-series [dataSource]='dataSource' type='Column'
xName='country' yName='silver' name="Silver" >

```

```

        </e-chart3d-series>
        <e-chart3d-series [dataSource]='dataSource' type='Column'
        xName='country' yName='bronze' name="Bronze">
        </e-chart3d-series>
    </e-chart3d-series-collection>
</ejs-chart3d>`
    })
    export class AppComponent {
        public dataSource?: Object[];
        public primaryXAxis?: Object;
        public enableRotation?: boolean;
        public legend?: Object;
        public primaryYAxis?: Object;
        ngOnInit(): void {
            this.dataSource = [
                { country: "USA", gold: 50, silver: 70, bronze: 45 },
                { country: "China", gold: 40, silver: 60, bronze: 55 },
                { country: "Japan", gold: 70, silver: 60, bronze: 50 },
                { country: "Australia", gold: 60, silver: 56, bronze: 40 },
                { country: "France", gold: 50, silver: 45, bronze: 35 },
                { country: "Germany", gold: 40, silver: 30, bronze: 22 },
                { country: "Italy", gold: 40, silver: 35, bronze: 37 },
                { country: "Sweden", gold: 30, silver: 25, bronze: 27 }
            ];
            this.primaryXAxis = {
                valueType: 'Category',
            };
            this.enableRotation = true;
            this.primaryYAxis = {
                minimum: 0, maximum: 80,
                interval: 20, title: 'Medals'
            };
            this.legend = {
                enablePages: false,
                itemPadding: 30
            };
        }
    }
}

```

MAIN.TS

```

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

```

Note: To use the legend feature, we need to inject the `Legend3DService` module into the `@NgModule.providers`.

Tooltip in Angular 3D Chart control

<!-- markdownlint-disable MD036 -->

The 3D Chart will display details about the points through tooltip, when the mouse is moved over the specific point.

Default tooltip

By default, tooltip is not visible. The tooltip can be enabled by setting the [enable](#) property in `tooltipSettings` to **true** and by injecting `Tooltip3DService` module into the `@NgModule.providers`.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts'
import { Component } from '@angular/core';
@Component({
  imports: [
    Chart3DAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the Chart component
  template: `<ejs-chart3d style='display:block;' align='center'
[primaryXAxis]='primaryXAxis' [tooltip]="tooltip"
rotation=7 tilt=10 depth=100 [enableRotation]='enableRotation'>
<e-chart3d-series-collection>
  <e-chart3d-series [dataSource]='dataSource' type='Column'
xName='month' yName='sales'>
    </e-chart3d-series>
  </e-chart3d-series-collection>
</ejs-chart3d>`
})
export class AppComponent {
  public dataSource?: Object[];
  public primaryXAxis?: Object;
  public enableRotation?: boolean;
  public tooltip?: Object;
  ngOnInit(): void {
    this.dataSource = [
      { month: 'Jan', sales: 35 }, { month: 'Feb', sales: 28 },
      { month: 'Mar', sales: 34 }, { month: 'Apr', sales: 32 },
      { month: 'May', sales: 40 }, { month: 'Jun', sales: 32 },
      { month: 'Jul', sales: 35 }, { month: 'Aug', sales: 55 },
      { month: 'Sep', sales: 38 }, { month: 'Oct', sales: 30 },
      { month: 'Nov', sales: 25 }, { month: 'Dec', sales: 32 }
    ];
    this.primaryXAxis = {
      valueType: 'Category',
      labelRotation: -45,
      labelPlacement: 'BetweenTicks'
    };
    this.enableRotation = true;
    this.tooltip = { enable: true };
  }
}
```

MAIN.TS

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

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

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

Fixed tooltip

By default, tooltip track the mouse movement, but the tooltip can be set in fixed position by using the [location](#) property.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts'
import { Component } from '@angular/core';
@Component({
  imports: [
    Chart3DAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the Chart component
  template: `<ejs-chart3d style='display:block;' align='center'
[primaryXAxis]='primaryXAxis' [tooltip]="tooltip"
rotation=7 tilt=10 depth=100 [enableRotation]='enableRotation'>
<e-chart3d-series-collection>
  <e-chart3d-series [dataSource]='dataSource' type='Column'
xName='month' yName='sales'>
    </e-chart3d-series>
  </e-chart3d-series-collection>
</ejs-chart3d>`
})
export class AppComponent {
  public dataSource?: Object[];
  public primaryXAxis?: Object;
  public enableRotation?: boolean;
  public tooltip?: Object;
  ngOnInit(): void {
    this.dataSource = [
      { month: 'Jan', sales: 35 }, { month: 'Feb', sales: 28 },
      { month: 'Mar', sales: 34 }, { month: 'Apr', sales: 32 },
      { month: 'May', sales: 40 }, { month: 'Jun', sales: 32 },
      { month: 'Jul', sales: 35 }, { month: 'Aug', sales: 55 },
      { month: 'Sep', sales: 38 }, { month: 'Oct', sales: 30 },
      { month: 'Nov', sales: 25 }, { month: 'Dec', sales: 32 }
    ];
    this.primaryXAxis = {
      valueType: 'Category',
      labelRotation: -45,
      labelPlacement: 'BetweenTicks'
    };
    this.enableRotation = true;
    this.tooltip = {
      enable: true,
      location: { x: 120, y: 20 }
    };
  }
}
```

```
}

```

MAIN.TS

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

Format the tooltip

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

By default, tooltip shows information of x and y value in points. In addition to that, more information can be shown in tooltip by using the [format](#) property. For example the format `${series.name}` : `${point.y}` shows series name and point y value.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts'
import { Component } from '@angular/core';
@Component({
  imports: [
    Chart3DAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the Chart component
  template: `<ejs-chart3d style='display:block;' align='center'
[primaryXAxis]='primaryXAxis' [tooltip]="tooltip"
rotation=7 tilt=10 depth=100 [enableRotation]='enableRotation'>
<e-chart3d-series-collection>
  <e-chart3d-series [dataSource]='dataSource' type='Column'
xName='month' yName='sales' name="Month">
  </e-chart3d-series>
</e-chart3d-series-collection>
</ejs-chart3d>`
})
export class AppComponent {
  public dataSource?: Object[];
  public primaryXAxis?: Object;
  public enableRotation?: boolean;
  public tooltip?: Object;
  ngOnInit(): void {
    this.dataSource = [
      { month: 'Jan', sales: 35 }, { month: 'Feb', sales: 28 },
      { month: 'Mar', sales: 34 }, { month: 'Apr', sales: 32 },
      { month: 'May', sales: 40 }, { month: 'Jun', sales: 32 },
      { month: 'Jul', sales: 35 }, { month: 'Aug', sales: 55 },
      { month: 'Sep', sales: 38 }, { month: 'Oct', sales: 30 },
      { month: 'Nov', sales: 25 }, { month: 'Dec', sales: 32 }
    ];
    this.primaryXAxis = {
      valueType: 'Category',
```

```

        labelRotation: -45,
        labelPlacement: 'BetweenTicks'
    };
    this.enableRotation = true;
    this.tooltip = {
        enable: true,
        header: 'Unemployment',
        format: '<b>${series.name} : ${point.y}</b>'
    };
}

```

MAIN.TS

```

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

```

Tooltip template

Any HTML elements can be displayed in the tooltip by using the [template](#) property of the tooltip. The `${x}` and `${y}` can be used as place holders in the HTML element to display the x and y values of the corresponding data point.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts'
import { Component } from '@angular/core';
@Component({
  imports: [
    Chart3DAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the Chart component
  template: `<ejs-chart3d style='display:block;' align='center'
[primaryXAxis]='primaryXAxis' [tooltip]="tooltip"
rotation=7 tilt=10 depth=100 [enableRotation]='enableRotation'>
<e-chart3d-series-collection>
  <e-chart3d-series [dataSource]='dataSource' type='Column'
xName='month' yName='sales' name="Month">
    </e-chart3d-series>
  </e-chart3d-series-collection>
</ejs-chart3d>`
})
export class AppComponent {
  public dataSource?: Object[];
  public primaryXAxis?: Object;
  public enableRotation?: boolean;
  public tooltip?: Object;
  ngOnInit(): void {
    this.dataSource = [
      { month: 'Jan', sales: 35 }, { month: 'Feb', sales: 28 },

```



```

        { month: 'Mar', sales: 34 }, { month: 'Apr', sales: 32 },
        { month: 'May', sales: 40 }, { month: 'Jun', sales: 32 },
        { month: 'Jul', sales: 35 }, { month: 'Aug', sales: 55 },
        { month: 'Sep', sales: 38 }, { month: 'Oct', sales: 30 },
        { month: 'Nov', sales: 25 }, { month: 'Dec', sales: 32 }
    ];
    this.primaryXAxis = {
        valueType: 'Category',
        labelRotation: -45,
        labelPlacement: 'BetweenTicks'
    };
    this.enableRotation = true;
    this.tooltip = {
        enable: true,
        template: '<div id="templateWrap"><table style="width:100%;
border: 1px solid black;"><tr><th colspan="2">
bgcolor="#00FFFF">Unemployment</th></tr><tr><td
bgcolor="#00FFFF">${x}</td><td
bgcolor="#00FFFF">${y}</td></tr></table></div>'
    };
}
}

```

MAIN.TS

```

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

```

Customize the appearance of tooltip

The [fill](#) and [border](#) properties are used to customize the background color and border of the tooltip respectively. The [textStyle](#) property in the tooltip is used to customize the font of the tooltip text.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts';
import { Component } from '@angular/core';
@Component({
    imports: [
        Chart3DAllModule
    ],
    standalone: true,
    selector: 'app-container',
    // specifies the template string for the Chart component
    template: `<ejs-chart3d style='display:block;' align='center'
[primaryXAxis]='primaryXAxis' [tooltip]="tooltip"
rotation=7 tilt=10 depth=100 [enableRotation]='enableRotation'>
<e-chart3d-series-collection>
    <e-chart3d-series [dataSource]='dataSource' type='Column'
xName='month' yName='sales' name="Month">
    </e-chart3d-series>
</e-chart3d-series-collection>

```

```

</ejs-chart3d>`
  })
  export class AppComponent {
    public dataSource?: Object[];
    public primaryXAxis?: Object;
    public enableRotation?: boolean;
    public tooltip?: Object;
    ngOnInit(): void {
      this.dataSource = [
        { month: 'Jan', sales: 35 }, { month: 'Feb', sales: 28 },
        { month: 'Mar', sales: 34 }, { month: 'Apr', sales: 32 },
        { month: 'May', sales: 40 }, { month: 'Jun', sales: 32 },
        { month: 'Jul', sales: 35 }, { month: 'Aug', sales: 55 },
        { month: 'Sep', sales: 38 }, { month: 'Oct', sales: 30 },
        { month: 'Nov', sales: 25 }, { month: 'Dec', sales: 32 }
      ];
      this.primaryXAxis = {
        valueType: 'Category',
        labelRotation: -45,
        labelPlacement: 'BetweenTicks'
      };
      this.enableRotation = true;
      this.tooltip = {
        enable: true,
        format: '${series.name} ${point.x} : ${point.y}',
        fill: '#7bb4eb',
        border: {
          width: 2,
          color: 'grey'
        }
      };
    }
  }
}

```

MAIN.TS

```

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

```

<!-- markdownlint-disable MD036 -->

Selection in Angular 3D Chart control

The 3D chart provides selection support for the series and its data points on mouse click.

When mouse is clicked on the data points, the corresponding series legend will also be selected.

We have different types of selection mode for selecting a data.

- None
- Point
- Series
- Cluster

Point

To select a point, set the [selectionMode](#) property to **Point**.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts'
import { Component } from '@angular/core';
@Component({
  imports: [
    Chart3DAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the Chart component
  template: `<ejs-chart3d style='display:block;' align='center'
[primaryXAxis]='primaryXAxis' [selectionMode]="selection"
rotation=7 tilt=10 depth=100 [enableRotation]='enableRotation'>
<e-chart3d-series-collection>
  <e-chart3d-series [dataSource]='dataSource' type='Column'
xName='country' yName='gold'>
  </e-chart3d-series>
  <e-chart3d-series [dataSource]='dataSource' type='Column'
xName='country' yName='silver'>
  </e-chart3d-series>
  <e-chart3d-series [dataSource]='dataSource' type='Column'
xName='country' yName='bronze'>
  </e-chart3d-series>
</e-chart3d-series-collection>
</ejs-chart3d>`
})
export class AppComponent {
  public dataSource?: Object[];
  public primaryXAxis?: Object;
  public selection?: String;
  public enableRotation?: boolean;
  ngOnInit(): void {
    this.dataSource = [
      { country: "USA", gold: 50, silver: 70, bronze: 45 },
      { country: "China", gold: 40, silver: 60, bronze: 55 },
      { country: "Japan", gold: 70, silver: 60, bronze: 50 },
      { country: "Australia", gold: 60, silver: 56, bronze: 40 },
      { country: "France", gold: 50, silver: 45, bronze: 35 },
      { country: "Germany", gold: 40, silver: 30, bronze: 22 },
      { country: "Italy", gold: 40, silver: 35, bronze: 37 },
      { country: "Sweden", gold: 30, silver: 25, bronze: 27 }
    ];
    this.primaryXAxis = {
      valueType: 'Category',
      labelRotation: -45,
      labelPlacement: 'BetweenTicks'
    };
    this.enableRotation = true;
    this.selection = "Point";
  }
}
```

MAIN.TS

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

Series

To select a series, set the [selectionMode](#) property to **Series**.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts'
import { Component } from '@angular/core';
@Component({
  imports: [
    Chart3DAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the Chart component
  template: `<ejs-chart3d style='display:block;' align='center'
[primaryXAxis]='primaryXAxis' [selectionMode]="selection"
rotation=7 tilt=10 depth=100 [enableRotation]='enableRotation'>
<e-chart3d-series-collection>
  <e-chart3d-series [dataSource]='dataSource' type='Column'
xName='country' yName='gold'>
    </e-chart3d-series>
  <e-chart3d-series [dataSource]='dataSource' type='Column'
xName='country' yName='silver'>
    </e-chart3d-series>
  <e-chart3d-series [dataSource]='dataSource' type='Column'
xName='country' yName='bronze'>
    </e-chart3d-series>
</e-chart3d-series-collection>
</ejs-chart3d>`
})
export class AppComponent {
  public dataSource?: Object[];
  public primaryXAxis?: Object;
  public selection?: String;
  public enableRotation?: boolean;
  ngOnInit(): void {
    this.dataSource = [
      { country: "USA", gold: 50, silver: 70, bronze: 45 },
      { country: "China", gold: 40, silver: 60, bronze: 55 },
      { country: "Japan", gold: 70, silver: 60, bronze: 50 },
      { country: "Australia", gold: 60, silver: 56, bronze: 40 },
      { country: "France", gold: 50, silver: 45, bronze: 35 },
      { country: "Germany", gold: 40, silver: 30, bronze: 22 },
      { country: "Italy", gold: 40, silver: 35, bronze: 37 },
      { country: "Sweden", gold: 30, silver: 25, bronze: 27 }
    ];
  }
}
```

```

        this.primaryXAxis = {
            valueType: 'Category',
            labelRotation: -45,
            labelPlacement: 'BetweenTicks'
        };
        this.enableRotation = true;
        this.selection = "Series";
    }
}

```

MAIN.TS

```

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

```

Cluster

To select the points that corresponds to the same index in all the series, set the [selectionMode](#) property to **Cluster**.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts';
import { Component } from '@angular/core';
@Component({
    imports: [
        Chart3DAllModule
    ],
    standalone: true,
    selector: 'app-container',
    // specifies the template string for the Chart component
    template: `<ejs-chart3d style='display:block;' align='center'
[primaryXAxis]='primaryXAxis' [selectionMode]="selection"
rotation=7 tilt=10 depth=100 [enableRotation]='enableRotation'>
<e-chart3d-series-collection>
    <e-chart3d-series [dataSource]='dataSource' type='Column'
xName='country' yName='gold'>
    </e-chart3d-series>
    <e-chart3d-series [dataSource]='dataSource' type='Column'
xName='country' yName='silver'>
    </e-chart3d-series>
    <e-chart3d-series [dataSource]='dataSource' type='Column'
xName='country' yName='bronze'>
    </e-chart3d-series>
</e-chart3d-series-collection>
</ejs-chart3d>`
})
export class AppComponent {
    public dataSource?: Object[];
    public primaryXAxis?: Object;
    public selection?: String;
    public enableRotation?: boolean;
}

```

```

ngOnInit(): void {
  this.dataSource = [
    { country: "USA", gold: 50, silver: 70, bronze: 45 },
    { country: "China", gold: 40, silver: 60, bronze: 55 },
    { country: "Japan", gold: 70, silver: 60, bronze: 50 },
    { country: "Australia", gold: 60, silver: 56, bronze: 40 },
    { country: "France", gold: 50, silver: 45, bronze: 35 },
    { country: "Germany", gold: 40, silver: 30, bronze: 22 },
    { country: "Italy", gold: 40, silver: 35, bronze: 37 },
    { country: "Sweden", gold: 30, silver: 25, bronze: 27 }
  ];
  this.primaryXAxis = {
    valueType: 'Category',
    labelRotation: -45,
    labelPlacement: 'BetweenTicks'
  };
  this.enableRotation = true;
  this.selection = "Cluster";
}
}

```

MAIN.TS

```

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

```

Selection type

To select multiple points or series, enable the [isMultiSelect](#) property.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts';
import { Component } from '@angular/core';
@Component({
  imports: [
    Chart3DAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the Chart component
  template: `<ejs-chart3d style='display:block;' align='center'
[primaryXAxis]='primaryXAxis' [selectionMode]="selection"
[isMultiSelect]=true
rotation=7 tilt=10 depth=100 [enableRotation]='enableRotation'>
<e-chart3d-series-collection>
  <e-chart3d-series [dataSource]='dataSource' type='Column'
xName='country' yName='gold'>
  </e-chart3d-series>
  <e-chart3d-series [dataSource]='dataSource' type='Column'
xName='country' yName='silver'>
  </e-chart3d-series>

```

```

    <e-chart3d-series [dataSource]='dataSource' type='Column'
    xName='country' yName='bronze'>
    </e-chart3d-series>
  </e-chart3d-series-collection>
</ejs-chart3d>`
})
export class AppComponent {
  public dataSource?: Object[];
  public primaryXAxis?: Object;
  public selection?: String;
  public enableRotation?: boolean;
  ngOnInit(): void {
    this.dataSource = [
      { country: "USA", gold: 50, silver: 70, bronze: 45 },
      { country: "China", gold: 40, silver: 60, bronze: 55 },
      { country: "Japan", gold: 70, silver: 60, bronze: 50 },
      { country: "Australia", gold: 60, silver: 56, bronze: 40 },
      { country: "France", gold: 50, silver: 45, bronze: 35 },
      { country: "Germany", gold: 40, silver: 30, bronze: 22 },
      { country: "Italy", gold: 40, silver: 35, bronze: 37 },
      { country: "Sweden", gold: 30, silver: 25, bronze: 27 }
    ];
    this.primaryXAxis = {
      valueType: 'Category',
      labelRotation: -45,
      labelPlacement: 'BetweenTicks'
    };
    this.enableRotation = true;
    this.selection = "Point";
  }
}

```

MAIN.TS

```

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

```

Selection during initial loading

In a 3D chart, selecting a point or series during initial loading can only be done programmatically. The [selectedDataIndexes](#) property can be used for this.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts'
import { Component } from '@angular/core';
@Component({
  imports: [
    Chart3DAllModule
  ],
  standalone: true,
  selector: 'app-container',

```

```

// specifies the template string for the Chart component
template: `<ejs-chart3d style='display:block;' align='center'
[primaryXAxis]='primaryXAxis' [selectionMode]="selection"
[isMultiSelect]=true [selectedDataIndexes]="selectedDataIndexes"
rotation=7 tilt=10 depth=100 [enableRotation]='enableRotation'>
  <e-chart3d-series-collection>
    <e-chart3d-series [dataSource]='dataSource' type='Column'
xName='country' yName='gold'>
    </e-chart3d-series>
    <e-chart3d-series [dataSource]='dataSource' type='Column'
xName='country' yName='silver'>
    </e-chart3d-series>
    <e-chart3d-series [dataSource]='dataSource' type='Column'
xName='country' yName='bronze'>
    </e-chart3d-series>
  </e-chart3d-series-collection>
</ejs-chart3d>`
))
export class AppComponent {
  public dataSource?: Object[];
  public selectedDataIndexes?: Object[];
  public primaryXAxis?: Object;
  public selection?: String;
  public enableRotation?: boolean;
  ngOnInit(): void {
    this.dataSource = [
      { country: "USA", gold: 50, silver: 70, bronze: 45 },
      { country: "China", gold: 40, silver: 60, bronze: 55 },
      { country: "Japan", gold: 70, silver: 60, bronze: 50 },
      { country: "Australia", gold: 60, silver: 56, bronze: 40 },
      { country: "France", gold: 50, silver: 45, bronze: 35 },
      { country: "Germany", gold: 40, silver: 30, bronze: 22 },
      { country: "Italy", gold: 40, silver: 35, bronze: 37 },
      { country: "Sweden", gold: 30, silver: 25, bronze: 27 }
    ];
    this.primaryXAxis = {
      valueType: 'Category',
      labelRotation: -45,
      labelPlacement: 'BetweenTicks'
    };
    this.enableRotation = true;
    this.selection = "Point";
    this.selectedDataIndexes = [
      { series: 0, point: 1 }, { series: 2, point: 3 }
    ];
  }
}

```

MAIN.TS

```

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

```


Selection through legend

To select a point or series through legend use the [toggleVisibility](#) property. Also, use [enableHighlight](#) property for highlighting the series through legend.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts'
import { Component } from '@angular/core';
@Component({
  imports: [
    Chart3DAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the Chart component
  template: `<ejs-chart3d style='display:block;' align='center'
[primaryXAxis]='primaryXAxis' [legendSettings]="legend"
rotation=7 tilt=10 depth=100 [enableRotation]='enableRotation'>
<e-chart3d-series-collection>
  <e-chart3d-series [dataSource]='dataSource' type='Column'
xName='country' yName='gold' name="Gold">
  </e-chart3d-series>
  <e-chart3d-series [dataSource]='dataSource' type='Column'
xName='country' yName='silver' name="Silver">
  </e-chart3d-series>
  <e-chart3d-series [dataSource]='dataSource' type='Column'
xName='country' yName='bronze' name="Bronze">
  </e-chart3d-series>
</e-chart3d-series-collection>
</ejs-chart3d>`
})
export class AppComponent {
  public dataSource?: Object[];
  public primaryXAxis?: Object;
  public legend?: Object;
  public enableRotation?: boolean;
  ngOnInit(): void {
    this.dataSource = [
      { country: "USA", gold: 50, silver: 70, bronze: 45 },
      { country: "China", gold: 40, silver: 60, bronze: 55 },
      { country: "Japan", gold: 70, silver: 60, bronze: 50 },
      { country: "Australia", gold: 60, silver: 56, bronze: 40 },
      { country: "France", gold: 50, silver: 45, bronze: 35 },
      { country: "Germany", gold: 40, silver: 30, bronze: 22 },
      { country: "Italy", gold: 40, silver: 35, bronze: 37 },
      { country: "Sweden", gold: 30, silver: 25, bronze: 27 }
    ];
    this.primaryXAxis = {
      valueType: 'Category',
      labelRotation: -45,
      labelPlacement: 'BetweenTicks'
    };
    this.enableRotation = true;
    this.legend = {
      visible: true,
```

```

        toggleVisibility: false,
        enableHighlight: true
    }
}
}

```

MAIN.TS

```

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

```

Print and Export in Angular 3D Chart control

Print

The rendered 3D chart can be printed directly from the browser by calling the public method [print](#). The ID of the 3D chart's div element must be passed as the input parameter to that method.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts'
import { ChartComponent } from '@syncfusion/ej2-angular-charts';
import { Component, ViewChild } from '@angular/core';
@Component({
  imports: [
    Chart3DAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the Chart component
  template: `<button ej-button id='print' (click)='print()'>Print</button>
    <ejs-chart3d #chart style='display:block;' align='center'
    [primaryXAxis]='primaryXAxis'
    rotation=7 tilt=10 depth=100 [enableRotation]='enableRotation'>
    <e-chart3d-series-collection>
      <e-chart3d-series [dataSource]='dataSource' type='Column'
    xName='country' yName='gold' name="Gold">
      </e-chart3d-series>
      <e-chart3d-series [dataSource]='dataSource' type='Column'
    xName='country' yName='silver' name="Silver">
      </e-chart3d-series>
      <e-chart3d-series [dataSource]='dataSource' type='Column'
    xName='country' yName='bronze' name="Bronze">
      </e-chart3d-series>
    </e-chart3d-series-collection>
  </ejs-chart3d>`
})
export class AppComponent {
  public dataSource?: Object[];
  public primaryXAxis?: Object;
  public enableRotation?: boolean;
  @ViewChild('chart')
  public chartObj?: ChartComponent;
}

```

```

ngOnInit(): void {
  this.dataSource = [
    { country: "USA", gold: 50, silver: 70, bronze: 45 },
    { country: "China", gold: 40, silver: 60, bronze: 55 },
    { country: "Japan", gold: 70, silver: 60, bronze: 50 },
    { country: "Australia", gold: 60, silver: 56, bronze: 40 },
    { country: "France", gold: 50, silver: 45, bronze: 35 },
    { country: "Germany", gold: 40, silver: 30, bronze: 22 },
    { country: "Italy", gold: 40, silver: 35, bronze: 37 },
    { country: "Sweden", gold: 30, silver: 25, bronze: 27 }
  ];
  this.primaryXAxis = {
    valueType: 'Category',
    labelRotation: -45,
    labelPlacement: 'BetweenTicks'
  };
  this.enableRotation = true;
}
print() {
  this.chartObj?.print();
}
}

```

MAIN.TS

```

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

```

Export

The rendered 3D chart can be exported to JPEG, PNG, SVG, or PDF format using the [export](#) method. The input parameters for this method are: `type` for format and `fileName` for result.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts'
import { ChartComponent } from '@syncfusion/ej2-angular-charts';
import { Component, ViewChild } from '@angular/core';
@Component({
  imports: [
    Chart3DAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the Chart component
  template: `<button ej-button id='print'
(click)='export()'>Export</button>
<ejs-chart3d #chart style='display:block;' align='center'
[primaryXAxis]='primaryXAxis'
rotation=7 tilt=10 depth=100 [enableRotation]='enableRotation'>
<e-chart3d-series-collection>

```

```

    <e-chart3d-series [dataSource]='dataSource' type='Column'
    xName='country' yName='gold' name="Gold">
    </e-chart3d-series>
    <e-chart3d-series [dataSource]='dataSource' type='Column'
    xName='country' yName='silver' name="Silver">
    </e-chart3d-series>
    <e-chart3d-series [dataSource]='dataSource' type='Column'
    xName='country' yName='bronze' name="Bronze">
    </e-chart3d-series>
  </e-chart3d-series-collection>
</ejs-chart3d>`
  })
  export class AppComponent {
    public dataSource?: Object[];
    public primaryXAxis?: Object;
    public enableRotation?: boolean;
    @ViewChild('chart')
    public chartObj?: ChartComponent;
    ngOnInit(): void {
      this.dataSource = [
        { country: "USA", gold: 50, silver: 70, bronze: 45 },
        { country: "China", gold: 40, silver: 60, bronze: 55 },
        { country: "Japan", gold: 70, silver: 60, bronze: 50 },
        { country: "Australia", gold: 60, silver: 56, bronze: 40 },
        { country: "France", gold: 50, silver: 45, bronze: 35 },
        { country: "Germany", gold: 40, silver: 30, bronze: 22 },
        { country: "Italy", gold: 40, silver: 35, bronze: 37 },
        { country: "Sweden", gold: 30, silver: 25, bronze: 27 }
      ];
      this.primaryXAxis = {
        valueType: 'Category',
        labelRotation: -45,
        labelPlacement: 'BetweenTicks'
      };
      this.enableRotation = true;
    }
    export() {
      this.chartObj?.export("PNG", "charts");
    }
  }
}

```

MAIN.TS

```

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

```

Appearance in Angular 3D Chart control

Custom color palette

The default color of series or points can be customized by providing a custom color palette of your choice by using the [palettes](#) property.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts'
import { Component } from '@angular/core';
@Component({
  imports: [
    Chart3DAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the Chart component
  template: `<ejs-chart3d #chart style='display:block;' align='center'
[primaryXAxis]='primaryXAxis' [palettes]="palettes"
rotation=7 tilt=10 depth=100 [enableRotation]='enableRotation'>
<e-chart3d-series-collection>
  <e-chart3d-series [dataSource]='dataSource' type='Column'
xName='country' yName='gold'>
  </e-chart3d-series>
  <e-chart3d-series [dataSource]='dataSource' type='Column'
xName='country' yName='silver'>
  </e-chart3d-series>
  <e-chart3d-series [dataSource]='dataSource' type='Column'
xName='country' yName='bronze'>
  </e-chart3d-series>
</e-chart3d-series-collection>
</ejs-chart3d>`
})
export class AppComponent {
  public dataSource?: Object[];
  public primaryXAxis?: Object;
  public primaryYAxis?: Object;
  public enableRotation?: boolean;
  public palettes?: String[];
  ngOnInit(): void {
    this.dataSource = [
      { country: "USA", gold: 50, silver: 70, bronze: 45 },
      { country: "China", gold: 40, silver: 60, bronze: 55 },
      { country: "Japan", gold: 70, silver: 60, bronze: 50 },
      { country: "Australia", gold: 60, silver: 56, bronze: 40 },
      { country: "France", gold: 50, silver: 45, bronze: 35 },
      { country: "Germany", gold: 40, silver: 30, bronze: 22 },
      { country: "Italy", gold: 40, silver: 35, bronze: 37 },
      { country: "Sweden", gold: 30, silver: 25, bronze: 27 }
    ];
    this.enableRotation = true;
    this.primaryXAxis = {
      valueType: 'Category',
    };
    this.palettes = ["#E94649", "#F6B53F", "#6FAAB0", "#C4C24A"];
  }
}

```

MAIN.TS

```

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

```

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

Data point customization

The color of an individual data point can be customized using the below options.

Point color mapping

The color for the points can be bound from the [dataSource](#) for the series by utilizing the [pointColorMapping](#) property.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts'
import { Component } from '@angular/core';
@Component({
  imports: [
    Chart3DAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the Chart component
  template: `<ejs-chart3d #chart style='display:block;' align='center'
[primaryXAxis]='primaryXAxis'
rotation=7 tilt=10 depth=100 [enableRotation]='enableRotation'>
  <e-chart3d-series-collection>
    <e-chart3d-series [dataSource]='dataSource' type='Column' xName='x'
yName='y' pointColorMapping="color">
    </e-chart3d-series>
  </e-chart3d-series-collection>
</ejs-chart3d>`
})
export class AppComponent {
  public dataSource?: Object[];
  public primaryXAxis?: Object;
  public primaryYAxis?: Object;
  public enableRotation?: boolean;
  public palettes?: String[];
  ngOnInit(): void {
    this.dataSource = [
      { x: 'Jan', y: 6.96, color: "#ed4c40" },
      { x: 'Feb', y: 8.9, color: "#3285f3"},
      { x: 'Mar', y: 12, color: "#1dd7f3"},
      { x: 'Apr', y: 17.5, color: "#fe1684" },
      { x: 'May', y: 22.1, color: "#4633f2" }
    ];
    this.enableRotation = true;
    this.primaryXAxis = {
      valueType: 'Category',
    };
  }
}
```

MAIN.TS

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

Point level customization

The data label and fill color of each data point can be customized using the [pointRender](#) and [textRender](#) events.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts'
import { Component } from '@angular/core';
import { Chart3DPointRenderEventArgs } from '@syncfusion/ej2-angular-charts';
@Component({
  imports: [
    Chart3DAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the Chart component
  template: `<ejs-chart3d #chart (pointRender)=pointRender($event) '
[primaryXAxis]='primaryXAxis'
rotation=7 tilt=10 depth=100 [enableRotation]='enableRotation'>
<e-chart3d-series-collection>
  <e-chart3d-series [dataSource]='dataSource' type='Column'
xName='country' yName='gold'>
    </e-chart3d-series>
  </e-chart3d-series-collection>
</ejs-chart3d>`
})
export class AppComponent {
  public dataSource?: Object[];
  public primaryXAxis?: Object;
  public primaryYAxis?: Object;
  public enableRotation?: boolean;
  public pointRender(args: Chart3DPointRenderEventArgs): void {
    let colors: string[] = ['#00bdae', '#404041', '#357cd2', '#e56590',
'#f8b883', '#70ad47', '#dd8abd', '#7f84e8', '#7bb4eb', '#ea7a57'];
    args.fill = colors[args.point.index];
  };
  ngOnInit(): void {
    this.dataSource = [
      { country: "USA", gold: 50, silver: 70, bronze: 45 },
      { country: "China", gold: 40, silver: 60, bronze: 55 },
      { country: "Japan", gold: 70, silver: 60, bronze: 50 },
      { country: "Australia", gold: 60, silver: 56, bronze: 40 },
      { country: "France", gold: 50, silver: 45, bronze: 35 },
      { country: "Germany", gold: 40, silver: 30, bronze: 22 },
      { country: "Italy", gold: 40, silver: 35, bronze: 37 },
      { country: "Sweden", gold: 30, silver: 25, bronze: 27 }
    ];
    this.enableRotation = true;
    this.primaryXAxis = {
```

```

        valueType: 'Category',
    };
}
}

```

MAIN.TS

```

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

```

<!-- markdownlint-disable MD036 -->

Chart area customization

<!-- markdownlint-disable MD036 -->

Customize the chart background

The background color and border of the 3D chart can be customized using the [background](#) and [border](#) properties.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts'
import { Component } from '@angular/core';
@Component({
  imports: [
    Chart3DAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the Chart component
  template: `<ejs-chart3d #chart style='display:block;' align='center'
[primaryXAxis]='primaryXAxis' background="skyblue" [border]="border"
rotation=7 tilt=10 depth=100 [enableRotation]='enableRotation'>
<e-chart3d-series-collection>
  <e-chart3d-series [dataSource]='dataSource' type='Column'
xName='country' yName='gold'>
    </e-chart3d-series>
  </e-chart3d-series-collection>
</ejs-chart3d>`
})
export class AppComponent {
  public dataSource?: Object[];
  public primaryXAxis?: Object;
  public primaryYAxis?: Object;
  public enableRotation?: boolean;
  public border?: Object;
  ngOnInit(): void {
    this.dataSource = [
      { country: "USA", gold: 50, silver: 70, bronze: 45 },
      { country: "China", gold: 40, silver: 60, bronze: 55 },
      { country: "Japan", gold: 70, silver: 60, bronze: 50 },
    ]
  }
}

```



```

        { country: "Australia", gold: 60, silver: 56, bronze: 40 },
        { country: "France", gold: 50, silver: 45, bronze: 35 },
        { country: "Germany", gold: 40, silver: 30, bronze: 22 },
        { country: "Italy", gold: 40, silver: 35, bronze: 37 },
        { country: "Sweden", gold: 30, silver: 25, bronze: 27 }
    ];
    this.enableRotation = true;
    this.primaryXAxis = {
        valueType: 'Category',
    };
    this.border = {
        color: "#FF0000", width: 2
    }
}
}

```

MAIN.TS

```

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

```

Chart margin

The 3D chart's margin can be set from its container using the [margin](#) property.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts'
import { Component } from '@angular/core';
@Component({
  imports: [
    Chart3DAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the Chart component
  template: `<ejs-chart3d #chart style='display:block;' align='center'
[primaryXAxis]='primaryXAxis' background="skyblue" [border]="border"
[margin]="margin"
rotation=7 tilt=10 depth=100 [enableRotation]='enableRotation'>
<e-chart3d-series-collection>
  <e-chart3d-series [dataSource]='dataSource' type='Column'
xName='country' yName='gold'>
  </e-chart3d-series>
</e-chart3d-series-collection>
</ejs-chart3d>`
})
export class AppComponent {
  public dataSource?: Object[];
  public primaryXAxis?: Object;
  public primaryYAxis?: Object;
  public enableRotation?: boolean;
}

```

```

public border?: Object;
public margin?: Object;
ngOnInit(): void {
    this.dataSource = [
        { country: "USA", gold: 50, silver: 70, bronze: 45 },
        { country: "China", gold: 40, silver: 60, bronze: 55 },
        { country: "Japan", gold: 70, silver: 60, bronze: 50 },
        { country: "Australia", gold: 60, silver: 56, bronze: 40 },
        { country: "France", gold: 50, silver: 45, bronze: 35 },
        { country: "Germany", gold: 40, silver: 30, bronze: 22 },
        { country: "Italy", gold: 40, silver: 35, bronze: 37 },
        { country: "Sweden", gold: 30, silver: 25, bronze: 27 }
    ];
    this.enableRotation = true;
    this.primaryXAxis = {
        valueType: 'Category',
    };
    this.border = {
        color: "#FF0000", width: 2
    };
    this.margin = {
        left: 40, right: 40, top: 40, bottom: 40
    };
}
}

```

MAIN.TS

```

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

```

Animation

To customize the animation for a particular series, the [animation](#) property can be used. It can be enabled or disabled by using the [enable](#) property. The [duration](#) property specifies the duration of an animation and the [delay](#) property allows us to start the animation at desire time.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts'
import { Component } from '@angular/core';
@Component({
  imports: [
    Chart3DAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the Chart component
  template: `<ejs-chart3d style='display:block;' align='center'
[primaryXAxis]='primaryXAxis'
rotation=7 tilt=10 depth=100 [enableRotation]='enableRotation'>
<e-chart3d-series-collection>

```

```

    <e-chart3d-series [dataSource]='dataSource' type='Column'
    xName='country' yName='gold' [animation]="animation" >
    </e-chart3d-series>
  </e-chart3d-series-collection>
</ejs-chart3d>`
  })
  export class AppComponent {
    public dataSource?: Object[];
    public primaryXAxis?: Object;
    public enableRotation?: boolean;
    public animation?: Object;
    ngOnInit(): void {
      this.dataSource = [
        { country: "USA", gold: 50, silver: 70, bronze: 45 },
        { country: "China", gold: 40, silver: 60, bronze: 55 },
        { country: "Japan", gold: 70, silver: 60, bronze: 50 },
        { country: "Australia", gold: 60, silver: 56, bronze: 40 },
        { country: "France", gold: 50, silver: 45, bronze: 35 },
        { country: "Germany", gold: 40, silver: 30, bronze: 22 },
        { country: "Italy", gold: 40, silver: 35, bronze: 37 },
        { country: "Sweden", gold: 30, silver: 25, bronze: 27 }
      ];
      this.primaryXAxis = {
        valueType: 'Category',
        labelRotation: -45,
        labelPlacement: 'BetweenTicks'
      };
      this.enableRotation = true;
      this.animation = {
        enable: true,
        duration: 2000,
        delay: 200
      }
    }
  }
}

```

MAIN.TS

```

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

```

Chart rotation

The 3D chart can be rotated by using the [enableRotation](#) property.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts'
import { Component } from '@angular/core';
@Component({
  imports: [
    Chart3DAllModule
  ],

```

```

standalone: true,
  selector: 'app-container',
  // specifies the template string for the Chart component
  template: `<ejs-chart3d style='display:block;' align='center'
[primaryXAxis]='primaryXAxis'
  rotation=7 tilt=10 depth=100 [enableRotation]='enableRotation'>
    <e-chart3d-series-collection>
      <e-chart3d-series [dataSource]='dataSource' type='Column'
xName='country' yName='gold'>
      </e-chart3d-series>
    </e-chart3d-series-collection>
</ejs-chart3d>`
})
export class AppComponent {
  public dataSource?: Object[];
  public primaryXAxis?: Object;
  public enableRotation?: boolean;
  ngOnInit(): void {
    this.dataSource = [
      { country: "USA", gold: 50, silver: 70, bronze: 45 },
      { country: "China", gold: 40, silver: 60, bronze: 55 },
      { country: "Japan", gold: 70, silver: 60, bronze: 50 },
      { country: "Australia", gold: 60, silver: 56, bronze: 40 },
      { country: "France", gold: 50, silver: 45, bronze: 35 },
      { country: "Germany", gold: 40, silver: 30, bronze: 22 },
      { country: "Italy", gold: 40, silver: 35, bronze: 37 },
      { country: "Sweden", gold: 30, silver: 25, bronze: 27 }
    ];
    this.primaryXAxis = {
      valueType: 'Category',
      labelRotation: -45,
      labelPlacement: 'BetweenTicks'
    };
    this.enableRotation = true;
  }
}

```

MAIN.TS

```

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

```

Title

The 3D chart can be given a title by using [title](#) property, to show the information about the data plotted.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts'
import { Component } from '@angular/core';
@Component({
  imports: [
    Chart3DAllModule

```

```

    ],
    standalone: true,
    selector: 'app-container',
    // specifies the template string for the Chart component
    template: `<ejs-chart3d style='display:block;' align='center'
[primaryXAxis]='primaryXAxis' title="Olympic Medals"
rotation=7 tilt=10 depth=100 [enableRotation]='enableRotation'>
    <e-chart3d-series-collection>
        <e-chart3d-series [dataSource]='dataSource' type='Column'
xName='country' yName='gold'>
            </e-chart3d-series>
        </e-chart3d-series-collection>
    </ejs-chart3d>`
  })
  export class AppComponent {
    public dataSource?: Object[];
    public primaryXAxis?: Object;
    public enableRotation?: boolean;
    public animation?: Object;
    ngOnInit(): void {
      this.dataSource = [
        { country: "USA", gold: 50, silver: 70, bronze: 45 },
        { country: "China", gold: 40, silver: 60, bronze: 55 },
        { country: "Japan", gold: 70, silver: 60, bronze: 50 },
        { country: "Australia", gold: 60, silver: 56, bronze: 40 },
        { country: "France", gold: 50, silver: 45, bronze: 35 },
        { country: "Germany", gold: 40, silver: 30, bronze: 22 },
        { country: "Italy", gold: 40, silver: 35, bronze: 37 },
        { country: "Sweden", gold: 30, silver: 25, bronze: 27 }
      ];
      this.primaryXAxis = {
        valueType: 'Category',
        labelRotation: -45,
        labelPlacement: 'BetweenTicks'
      };
      this.enableRotation = true;
    }
  }
}

```

MAIN.TS

```

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

```

Title position

By using the [position](#) property in [titleStyle](#), the [title](#) can be positioned at left, right, top or bottom of the 3D chart. The title is positioned at the top of the 3D chart, by default.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts'

```

```

import { Component } from '@angular/core';
@Component({
  imports: [
    Chart3DAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the Chart component
  template: `<ejs-chart3d style='display:block;' align='center'
[primaryXAxis]='primaryXAxis' title="Olympic Medals"
[titleStyle]="titleStyle"
rotation=7 tilt=10 depth=100 [enableRotation]='enableRotation'>
  <e-chart3d-series-collection>
    <e-chart3d-series [dataSource]='dataSource' type='Column'
xName='country' yName='gold'>
    </e-chart3d-series>
  </e-chart3d-series-collection>
</ejs-chart3d>`
})
export class AppComponent {
  public dataSource?: Object[];
  public primaryXAxis?: Object;
  public enableRotation?: boolean;
  public titleStyle?: Object;
  ngOnInit(): void {
    this.dataSource = [
      { country: "USA", gold: 50, silver: 70, bronze: 45 },
      { country: "China", gold: 40, silver: 60, bronze: 55 },
      { country: "Japan", gold: 70, silver: 60, bronze: 50 },
      { country: "Australia", gold: 60, silver: 56, bronze: 40 },
      { country: "France", gold: 50, silver: 45, bronze: 35 },
      { country: "Germany", gold: 40, silver: 30, bronze: 22 },
      { country: "Italy", gold: 40, silver: 35, bronze: 37 },
      { country: "Sweden", gold: 30, silver: 25, bronze: 27 }
    ];
    this.primaryXAxis = {
      valueType: 'Category',
      labelRotation: -45,
      labelPlacement: 'BetweenTicks'
    };
    this.enableRotation = true;
    this.titleStyle = {
      position: "Bottom"
    };
  }
}

```

MAIN.TS

```

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

```

The custom option is used to position the title anywhere in the 3D chart using [x](#) and [y](#) coordinates.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts'
import { Component } from '@angular/core';
@Component({
  imports: [
    Chart3DAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the Chart component
  template: `<ejs-chart3d style='display:block;' align='center'
[primaryXAxis]='primaryXAxis' title="Olympic Medals"
[titleStyle]="titleStyle"
rotation=7 tilt=10 depth=100 [enableRotation]='enableRotation'>
<e-chart3d-series-collection>
  <e-chart3d-series [dataSource]='dataSource' type='Column'
xName='country' yName='gold'>
    </e-chart3d-series>
  </e-chart3d-series-collection>
</ejs-chart3d>`
})
export class AppComponent {
  public dataSource?: Object[];
  public primaryXAxis?: Object;
  public enableRotation?: boolean;
  public titleStyle?: Object;
  ngOnInit(): void {
    this.dataSource = [
      { country: "USA", gold: 50, silver: 70, bronze: 45 },
      { country: "China", gold: 40, silver: 60, bronze: 55 },
      { country: "Japan", gold: 70, silver: 60, bronze: 50 },
      { country: "Australia", gold: 60, silver: 56, bronze: 40 },
      { country: "France", gold: 50, silver: 45, bronze: 35 },
      { country: "Germany", gold: 40, silver: 30, bronze: 22 },
      { country: "Italy", gold: 40, silver: 35, bronze: 37 },
      { country: "Sweden", gold: 30, silver: 25, bronze: 27 }
    ];
    this.primaryXAxis = {
      valueType: 'Category',
      labelRotation: -45,
      labelPlacement: 'BetweenTicks'
    };
    this.enableRotation = true;
    this.titleStyle = {
      position: 'Custom',
      x: 300,
      y: 60
    };
  }
}

```

MAIN.TS

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

Title alignment

The title can be aligned to the near, far, or center of the 3D chart by using the [textAlignment](#) property.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts'
import { Component } from '@angular/core';
@Component({
  imports: [
    Chart3DAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the Chart component
  template: `<ejs-chart3d style='display:block;' align='center'
[primaryXAxis]='primaryXAxis' title="Olympic Medals"
[titleStyle]="titleStyle"
rotation=7 tilt=10 depth=100 [enableRotation]='enableRotation'>
<e-chart3d-series-collection>
  <e-chart3d-series [dataSource]='dataSource' type='Column'
xName='country' yName='gold'>
    </e-chart3d-series>
  </e-chart3d-series-collection>
</ejs-chart3d>`
})
export class AppComponent {
  public dataSource?: Object[];
  public primaryXAxis?: Object;
  public enableRotation?: boolean;
  public titleStyle?: Object;
  ngOnInit(): void {
    this.dataSource = [
      { country: "USA", gold: 50, silver: 70, bronze: 45 },
      { country: "China", gold: 40, silver: 60, bronze: 55 },
      { country: "Japan", gold: 70, silver: 60, bronze: 50 },
      { country: "Australia", gold: 60, silver: 56, bronze: 40 },
      { country: "France", gold: 50, silver: 45, bronze: 35 },
      { country: "Germany", gold: 40, silver: 30, bronze: 22 },
      { country: "Italy", gold: 40, silver: 35, bronze: 37 },
      { country: "Sweden", gold: 30, silver: 25, bronze: 27 }
    ];
    this.primaryXAxis = {
      valueType: 'Category',
      labelRotation: -45,
      labelPlacement: 'BetweenTicks'
    };
    this.enableRotation = true;
    this.titleStyle = {
      textAlignment: 'Far'
    }
  }
}
```



```

    };
  }
}

```

MAIN.TS

```

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

```

Title customization

The [titleStyle](#) property of the 3D chart provides options to customize the title by using the following properties.

- [size](#) - Specifies the size of the title.
- [color](#) - Specifies the color for the title.
- [fontFamily](#) - Specifies the font family for the title.
- [fontWeight](#) - Specifies the font weight of the title.
- [fontStyle](#) - Specifies the font style for the title.
- [opacity](#) - Specifies the opacity for the color of the title.
- [textAlignment](#) - Specifies the alignment of the title.
- [textOverflow](#) - Specifies the overflow of the title.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { Chart3DAllModule } from '@syncfusion/ej2-angular-charts';
import { Component } from '@angular/core';
@Component({
  imports: [
    Chart3DAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the Chart component
  template: `<ejs-chart3d style='display:block;' align='center'
[primaryXAxis]='primaryXAxis' title="Olympic Medals"
[titleStyle]="titleStyle"
rotation=7 tilt=10 depth=100 [enableRotation]='enableRotation'>
<e-chart3d-series-collection>
  <e-chart3d-series [dataSource]='dataSource' type='Column'
xName='country' yName='gold'>
    </e-chart3d-series>
  </e-chart3d-series-collection>
</ejs-chart3d>`
})
export class AppComponent {
  public dataSource?: Object[];
  public primaryXAxis?: Object;
  public enableRotation?: boolean;
  public titleStyle?: Object;
}

```

```

ngOnInit(): void {
  this.dataSource = [
    { country: "USA", gold: 50, silver: 70, bronze: 45 },
    { country: "China", gold: 40, silver: 60, bronze: 55 },
    { country: "Japan", gold: 70, silver: 60, bronze: 50 },
    { country: "Australia", gold: 60, silver: 56, bronze: 40 },
    { country: "France", gold: 50, silver: 45, bronze: 35 },
    { country: "Germany", gold: 40, silver: 30, bronze: 22 },
    { country: "Italy", gold: 40, silver: 35, bronze: 37 },
    { country: "Sweden", gold: 30, silver: 25, bronze: 27 }
  ];
  this.primaryXAxis = {
    valueType: 'Category',
    labelRotation: -45,
    labelPlacement: 'BetweenTicks'
  };
  this.enableRotation = true;
  this.titleStyle = {
    size: '18px', color: 'Red', textOverflow: 'Wrap'
  };
}
}

```

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 3D Chart control

Accessibility is achieved in the 3D chart control through WAI-ARIA standard and keyboard navigation. The 3D chart features can be effectively accessed through assistive technologies such as screen readers.

WAI-ARIA

WAI-ARIA (Accessibility Initiative – Accessible Rich Internet Applications) defines a way to increase the accessibility of web pages, dynamic content, and user interface components developed with AJAX, HTML, JavaScript, and related technologies. ARIA provides additional semantics to describe the role, state, and functionality of web components.

- img (role)
- button (role)
- region (role)
- aria-label (attribute)
- aria-hidden (attribute)
- aria-pressed (attribute)

Keyboard navigation

All the 3D chart actions can be controlled via keyboard keys. The applicable key combinations and their relative functionalities are listed below.

Interaction Keys | Description

Tab | Moves the focus to the next element in the chart.

Shift + Tab | Moves the focus to the previous element in the chart.

DownArrow | Moves the focus to the data point left side from the selected point.

UpArrow | Moves the focus to the data point right side from the selected point.

Left Arrow | Moves the focus to the next series in the chart.

Right Arrow | Moves the focus to the previous series in the chart.

ESC | Cancel the tooltip for the data point.

Enter/Space | Selects the data point in the series.

Down/Left Arrow | Moves the focus to the legend left side from the selected legend.

Up/Right Arrow | Moves the focus to the legend right side from the selected legend.

Enter/Space | Toggles the visibility of the corresponding series.

Ctrl + P | Print the chart.

Accordion

Getting started with Angular Accordion component

This section briefly explains about how to create a simple [Angular AccordionLink to the Video](#) and configure its available functionalities in Angular using Angular quickstart.

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

Dependencies

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

```
`javascript
|-- @syncfusion/ej2-angular-navigations
|-- @syncfusion/ej2-angular-base
|-- @syncfusion/ej2-base
|-- @syncfusion/ej2-navigations
`,`
```

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 Accordion 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-navigations](#) package to the application.

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

Angular compatibility compiled package(ngcc)

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

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

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

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

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

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

Registering Accordion Module

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

```
`javascript
```

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
// import the AccordionModule for the Accordion component
import { AccordionModule } from '@syncfusion/ej2-angular-navigations';
import { AppComponent } from './app.component';
@NgModule({
  //declaration of ej2-angular-navigations module into NgModule
  imports: [ BrowserModule, AccordionModule ],
  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-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-navigations/styles/material.css';
`
```

Add Accordion component

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

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

```
`typescript
import { Component, OnInit } from '@angular/core';
@Component({
  selector: 'app-root',
  // specifies the template string for the Accordion component
  template: `<ejs-accordion>
<e-accordionitems>
<e-accordionitem expanded='true'>
```

```
<ng-template #header>
<div>ASP.NET</div>
</ng-template>
<ng-template #content>
<div>Microsoft ASP.NET is a set of technologies in the Microsoft .NET Framework for building Web
applications and XML Web
services. ASP.NET pages execute on the server and generate markup such as HTML, WML, or XML that is
sent to a desktop
or mobile browser. ASP.NET pages use a compiled,event-driven programming model that improves
performance and enables
the separation of application logic and user interface.</div>
</ng-template>
</e-accordionitem>
<e-accordionitem>
<ng-template #header>
<div>ASP.NET MVC</div>
</ng-template>
<ng-template #content>
<div>The Model-View-Controller (MVC) architectural pattern separates an application into three main
components: the model,
the view, and the controller. The ASP.NET MVC framework provides an alternative to the ASP.NET Web
Forms pattern for
creating Web applications. The ASP.NET MVC framework is a lightweight, highly testable presentation
framework that
(as with Web Forms-based applications) is integrated with existing ASP.NET features, such as master
pages and membership-based
authentication.
</div>
</ng-template>
</e-accordionitem>
<e-accordionitem>
<ng-template #header>
<div>JavaScript</div>
</ng-template>
<ng-template #content>
```

<div>JavaScript (JS) is an interpreted computer programming language.It was originally implemented as part of web browsers

so that client-side scripts could interact with the user, control the browser, communicate asynchronously, and alter

the document content that was displayed.More recently, however, it has become common in both game development and

the creation of desktop applications.</div>

</ng-template>

</e-accordionitem>

</e-accordionitems>

</ejs-accordion>`

}}

export class AppComponent {

}

,

Initialize the Accordion using Items

The Accordion can be rendered by defining an array of `items`.

```
`javascript
```

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

```
@Component({
```

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

```
  template: `
```

```
    <ejs-accordion>
```

```
    <e-accordionitems>
```

```
    <e-accordionitem expanded='true'>
```

```
      <ng-template #header>
```

```
        <div>ASP.NET</div>
```

```
      </ng-template>
```

```
    <ng-template #content>
```

```
      <div>Microsoft ASP.NET is a set of technologies in the Microsoft .NET Framework for building Web applications and XML Web
```

```
services. ASP.NET pages execute on the server and generate markup such as HTML, WML, or XML that is sent to a desktop
```

```
or mobile browser. ASP.NET pages use a compiled,event-driven programming model that improves performance and enables
```

the separation of application logic and user interface.</div>

</ng-template>

</e-accordionitem>

<e-accordionitem>

<ng-template #header>

<div>ASP.NET MVC</div>

</ng-template>

<ng-template #content>

<div>The Model-View-Controller (MVC) architectural pattern separates an application into three main components: the model,

the view, and the controller. The ASP.NET MVC framework provides an alternative to the ASP.NET Web Forms pattern for

creating Web applications. The ASP.NET MVC framework is a lightweight, highly testable presentation framework that

(as with Web Forms-based applications) is integrated with existing ASP.NET features, such as master pages and membership-based

authentication.

</div>

</ng-template>

</e-accordionitem>

<e-accordionitem>

<ng-template #header>

<div>JavaScript</div>

</ng-template>

<ng-template #content>

<div>JavaScript (JS) is an interpreted computer programming language.It was originally implemented as part of web browsers

so that client-side scripts could interact with the user, control the browser, communicate asynchronously, and alter

the document content that was displayed.More recently, however, it has become common in both game development and

the creation of desktop applications.</div>

</ng-template>

</e-accordionitem>

</e-accordionitems>


```
</ejs-accordion>
```

```
,
```

```
})
```

```
export class AppComponent {
```

```
}
```

```
,
```

- Run the application in the browser using the following command.

```
`shell
```

```
npm start
```

```
,
```

The following code example depicts the way to initialize the Accordion on a single element.

Output will be as follows:

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { AccordionModule } from '@syncfusion/ej2-angular-navigations'
import { Component, ViewChild } from '@angular/core';
@Component({
  imports: [
    AccordionModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `
    <ejs-accordion>
      <e-accordionitems>
        <e-accordionitem expanded='true'>
          <ng-template #header>
            <div>ASP.NET</div>
          </ng-template>
          <ng-template #content>
            <div>Microsoft ASP.NET is a set of technologies in the
Microsoft .NET Framework for building Web applications and XML Web
services. ASP.NET pages execute on the server and generate
markup such as HTML, WML, or XML that is sent to a desktop
or mobile browser. ASP.NET pages use a compiled, event-driven
programming model that improves performance and enables
the separation of application logic and user interface.</div>
          </ng-template>
        </e-accordionitem>
        <e-accordionitem>
          <ng-template #header>
            <div>ASP.NET MVC</div>
          </ng-template>
          <ng-template #content>
```

```

        <div>The Model-View-Controller (MVC) architectural pattern
        separates an application into three main components: the model,
        the view, and the controller. The ASP.NET MVC framework
        provides an alternative to the ASP.NET Web Forms pattern for
        creating Web applications. The ASP.NET MVC framework is a
        lightweight, highly testable presentation framework that
        (as with Web Forms-based applications) is integrated with
        existing ASP.NET features, such as master pages and membership-based
        authentication.
    </div>
</ng-template>
</e-accordionitem>
<e-accordionitem>
    <ng-template #header>
        <div>JavaScript</div>
    </ng-template>
    <ng-template #content>
        <div>JavaScript (JS) is an interpreted computer programming
        language.It was originally implemented as part of web browsers
        so that client-side scripts could interact with the user,
        control the browser, communicate asynchronously, and alter
        the document content that was displayed.More recently,
        however, it has become common in both game development and
        the creation of desktop applications.</div>
    </ng-template>
</e-accordionitem>
</e-accordionitems>
</ejs-accordion>

    })
    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));

```

Initialize the Accordion using HTML elements

The Accordion component can be rendered based on the given HTML element using `<ejs-accordion>`.

You need to follow the below structure of HTML elements to render the Accordion inside the `<ejs-accordion>` tag.

```
`html`
```

```
<ejs-accordion> --> Root Accordion Element
```

```
<div> --> Accordion Item Container
```

```
<div> --> Accordion Header Container
```

```
<div> </div> --> Accordion Header
```

```

</div>
<div> --> Accordion Panel Container
<div> </div> --> Accordion Content
</div>
</div>
</ejs-accordion>
,

```

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { AccordionModule } from '@syncfusion/ej2-angular-navigations'
import { Component, ViewChild } from '@angular/core';
@Component({
  imports: [
    AccordionModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-accordion>
    <div>
      <div>
        <div> ASP.NET </div>
      </div>
      <div>
        <div> Microsoft ASP.NET is a set of technologies in the Microsoft
.NET Framework for building Web applications
and XML Web services </div>
      </div>
    </div>
    <div>
      <div>
        <div> ASP.NET MVC </div>
      </div>
      <div>
        <div>The Model-View-Controller (MVC) architectural pattern
separates an application into three main components:
the model, the view, and the controller </div>
      </div>
    </div>
    <div>
      <div>
        <div> JavaScript </div>
      </div>
      <div>
        <div>JavaScript (JS) is an interpreted computer programming
language.It was originally implemented as part
of web browsers so that client-side scripts could interact
with the user, control the browser </div>
      </div>
    </div>
  </ejs-accordion>`

```

```

    })
    export class AppComponent {
    }

```

MAIN.TS

```

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

```

You can add the custom class into Accordion component using [cssClass](#) property which is used to customize the Accordion component.

See Also

- [How to load accordion items dynamically](#)

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

Expand mode in Angular Accordion component

The Accordion supports the two listed types of expand modes while expanding or collapsing the item.

- Single
- Multiple

Single

The property enables to expand only one Accordion item at a time. If you expand any new item, the previously expanded one is collapsed and new item changed to expanded state.

You can also choose which accordion pane is expanded state at initial rendering by enabling the [expanded](#) property on accordion items.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { AccordionModule } from '@syncfusion/ej2-angular-navigations'
import { Component, ViewChild } from '@angular/core';
@Component({
  imports: [
    AccordionModule
  ],
  standalone: true,
  selector: 'app-container',
  template: `
    <ejs-accordion expandMode='Single'>
      <e-accordionitems>
        <e-accordionitem expanded='true'>
          <ng-template #header>
            <div>ASP.NET</div>

```

```

        </ng-template>
        <ng-template #content>
            <div>Microsoft ASP.NET is a set of technologies in the
Microsoft .NET Framework for building Web applications and XML Web
            services. ASP.NET pages execute on the server and generate
markup such as HTML, WML, or XML that is sent to a desktop
            or mobile browser. ASP.NET pages use a compiled, event-driven
programming model that improves performance and enables
            the separation of application logic and user interface.</div>
        </ng-template>
    </e-accordionitem>
    <e-accordionitem>
        <ng-template #header>
            <div>ASP.NET MVC</div>
        </ng-template>
        <ng-template #content>
            <div>The Model-View-Controller (MVC) architectural pattern
separates an application into three main components: the model,
            the view, and the controller. The ASP.NET MVC framework
provides an alternative to the ASP.NET Web Forms pattern for
            creating Web applications. The ASP.NET MVC framework is a
lightweight, highly testable presentation framework that
            (as with Web Forms-based applications) is integrated with
existing ASP.NET features, such as master pages and membership-based
            authentication.
        </div>
        </ng-template>
    </e-accordionitem>
    <e-accordionitem>
        <ng-template #header>
            <div>JavaScript</div>
        </ng-template>
        <ng-template #content>
            <div>JavaScript (JS) is an interpreted computer programming
language.It was originally implemented as part of web browsers
            so that client-side scripts could interact with the user,
control the browser, communicate asynchronously, and alter
            the document content that was displayed.More recently,
however, it has become common in both game development and
            the creation of desktop applications.</div>
        </ng-template>
    </e-accordionitem>
</e-accordionitems>
</ejs-accordion>

    ))
    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));

```

Multiple

Default [expandMode](#) of the Accordion is **Multiple**. It enables you to expand more than one Accordion item at a time. Expand/collapse action can also be toggled by clicking on it again. For example, expanded item is collapsed when you click on it again.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { AccordionModule } from '@syncfusion/ej2-angular-navigations'
import { Component, ViewChild } from '@angular/core';
@Component({
  imports: [
    AccordionModule
  ],
  standalone: true,
  selector: 'app-container',
  template: `
    <ejs-accordion expandMode='Multiple'>
      <e-accordionitems>
        <e-accordionitem expanded='true'>
          <ng-template #header>
            <div>ASP.NET</div>
          </ng-template>
          <ng-template #content>
            <div>Microsoft ASP.NET is a set of technologies in the
Microsoft .NET Framework for building Web applications and XML Web
services. ASP.NET pages execute on the server and generate
markup such as HTML, WML, or XML that is sent to a desktop
or mobile browser. ASP.NET pages use a compiled, event-driven
programming model that improves performance and enables
the separation of application logic and user interface.</div>
          </ng-template>
        </e-accordionitem>
        <e-accordionitem>
          <ng-template #header>
            <div>ASP.NET MVC</div>
          </ng-template>
          <ng-template #content>
            <div>The Model-View-Controller (MVC) architectural pattern
separates an application into three main components: the model,
the view, and the controller. The ASP.NET MVC framework
provides an alternative to the ASP.NET Web Forms pattern for
creating Web applications. The ASP.NET MVC framework is a
lightweight, highly testable presentation framework that
(as with Web Forms-based applications) is integrated with
existing ASP.NET features, such as master pages and membership-based
authentication.
          </div>
          </ng-template>
        </e-accordionitem>
        <e-accordionitem>
          <ng-template #header>
            <div>JavaScript</div>
          </ng-template>
          <ng-template #content>
```

```

        <div>JavaScript (JS) is an interpreted computer programming
        language.It was originally implemented as part of web browsers
        so that client-side scripts could interact with the user,
        control the browser, communicate asynchronously, and alter
        the document content that was displayed.More recently,
        however, it has become common in both game development and
        the creation of desktop applications.</div>
    </ng-template>
</e-accordionitem>
</e-accordionitems>
</ejs-accordion>

})
export class AppComponent {
}

```

MAIN.TS

```

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

```

See Also

- [How to keep single pane open always](#)

Accessibility in Angular Accordion component

The Accordion component has been designed keeping in mind the [WAI-ARIA](#) specifications, by applying the prompt WAI-ARIA roles, states, and properties along with the keyboard support. Thus, making it usable for people who use assistive WAI-ARIA Accessibility supports that is achieved through the attributes like `aria-labelledby`. It helps to provides information about the elements in a document for assistive technology. The component implements the keyboard navigation support by following the [WAI-ARIA practices](#) and tested in major screen readers.

The accessibility compliance for the Accordion component is outlined below.

| Accessibility Criteria | Compatibility |

| -- | -- |

| [WCAG 2.2](#) Support | |

| [Section 508](#) Support | |

| Screen Reader Support | |

| Right-To-Left Support | |

| Color Contrast | |

| Mobile Device Support | |

| Keyboard Navigation Support | |

| [Accessibility Checker](#) Validation | |

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

<style>

.post .post-content img {

display: inline-block;

margin: 0.5em 0;

}

</style>

<div> - All features of the component meet the requirement.</div>

<div> - Some features of the component do not meet the requirement.</div>

<div> - The component does not meet the requirement.</div>

ARIA attributes

| Roles and Attributes | Functionalities

|

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

| role | **Button:** Attribute is set to the Accordion header elements to indicate that the element can be used to toggle the visibility of the associated content section, describing the actual role of the element.
 Region: Attribute is set to the Accordion panel elements to create a landmark region that contains the currently expanded accordion panel, describing the actual role of the element.
|

| aria-labelledby | Attribute is set to content (panel) and it points to the corresponding Accordion header. |

| aria-controls | Attribute is set to the header and it points to the corresponding Accordion content. |

| aria-expanded | Attribute is set to the Accordion header elements to indicates the expand state of the Accordion Item. Default value of this attribute is false. If an item is expanded, the attribute value changes to 'true'. |

| aria-hidden | Attribute is set to the Accordion panel elements to indicates the content visible state of the Accordion Item. Default value of this attribute is `true`. If an item content is visible, the attribute value changes to `false`. |

| aria-disabled | It indicates the disabled state of the Accordion and its items. |

Keyboard interaction

Keyboard navigation is enabled by default. Possible keys are:

Key	Description
----- -----	-----
Space or Enter	When focus is on the Accordion header, click on the focused element makes the element to expand and collapse.
Down Arrow	Focus the next Accordion header.
Up Arrow	Focus the previous Accordion header.
Home	Focus the first Accordion header.
End	Focus the last Accordion header.
Tab	To Move focus through the interactive elements.
Shift + Tab	To Move focus through the interactive elements.

Ensuring accessibility

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

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

See also

- [Accessibility in Syncfusion Angular components](#)

Style in Angular Accordion component

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

Customizing Accordion

Use the following CSS to customize the Accordion.

```
`CSS
.e-accordion {
border: 5px solid rgb(173, 255, 47);
}
```

Customizing the list items

Use the following CSS to customize the items of Accordion.

```
`CSS
.e-accordion .e-acrdn-item {
text-align: center;
color: pink;
background-color: #2fa1ff;
}
`
```

Customizing Accordion's header

Use the following CSS to customize the header of Accordion control.

```
`CSS
.e-accordion .e-acrdn-item.e-select > .e-acrdn-header {
background: #2fa1ff !important;
justify-content: center;
}
`
```

Customizing Accordion's expand and collapse icons

Use the following CSS to customize the expand and collapse icons of Accordion control.

```
`CSS
.e-accordion .e-acrdn-item .e-acrdn-header .e-toggle-icon .e-icons {
color: pink;
}
`
```

Customizing the hover state of Accordion control

Use the following CSS to customize the accordion item when hovering.

```
`CSS
.e-accordion .e-acrdn-item .e-acrdn-header:hover {
border: 2px solid gray;
}
`
```

Customizing selected item of Accordion control

Use the following CSS to customize the selected accordion item.

```
`CSS
```

```
.e-accordion .e-acrdn-item.e-select.e-active>.e-acrdn-header,
.e-accordion .e-acrdn-item.e-select.e-item-focus>.e-acrdn-header {
background-color: rgb(0, 15, 100) !important;
}
`
```

Use the following CSS to customize the selected accordion item text.

```
`CSS
.e-accordion .e-acrdn-item.e-select.e-active>.e-acrdn-header .e-acrdn-header-content,
.e-accordion .e-acrdn-item.e-select.e-item-focus>.e-acrdn-header .e-acrdn-header-content {
color: #2fa1ff !important;
}
`
```

How To

Set the nested accordion in Angular Accordion component

You can render Accordion components inside the parent Accordion content using Angular **ng-template**. Through this, we can add content as Accordion components directly with all their functionalities to our Accordion. We need to use **ng-template** inside the each **e-accordionitem** tag with **#content** attribute, which is mandatory to render content. And now use **ng-template** tag with select attribute of id or class name for mapping required content.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { AccordionModule } from '@syncfusion/ej2-angular-navigations'
import { Component, ViewChild } from '@angular/core';
import { AccordionComponent } from '@syncfusion/ej2-angular-navigations';
import { ExpandEventArgs, Accordion, AccordionClickArgs } from
 '@syncfusion/ej2-navigations';
@Component({
imports: [
    AccordionModule
],
standalone: true,
selector: 'app-container',
template: `
    <ejs-accordion>
        <e-accordionitems>
            <e-accordionitem expanded="true">
                <ng-template #header>
                    <div>Video</div>
                </ng-template>
                <ng-template #content>
                    <ejs-accordion>
                        <e-accordionitems>
                            <e-accordionitem>
                                <ng-template #header>
```

```
<div>Video Track 1</div>  
  </ng-template>  
</e-accordionitem>  
<e-accordionitem>  
  <ng-template #header>  
    <div>Video Track 2</div>  
  </ng-template>  
</e-accordionitem>  
</e-accordionitems>  
</ejs-accordion>  
</ng-template>  
</e-accordionitem>  
<e-accordionitem>  
  <ng-template #header>  
    <div>Music</div>  
  </ng-template>  
  <ng-template #content>  
    <ejs-accordion>  
      <e-accordionitems>  
        <e-accordionitem>  
          <ng-template #header>  
            <div>Music Track 1</div>  
          </ng-template>  
        </e-accordionitem>  
        <e-accordionitem>  
          <ng-template #header>  
            <div>Music Track 2</div>  
          </ng-template>  
        </e-accordionitem>  
        <e-accordionitem>  
          <ng-template #header>  
            <div>Music New</div>  
          </ng-template>  
          <ng-template #content>  
            <ejs-accordion>  
              <e-accordionitems>  
                <e-accordionitem>  
                  <ng-template #header>  
                    <div>New Track 1</div>  
                  </ng-template>  
                </e-accordionitem>  
                <e-accordionitem>  
                  <ng-template #header>  
                    <div>New Track 2</div>  
                  </ng-template>  
                </e-accordionitem>  
              </e-accordionitems>  
            </ejs-accordion>  
          </ng-template>  
        </e-accordionitem>  
      </e-accordionitems>  
    </ejs-accordion>  
  </ng-template>  
</e-accordionitem>  
</e-accordionitems>  
</ejs-accordion>  
</ng-template>  
</e-accordionitem>  
<e-accordionitem>  
  <ng-template #header>  
    <div>Images</div>
```

```

        </ng-template>
        <ng-template #content>
          <ejs-accordion>
            <e-accordionitems>
              <e-accordionitem>
                <ng-template #header>
                  <div>Track 1</div>
                </ng-template>
              </e-accordionitem>
              <e-accordionitem>
                <ng-template #header>
                  <div>Track 2</div>
                </ng-template>
              </e-accordionitem>
            </e-accordionitems>
          </e-accordion>
        </ng-template>
      </e-accordionitem>
    </e-accordionitems>
  </ejs-accordion>
}
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));

```

Load content through post in Angular Accordion component

Accordion supports to load external contents through **AJAX** library. Refer the below steps.

- Import the **Ajax** module from **ej2-base** and initialize with URL path.
- Get data from the **Ajax Success** event to initialize Accordion with retrieved external path data.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { AccordionModule } from '@syncfusion/ej2-angular-navigations'
import { Component, ViewChild } from '@angular/core';
import { Ajax } from '@syncfusion/ej2-base';
import { AccordionComponent } from '@syncfusion/ej2-angular-navigations';
@Component({
  imports: [
    AccordionModule
  ],
  standalone: true,
  selector: 'app-container',
  template: `
    <div id="acrdnContnet1" style="display:none">

```

```

        <ul style="margin : 0px;padding:0px 16px; list-style-type: none">
            <li>Testing</li>
            <li>Development</li>
        </ul>
    </div>
    <div id="acrdnContnet2" style="display:none">
        <ul style="margin : 0px;padding:0px 16px; list-style-type: none">
            <li>Mobile</li>
            <li>Web</li>
        </ul>
    </div>
    <ejs-accordion #acrdnInstance>
        <e-accordionitems>
            <e-accordionitem header='Department' content = '#acrdnContnet1'></e-
accordionitem>
            <e-accordionitem header='Platform' content = '#acrdnContnet2'></e-
accordionitem>
            <e-accordionitem header='Employee Details'></e-accordionitem>
        </e-accordionitems>
    </ejs-accordion>
}
}
export class AppComponent {
    @ViewChild('acrdnInstance') acrdnInstance?: AccordionComponent;
    public contentData?: string;
    ngOnInit() {
        let ajax: Ajax = new Ajax('./ajax.html', 'GET', true);
        ajax.send().then();
        ajax.onSuccess = (data: string): void => {
            (this.acrdnInstance as AccordionComponent).items[2].content = data;
            (this.acrdnInstance as AccordionComponent).refresh();
        };
    }
}

```

MAIN.TS

```

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

```

Set custom animation in Angular Accordion component

Accordion supports custom animations for both expand and collapse actions from the provided animation option of **Animation** library.

The [animation](#) property also allows you to set [easing](#), [duration](#), and various other effects of your choice.

Default animation is given as **SlideDown** for expanding the panel using [expand](#) animation property and **SlideUp** for collapsing the panel using [collapse](#) animation property. You can also disable the animation by setting animation [effect](#) as **none**.

The sample demonstrates some types of animation that suits for Accordion. You can check all the animation effects [here](#).

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { AccordionModule } from '@syncfusion/ej2-angular-navigations'
import { DropDownListModule } from '@syncfusion/ej2-angular-dropdowns'
import { Component, ViewChild } from '@angular/core';
import { Effect } from '@syncfusion/ej2-base';
import { DropDownListComponent } from '@syncfusion/ej2-angular-dropdowns';
import { AccordionComponent } from '@syncfusion/ej2-angular-navigations';
@Component({
  imports: [
    AccordionModule, DropDownListModule
  ],
  standalone: true,
  selector: 'app-container',
  template: `
<div style='padding-top: 25px'>
<div class='row'>
<div class="col-xs-6 col-sm-6 col-lg-6 col-md-6">
  <label> Expand Animation </label></div>
  <div class="col-xs-6 col-sm-6 col-lg-6 col-md-6">
    <div class='custom_drop'><ejs-dropdownlist #expandAnimation
(change)='expandAnimationChange()' index='0' [dataSource]='expandAni'
placeholder='Expand Animation'></ejs-dropdownlist></div>
</div></div>
<div class='row'>
<div class="col-xs-6 col-sm-6 col-lg-6 col-md-6">
<label> Collapse Animation </label></div>
<div class="col-xs-6 col-sm-6 col-lg-6 col-md-6">
  <div class='custom_drop'><ejs-dropdownlist #collapseAnimation
(change)='collapseAnimationChange()' index='1' [dataSource]='expandAni'
placeholder='Collapse Animation'></ejs-dropdownlist></div>
</div></div>
<div style='padding-top: 25px'>
<ejs-accordion #acrInInstance>
  <e-accordionitems>
    <e-accordionitem expanded='true'>
      <ng-template #header>
        <div>ASP.NET</div>
      </ng-template>
      <ng-template #content>
        <div>Microsoft ASP.NET is a set of technologies in the
Microsoft .NET Framework for building Web applications and XML Web
services. ASP.NET pages execute on the server and generate
markup such as HTML, WML, or XML that is sent to a desktop
or mobile browser. ASP.NET pages use a compiled, event-driven
programming model that improves performance and enables
the separation of application logic and user interface.</div>
      </ng-template>
    </e-accordionitem>
    <e-accordionitem>
      <ng-template #header>
        <div>ASP.NET MVC</div>
      </ng-template>
      <ng-template #content>

```

```

        <div>The Model-View-Controller (MVC) architectural pattern
        separates an application into three main components: the model,
        the view, and the controller. The ASP.NET MVC framework
        provides an alternative to the ASP.NET Web Forms pattern for
        creating Web applications. The ASP.NET MVC framework is a
        lightweight, highly testable presentation framework that
        (as with Web Forms-based applications) is integrated with
        existing ASP.NET features, such as master pages and membership-based
        authentication.
    </div>
    </ng-template>
</e-accordionitem>
<e-accordionitem>
    <ng-template #header>
        <div>JavaScript</div>
    </ng-template>
    <ng-template #content>
        <div>JavaScript (JS) is an interpreted computer programming
        language.It was originally implemented as part of web browsers
        so that client-side scripts could interact with the user,
        control the browser, communicate asynchronously, and alter
        the document content that was displayed.More recently,
        however, it has become common in both game development and
        the creation of desktop applications.</div>
    </ng-template>
</e-accordionitem>
</e-accordionitems>
</ejs-accordion>
</div></div>

```

```

    })
    export class AppComponent {
        @ViewChild('acrdnInstance') acrdnInstance?: AccordionComponent;
        @ViewChild('expandAnimation') expandInstance?: DropDownListComponent;
        @ViewChild('collapseAnimation') collapseInstance?: DropDownListComponent;
        public expandAni: string[] = ['SlideDown', 'SlideUp', 'FadeIn',
        'FadeOut', 'FadeZoomIn', 'FadeZoomOut', 'ZoomIn', 'ZoomOut', 'None'];
        public expandAnimationChange(): void {
            (this.acrdnInstance as AccordionComponent).animation.expand = { effect:
            this.expandInstance?.value as Effect }
        }
        public collapseAnimationChange(): void {
            (this.acrdnInstance as AccordionComponent).animation.collapse = {
            effect: this.collapseInstance?.value as Effect }
        }
    }
}

```

MAIN.TS

```

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

```


Expand or collapse accordion items on checkbox click in Accordion component

By default, accordion items expand or collapse by clicking the accordion item header or clicking expand/collapse icon in accordion header.

You can also expand or collapse the accordion items through external button click. In the following example, when you change the checkbox provided then the accordion items will expand/collapse accordingly. This requirement can be achieved with the help of accordion's [click](#), [expanding](#) events, [expandItem](#) public method and checkbox's [change](#) event.

APP.COMPONENT.HTML

```
<div class="control-section accordion-control-section">
  <div class="e-sample-resize-container">
    <!-- Render the Accoridon Component -->
    <ejs-accordion
      #element
      (expanding)="expanding($event) "
      (clicked)="onClick($event) "
    >
      <e-accordionitems>
        <e-accordionitem>
          <ng-template #header>
            <div class="ib">
              <ejs-checkbox
                #checkbox1
                (change)="changeHandler1() "
              ></ejs-checkbox>
            </div>
            <div class="ib">ASP.NET</div>
          </ng-template>
          <ng-template #content>
            <div>
              Microsoft ASP.NET is a set of technologies in the Microsoft
.NET
              Framework for building Web applications and XML Web services.
              ASP.NET pages execute on the server and generate markup such
as
              HTML, WML, or XML that is sent to a desktop or mobile
browser.
              ASP.NET pages use a compiled,event-driven programming model
that
              improves performance and enables the separation of
application
              logic and user interface.
            </div>
          </ng-template>
        </e-accordionitem>
        <e-accordionitem>
          <ng-template #header>
            <div class="ib">
              <ejs-checkbox
                #checkbox2
                (change)="changeHandler2() "
              ></ejs-checkbox>
            </div>
            <div class="ib">ASP.NET MVC</div>
```

```

        </ng-template>
        <ng-template #content>
            <div>
                The Model-View-Controller (MVC) architectural pattern
separates an
                application into three main components: the model, the view,
and
                the controller. The ASP.NET MVC framework provides an
alternative
                to the ASP.NET Web Forms pattern for creating Web
applications.
                The ASP.NET MVC framework is a lightweight, highly testable
applications)
                presentation framework that (as with Web Forms-based
pages
                is integrated with existing ASP.NET features, such as master
                and membership-based authentication.
            </div>
        </ng-template>
    </e-accordionitem>
<e-accordionitem>
    <ng-template #header>
        <div class="ib">
            <ejs-checkbox
                #checkbox3
                (change)="changeHandler3()"
            ></ejs-checkbox>
        </div>
        <div class="ib">JavaScript</div>
    </ng-template>
    <ng-template #content>
        <div>
            JavaScript (JS) is an interpreted computer programming
language.It
            was originally implemented as part of web browsers so that
            client-side scripts could interact with the user, control the
            browser, communicate asynchronously, and alter the document
            content that was displayed.More recently, however, it has
become
            common in both game development and the creation of desktop
            applications.
        </div>
    </ng-template>
</e-accordionitem>
</e-accordionitems>
</ejs-accordion>
</div>
</div>

```

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { AccordionModule } from '@syncfusion/ej2-angular-navigations';

```

```
import { Component, ViewEncapsulation, Inject, ViewChild } from
"@angular/core";
import {
    ExpandEventArgs,
    Accordion,
    AccordionClickArgs
} from "@syncfusion/ej2-navigations";
import { closest } from "@syncfusion/ej2-base";
import { AccordionComponent, AccordionItemModel } from "@syncfusion/ej2-
angular-navigations";
import { CheckBoxComponent, CheckBoxModule } from "@syncfusion/ej2-angular-
buttons";
@Component({
    imports: [
        AccordionModule, CheckBoxModule
    ],
    standalone: true,
    selector: "app-container",
    templateUrl: "./app.component.html"
})
export class AppComponent {
    @ViewChild("element") acrdnInstance?: AccordionComponent;
    @ViewChild("checkbox1") chk1Instance?: CheckBoxComponent;
    @ViewChild("checkbox2") chk2Instance?: CheckBoxComponent;
    @ViewChild("checkbox3") chk3Instance?: CheckBoxComponent;
    public clickEventArgs?: Event;
    public expanding(e: ExpandEventArgs) {
        if (this.clickEventArgs) {
            let header = closest(
                this.clickEventArgs.target as Element,
                ".e-acrdn-header"
            );
            let checkboxEle = closest(
                this.clickEventArgs.target as Element,
                ".e-checkbox-wrapper"
            );
            if (header && !checkboxboxEle) {
                e.cancel = true;
                return;
            }
            let index = (this.acrdnInstance as
AccordionComponent).items.indexOf(e.item as AccordionItemModel);
            if (index == 0 && !(this.chk1Instance as CheckBoxComponent).checked) {
                e.cancel = true;
                return;
            }
            if (index == 1 && !(this.chk2Instance as CheckBoxComponent).checked) {
                e.cancel = true;
                return;
            }
            if (index == 2 && !(this.chk3Instance as CheckBoxComponent).checked) {
                e.cancel = true;
                return;
            }
        }
    }
    public onClick(e: any) {
```

```

    this.clickEventArgs = e.originalEvent;
  }
  public changeHandler1() {
    this.clickEventArgs = null as any;
    (this.acrdnInstance as AccordionComponent).expandItem(true, 0);
  }
  public changeHandler2() {
    this.clickEventArgs = null as any;
    (this.acrdnInstance as AccordionComponent).expandItem(true, 1);
  }
  public changeHandler3() {
    this.clickEventArgs = null as any;
    (this.acrdnInstance as AccordionComponent).expandItem(true, 2);
  }
}

```

MAIN.TS

```

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

```

To keep single pane open always in Angular Accordion component

By default, all Accordion panels are collapsible. You can customize the Accordion to keep one panel as expanded state always. This is applicable for **Single** expand mode.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { AccordionModule } from '@syncfusion/ej2-angular-navigations'
import { Component, ViewChild } from '@angular/core';
import { AccordionComponent } from '@syncfusion/ej2-angular-navigations';
import { ExpandEventArgs, AccordionClickArgs } from '@syncfusion/ej2-
navigations';
@Component({
  imports: [
    AccordionModule
  ],
  standalone: true,
  selector: 'app-container',
  template: `
    <ejs-accordion #element expandMode='Single'
    (expanding)="beforeExpand($event)" (clicked)="clicked($event)">
      <e-accordionitems>
        <e-accordionitem expanded='true'>
          <ng-template #header>
            <div>ASP.NET</div>
          </ng-template>
          <ng-template #content>
            <div>Microsoft ASP.NET is a set of technologies in the
Microsoft .NET Framework for building Web applications and XML Web
services. ASP.NET pages execute on the server and generate
markup such as HTML, WML, or XML that is sent to a desktop

```

```

        or mobile browser. ASP.NET pages use a compiled, event-driven
programming model that improves performance and enables
        the separation of application logic and user interface.into three main components: the model,
        the view, and the controller. The ASP.NET MVC framework
provides an alternative to the ASP.NET Web Forms pattern for
        creating Web applications. The ASP.NET MVC framework is a
lightweight, highly testable presentation framework that
        (as with Web Forms-based applications) is integrated with
existing ASP.NET features, such as master pages and membership-based
        authentication.
    </div>
    </ng-template>
</e-accordionitem>
<e-accordionitem>
    <ng-template #header>
        <div>JavaScript</div>
    </ng-template>
    <ng-template #content>
        <div>JavaScript (JS) is an interpreted computer programming
language. It was originally implemented as part of web browsers
        so that client-side scripts could interact with the user,
control the browser, communicate asynchronously, and alter
        the document content that was displayed. More recently,
however, it has become common in both game development and
        the creation of desktop applications.</div>
    </ng-template>
</e-accordionitem>
</e-accordionitems>
</ejs-accordion>

))
export class AppComponent {
    @ViewChild('element') acrdnInstance?: AccordionComponent;
    public clickEle?: HTMLElement;
    public clicked(e: AccordionClickArgs) {
        this.clickEle = ((e.originalEvent as Event).target as Element).closest(
            '.e-acrdn-header'
        ) as HTMLElement;
    }
    public beforeExpand(e: ExpandEventArgs): void {
        const childrenArray = Array.from((this.acrdnInstance as
AccordionComponent).element.children);
        const expandCount = childrenArray.filter(el => el.classList.contains('e-
selected')).length;
        let ele = childrenArray.filter(el => el.classList.contains('e-
selected'))[0];
        if (ele) {
            ele = ele.firstChild as Element;

```

```

    }
    if (expandCount === 1 && ele === this.clickEle) {
      e.cancel = true;
    }
  }
}

```

MAIN.TS

```

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

```

Create wizard using accordion in Angular Accordion component

Accordion items can be disabled dynamically by passing the index and boolean value with the [enableItem](#) method and also dynamically expand the item using [expandItem](#) method.

The below Wizard sample is designed for Online Shopping model. In this, each Accordion item is integrated with required components to fill the details and designed for getting user details and making payment at the end. Each field is provided with validation for all mandatory option to enable/disable to next Accordion. In below sample, accordion items can be disabled dynamically with [enableItem](#) method using [created](#) event.

APP.COMPONENT.HTML

```

<ejs-dialog #alertDlg header='Alert' width=200 isModal=true content=''
[target]='dlgTarget' [buttons]='dlgButtons'
(created)='dlgCreated()'></ejs-dialog>
<ejs-accordion #accordion (created)='acrdnCreated()'>
  <e-accordionitems>
    <e-accordionitem expanded=true header='Sign In'>
      <ng-template #content>
        <div id="Sign_In_Form" style="padding: 3px 0">
          <form id="formId">
            <div class="form-group">
              <div class="e-float-input">
                <input type="text" id="email" name="Email"
required="" #emailRef/>
                <span class="e-float-line"></span>
                <label class="e-float-text"
for="email">Email</label>
              </div>
              <div class="e-float-input">
                <input id="password" type="password"
name="Password" required="" #passwordRef/>
                <span class="e-float-line"></span>
                <label class="e-float-text"
for="password">Password</label>
              </div>
            </div>
          </form>
          <div style="text-align: center">
            <button class='e-btn' id="Continue_Btn"
(click)='btnClick($event)'>Continue</button>

```

```

        <div id="err1" #error1>* Please fill all fields</div>
    </div>
</div>
</ng-template>
</e-accordionitem>
<e-accordionitem header='Delivery Address'>
    <ng-template #content>
        <div id="Address_Fill" style="padding: 3px 0">
            <form id="formId_Address">
                <div class="form-group">
                    <div class="e-float-input">
                        <input type="text" id="name" name="Name"
required="" #nameRef/>
                        <span class="e-float-line"></span>
                        <label class="e-float-text"
for="name">Name</label>
                    </div>
                </div>
                <div class="form-group">
                    <div class="e-float-input">
                        <input type="text" id="address"
name="Address" required="" #addressRef/>
                        <span class="e-float-line"></span>
                        <label class="e-float-text"
for="address">Address</label>
                    </div>
                </div>
                <div class="form-group">
                    <ejs-numerictextbox #mobile format='0'
placeholder='Mobile' floatLabelType='Auto'
[showSpinButton]=false></ejs-numerictextbox>
                </div>
            </form>
            <div style="text-align: center">
                <button class='e-btn' id="Continue_BtnAdr"
(click)='btnClick($event)'>Continue</button>
                <div id="err2" #error2>* Please fill all fields</div>
            </div>
        </div>
    </ng-template>
</e-accordionitem>
<e-accordionitem header='Card Details'>
    <ng-template #content>
        <div id="Card_Fill" style="padding: 3px 0;">
            <div class="col-xs-6 col-sm-6 col-lg-6 col-md-6">
                <div class="form-group">
                    <ejs-numerictextbox #cardNo format='0'
placeholder='Card No' floatLabelType='Auto'
[showSpinButton]=false></ejs-numerictextbox>
                </div>
            </div>
            <div class="col-xs-6 col-sm-6 col-lg-6 col-md-6">
                <div class="form-group">
                    <div class="e-float-input">
                        <input type="text" id="cardHolder"
name="cardHolder" required="" #cardHolderRef/>
                        <span class="e-float-line"></span>

```

```

                                <label class="e-float-text"
for="cardHolder">CardHolder Name</label>
                                </div>
                                </div>
                                </div>
                                <div class="col-xs-6 col-sm-6 col-lg-6 col-md-6">
                                    <ejs-datepicker #date width='100%' format='MM/yyyy'
placeholder='Expiry Date'
                                    floatLabelType='Auto'></ejs-datepicker>
                                </div>
                                <div class="col-xs-6 col-sm-6 col-lg-6 col-md-6">
                                    <div class="form-group">
                                        <ejs-numerictextbox #cvv format='0'
placeholder='CVV' floatLabelType='Auto'
                                        [showSpinButton]=false></ejs-numerictextbox>
                                    </div>
                                </div>
                                <div style="text-align: center">
                                    <button class='e-btn' id="Back_Btn"
(click)='btnClick($event)'>Back</button>
                                    <button class='e-btn' id="Save_Btn"
(click)='btnClick($event)'>Save</button>
                                    <div id="err3" #error1>* Please fill all fields</div>
                                </div>
                                </div>
                                </ng-template>
                                </e-accordionitem>
                                </e-accordionitems>
                                </ejs-accordion>

```

APP.COMPONENT.TS

```

import { BrowserModule } from '@angular/platform-browser'
import { NgModule, CUSTOM_ELEMENTS_SCHEMA } from '@angular/core'
import { DialogAllModule } from '@syncfusion/ej2-angular-popups'
import { DatePickerAllModule } from '@syncfusion/ej2-angular-calendars'
import { NumericTextBoxAllModule } from '@syncfusion/ej2-angular-inputs'
import { AccordionModule } from '@syncfusion/ej2-angular-navigations'
import { Component, ViewChild, OnInit } from '@angular/core';
import { enableRipple, isNullOrUndefined as isNOU } from '@syncfusion/ej2-
base';
import { DialogComponent } from '@syncfusion/ej2-angular-popups';
import { DatePicker } from '@syncfusion/ej2-calendars';
import { NumericTextBoxComponent } from '@syncfusion/ej2-angular-inputs';
import { AccordionComponent, AccordionItemsDirective, AccordionItemDirective
} from '@syncfusion/ej2-angular-navigations';
import { DatePickerComponent } from '@syncfusion/ej2-angular-calendars';
import { ElementRef } from '@angular/core';
enableRipple(true);
@Component({
imports: [

        DialogAllModule,
        AccordionModule,
        DatePickerAllModule,
        NumericTextBoxAllModule

```



```

    ],
    standalone: true,
    selector: 'app-container',
    templateUrl: './app.component.html'
  })
  export class AppComponent implements OnInit {
    @ViewChild('alertDlg') alertDlg?: DialogComponent;
    @ViewChild('accordion') accordion?: AccordionComponent;
    @ViewChild('mobile') mobile?: NumericTextBoxComponent;
    @ViewChild('cardNo') cardNo?: NumericTextBoxComponent;
    @ViewChild('date') expiry?: DatePickerComponent;
    @ViewChild('cvv') cvv?: NumericTextBoxComponent;
    @ViewChild('emailRef') emailReference!: ElementRef<HTMLInputElement>;
    @ViewChild('passwordRef') passwordRef!: ElementRef<HTMLInputElement>;
    @ViewChild('nameRef') nameRef!: ElementRef<HTMLInputElement>;
    @ViewChild('addressRef') addressRef!: ElementRef<HTMLInputElement>;
    @ViewChild('cardHolderRef') cardHolderRef!: ElementRef<HTMLInputElement>;
    @ViewChild('error1') error1!: ElementRef<HTMLDivElement>;
    @ViewChild('error2') error2!: ElementRef<HTMLDivElement>;
    @ViewChild('error3') error3!: ElementRef<HTMLDivElement>;
    public dlgTarget?: HTMLElement;
    public dlgButtons?: Object[];
    public success: string = 'Your payment successfully processed';
    public email_alert: string = 'Enter valid email address';
    public mobile_alert: string = 'Mobile number length should be 10';
    public card_alert: string = 'Card number length should be 16';
    public cvv_alert: string = 'CVV number length should be 3';
    public ngOnInit(): void {
      this.dlgTarget = document.body;
      this.dlgButtons = [{
        buttonModel: { content: 'Ok', isPrimary: true },
        click: () => {
          this.alertDlg?.hide();
          if ((this.accordion as AccordionComponent).expandedIndices[0] === 2
          && this.alertDlg?.content === this.success) {
            (this.accordion as AccordionComponent).enableItem(0, true);
            (this.accordion as AccordionComponent).enableItem(1, false);
            (this.accordion as AccordionComponent).enableItem(2, false);
            (this.accordion as AccordionComponent).expandItem(true, 0);
          }
        }
      }
    ];
  }
  public dlgCreated(): void {
    this.alertDlg?.hide();
  }
  public acrdnCreated(): void {
    (this.accordion as AccordionComponent).enableItem(1, false);
    (this.accordion as AccordionComponent).enableItem(2, false);
  }
  public btnClick(e: any): void {
    switch (e.target.id) {
      case 'Continue_Btn':
        let email: string | any = this.emailReference.nativeElement.value;
        let password: string | any = this.passwordRef.nativeElement.value;
        if (email !== '' && password !== '') {
          if (this.checkMail(email)) {

```

```

        email = password = '';
        (this.accordion as AccordionComponent).enableItem(1, true);
        (this.accordion as AccordionComponent).enableItem(0, false);
        (this.accordion as AccordionComponent).expandItem(true, 1);
    }
    this.error1.nativeElement.classList.remove('show');
} else {
    this.error1.nativeElement.classList.add('show');
}
break;
case 'Continue_BtnAdr':
    let name: string | any = this.nameRef.nativeElement.value;
    let address: string | any = this.addressRef.nativeElement.value;
    if((name !== '') && (address !== '') && (!isNOU((this.mobile as
NumericTextBoxComponent).value))) {
        if(this.checkMobile((this.mobile as
NumericTextBoxComponent).value)) {
            (this.accordion as AccordionComponent).enableItem(2, true);
            (this.accordion as AccordionComponent).enableItem(1, false);
            (this.accordion as AccordionComponent).expandItem(true, 2);
        }
        this.error2.nativeElement.classList.remove('show');
    } else {
        this.error2.nativeElement.classList.add('show');
    }
    break;
case 'Back_Btn':
    (this.accordion as AccordionComponent).enableItem(1, true);
    (this.accordion as AccordionComponent).enableItem(2, false);
    (this.accordion as AccordionComponent).expandItem(true, 1);
    break;
case 'Save_Btn':
    let cardHolder: string | any =
this.cardHolderRef.nativeElement.value;
    if(!isNOU((this.cardNo as NumericTextBoxComponent).value) &&
(cardHolder !== '') && (!isNOU((this.expiry as DatePickerComponent).value))
&& !isNOU((this.cvv as NumericTextBoxComponent).value)) {
        if (this.checkCardNo((this.cardNo as
NumericTextBoxComponent).value)) {
            if (this.checkCVV((this.cvv as NumericTextBoxComponent).value)) {
                (this.alertDlg as DialogComponent).content = this.success;
                (this.alertDlg as DialogComponent).show();
            }
        }
        this.error3.nativeElement.classList.remove('show');
    } else {
        this.error3.nativeElement.classList.add('show');
    }
    break;
}
}
public checkMail(mail: string) {
    if (/^\w+([\.-]?\w+)*@\w+([\.-]?\w+)*(\.\w{2,3})+$/ .test(mail)) {
        return true;
    } else {
        (this.alertDlg as DialogComponent).content = this.email_alert;
        (this.alertDlg as DialogComponent).show();
    }
}

```

```

        return false;
    }
}
public checkMobile(mobile: number) {
    if (/^\d{10}$/.test(mobile as any)) {
        return true;
    } else {
        (this.alertDlg as DialogComponent).content = this.mobile_alert;
        (this.alertDlg as DialogComponent).show();
        return false;
    }
}
public checkCardNo(cardNo: number) {
    if (/^\d{16}$/.test(cardNo as any)) {
        return true;
    } else {
        (this.alertDlg as DialogComponent).content = this.card_alert;
        (this.alertDlg as DialogComponent).show();
        return false;
    }
}
public checkCVV(cvv: number) {
    if (/^\d{3}$/.test(cvv as any)) {
        return true;
    } else {
        (this.alertDlg as DialogComponent).content = this.cvv_alert;
        (this.alertDlg as DialogComponent).show();
        return false;
    }
}
}
}

```

MAIN.TS

```

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

```

Load accordion with data source in Angular Accordion component

You can bind any data object to Accordion items, by mapping it to [header](#) and [content](#) property.

In the below demo, Data is fetched from an **OData** service using **DataManager**. The result data is formatted as a JSON object with [header](#) and [content](#) fields, which is set to [items](#) property of Accordion.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { AccordionModule } from '@syncfusion/ej2-angular-navigations'
import { Component, OnInit, ViewChild } from '@angular/core';
import { AccordionComponent } from '@syncfusion/ej2-angular-navigations';
import { DataManager, Query, ODataV4Adaptor, ReturnOption } from
 '@syncfusion/ej2-data';
const SERVICE_URI: string =
 'https://services.odata.org/V4/Northwind/Northwind.svc/Employees';

```

```

@Component({
  imports: [
    AccordionModule
  ],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-accordion #element></ejs-accordion>`
})
export class AppComponent implements OnInit {
  @ViewChild('element') accordionObj?: AccordionComponent;
  public itemsData: any = [];
  public mapping = { header: 'FirstName', content: 'Notes' };
  public ngOnInit(): void {
    new DataManager({ url: SERVICE_URI, adaptor: new ODataV4Adaptor })
      .executeQuery(new Query().range(1, 4)).then((e: ReturnOption) => {
        let result: any = e.result;
        for(let i: number = 0; i < result.length; i++) {
          this.itemsData.push({ header: result[i][this.mapping.header],
            content: result[i][this.mapping.content] });
        }
        (this.accordionObj as AccordionComponent).items = this.itemsData;
        (this.accordionObj as AccordionComponent).refresh();
      });
  }
}

```

MAIN.TS

```

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

```

Load accordion items dynamically in Angular Accordion component

Accordion items can be added dynamically by passing the item and index value with the [addItem](#) method.

In the following demo, you can add the accordion content by expanding any accordion header content using

[expanded](#) event.

- Data is fetched from the data source.
- The data is formatted as a JSON object with [header](#)

and [content](#) fields.

- Here last index is calculated to append the new accordion at the end.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { AccordionModule } from '@syncfusion/ej2-angular-navigations';

```

```
import { Component, ViewChild } from '@angular/core';
import { AccordionComponent } from '@syncfusion/ej2-angular-navigations';
import { Accordion, ExpandEventArgs, AccordionClickArgs, AccordionItemModel }
from '@syncfusion/ej2-navigations';
import { accordion } from './datasource';
let dbFlag: number = 0;
let dynamciAcrdnCount: number = 2;
@Component({
  imports: [
    AccordionModule
  ],
  standalone: true,
  selector: 'app-container',
  template: `
    <ejs-accordion #element (expanded)="expanded($event)">
    <e-accordionitems>
      <e-accordionitem expanded='true'>
        <ng-template #header>
          <div>ASP.NET</div>
        </ng-template>
        <ng-template #content>
          <div>Microsoft ASP.NET is a set of technologies in the
Microsoft .NET Framework for building Web applications and XML Web
services. ASP.NET pages execute on the server and generate
markup such as HTML, WML, or XML that is sent to a desktop
or mobile browser. ASP.NET pages use a compiled, event-driven
programming model that improves performance and enables
the separation of application logic and user interface.</div>
        </ng-template>
      </e-accordionitem>
      <e-accordionitem>
        <ng-template #header>
          <div>ASP.NET MVC</div>
        </ng-template>
        <ng-template #content>
          <div>The Model-View-Controller (MVC) architectural pattern
separates an application into three main components: the model,
the view, and the controller. The ASP.NET MVC framework
provides an alternative to the ASP.NET Web Forms pattern for
creating Web applications. The ASP.NET MVC framework is a
lightweight, highly testable presentation framework that
(as with Web Forms-based applications) is integrated with
existing ASP.NET features, such as master pages and membership-based
authentication.
          </div>
        </ng-template>
      </e-accordionitem>
    </e-accordionitems>
    </ejs-accordion>
  `
})
export class AppComponent {
  @ViewChild('element') acrdnInstance?: AccordionComponent;
  public expanded(e: ExpandEventArgs) {
    let Elementindex = document.getElementsByClassName("e-expand-state
e-selected e-active") [0];
```

```

        if([].slice.call((e.element as HTMLElement).parentElement as
HTMLElement).children).indexOf((e as any).element as never) ==
[].slice.call((e.element as HTMLElement).parentElement as
HTMLElement).children).indexOf(Elementindex as never)) {
            let array: AccordionItemModel[] = accordion as
AccordionItemModel[];
            for(let i: number = 0 ; i < dynamciAcrdnCount; i++)
            {
                if (dbFlag === array.length) {
                    return; }
                (this.acrdnInstance as AccordionComponent).addItem( array[dbFlag]
, (this.acrdnInstance as AccordionComponent).items.length );
                ++dbFlag;
            }
        }
        ngAfterViewInit() {
        }
    }
}

```

MAIN.TS

```

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

```

Customize expand collapse actions in Angular Accordion component

Accordion component supports customizing the expand or collapse animation action behavior. You can manually change the expand animation action performed after the collapse animation operation performed on already expand pane when the expand icons are clicked.

By default, the Accordion component pane is expanded or collapsed, when click the expand or collapse icon. It is not affected on already expand pane.

The following sample demonstrates, how to expand the collapsed Accordion item after collapse animation performed on the expanded Accordion item using [created](#), [expanding](#), and [expanded](#) event. In the Expanding event, get the previously expanded item index and prevent the expanding behavior using `args.cancel` option. Expand the Accordion item dynamically based on specifying the `index` value using the [expandItem](#) public method and [expanded](#) event.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { AccordionComponent, AccordionItemDirective, AccordionItemsDirective,
AccordionModule } from '@syncfusion/ej2-angular-navigations'
import { Component, ViewChild } from '@angular/core';
import { AccordionComponent, AccordionItemModel } from '@syncfusion/ej2-
angular-navigations';
import { Accordion, ExpandEventArgs } from '@syncfusion/ej2-navigations';
@Component({
  imports: [

    AccordionModule

```

```

    ],
    standalone: true,
    selector: 'app-container',
    template: `
      <ejs-accordion #element (expanding)="expanding($event)"
      (expanded)="expanded($event)" (created)="created($event)" expandMode='Single'
      >
        <e-accordionitems>
          <e-accordionitem>
            <ng-template #header>
              <div>ASP.NET</div>
            </ng-template>
            <ng-template #content>
              <div>Microsoft ASP.NET is a set of technologies in the
              Microsoft .NET Framework for building Web applications and XML Web
              services. ASP.NET pages execute on the server and generate
              markup such as HTML, WML, or XML that is sent to a desktop
              or mobile browser. ASP.NET pages use a compiled, event-driven
              programming model that improves performance and enables
              the separation of application logic and user interface.</div>
            </ng-template>
          </e-accordionitem>
          <e-accordionitem>
            <ng-template #header>
              <div>ASP.NET MVC</div>
            </ng-template>
            <ng-template #content>
              <div>The Model-View-Controller (MVC) architectural pattern
              separates an application into three main components: the model,
              the view, and the controller. The ASP.NET MVC framework
              provides an alternative to the ASP.NET Web Forms pattern for
              creating Web applications. The ASP.NET MVC framework is a
              lightweight, highly testable presentation framework that
              (as with Web Forms-based applications) is integrated with
              existing ASP.NET features, such as master pages and membership-based
              authentication.
            </div>
            </ng-template>
          </e-accordionitem>
          <e-accordionitem>
            <ng-template #header>
              <div>JavaScript</div>
            </ng-template>
            <ng-template #content>
              <div>JavaScript (JS) is an interpreted computer programming
              language. It was originally implemented as part of web browsers
              so that client-side scripts could interact with the user,
              control the browser, communicate asynchronously, and alter
              the document content that was displayed. More recently,
              however, it has become common in both game development and
              the creation of desktop applications.</div>
            </ng-template>
          </e-accordionitem>
        </e-accordionitems>
      </ejs-accordion>
    `
  })

```

```

export class AppComponent {
  @ViewChild('element') acrdnInstance?: AccordionComponent;
  public initialLoad = true;
  public isCollapsed = false;
  public expandIndex?: number;
  public expanding(e: ExpandEventArgs) {
    if (e.isExpanded && !this.initialLoad && !this.isCollapsed) {
      e.cancel = true;
      this.expandIndex = (this.acrdnInstance as
AccordionComponent).items.indexOf((e as ExpandEventArgs).item as
AccordionItemModel);
      this.isCollapsed = true;
    }
  }
  public expanded(e: ExpandEventArgs) {
    if (!e.isExpanded && !this.initialLoad && this.isCollapsed) {
      (this.acrdnInstance as AccordionComponent).expandItem(true,
this.expandIndex);
      this.isCollapsed = false;
    }
  }
  public created(args: any): void {
    this.initialLoad = false;
  }
  constructor() {
  }
}

```

MAIN.TS

```

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

```

Render accordion content using angular content in Accordion component

To render the Accordion contents using ng-content, we need to use ng-template inside the each e-accordionitem tag with #content attribute, which is mandatory to render content. Now include ng-content inside the ng-template tag with select attribute of id or class name for mapping required content.

`javascript

```
<e-accordionitem expanded='true' header='Athletics'>
```

```
<ng-template #content>
```

```
<ng-content select='div.content0'></ng-content>
```

```
</ng-template>
```

```
</e-accordionitem>
```

,

Here `div.content0` mapped to `ng-content` is reusable content. It can be used in multiple scenarios within the application.

APP.COMPONENT.HTML

```
<!-- Render the Accoridon Component by using ng-content -->
<ejs-accordion expandMode='Single'>
  <e-accordionitems>
    <e-accordionitem expanded='true' header='Athletics'>
      <ng-template #content>
        <ng-content select="div.content0"></ng-content>
      </ng-template>
    </e-accordionitem>
    <e-accordionitem header='Water Games'>
      <ng-template #content>
        <ng-content select="div.content1"></ng-content>
      </ng-template>
    </e-accordionitem>
    <e-accordionitem header='Racing'>
      <ng-template #content>
        <ng-content select="div.content2"></ng-content>
      </ng-template>
    </e-accordionitem>
    <e-accordionitem header='Indoor Games'>
      <ng-template #content>
        <ng-content select="div.content3"></ng-content>
      </ng-template>
    </e-accordionitem>
  </e-accordionitems>
</ejs-accordion>
```

APP.COMPONENT.TS

```
import { BrowserModule } from '@angular/platform-browser'
import { NgModule, ModuleWithProviders, CUSTOM_ELEMENTS_SCHEMA } from '@angular/core'
import { AccordionModule } from '@syncfusion/ej2-angular-navigations'
import { Component, ViewEncapsulation, Inject } from '@angular/core';
@Component({
  imports: [
    AccordionModule
  ],
  standalone: true,
  selector: 'my-thing',
  templateUrl: './app.component.html'
})
export class AccordionComponent {}
@Component({
  selector: 'control-content',
  templateUrl: './reusable-content.html',
  styleUrls: ['./app.component.css'],
  encapsulation: ViewEncapsulation.None
})
export class MyApp {}
```

MAIN.TS

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

Integrate treeview inside the accordion in Angular Accordion component

Accordion supports to render other Essential JS 2 Components by using content property.

You can give content as an element string like below, for initializing the component.

```
`js
```

```
content: '<div id="element"> </div>'
```

```
,
```

The other component can be rendered with the use of provided events, such as [clicked](#) and [expanding](#).

The following procedure is to render a TreeView within the Accordion,

- Import the **TreeView** module from **ej2-navigations**, for adding TreeView. Please refer the [TreeView initialization steps](#)
- You can initialize the TreeView component in [expanding](#) event,

by getting the element and defining the required TreeView properties.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { AccordionModule, TreeViewAllModule } from '@syncfusion/ej2-angular-navigations'
import { Component, ViewChild, ElementRef } from '@angular/core';
import { AccordionComponent, AccordionItemModel, TreeViewComponent } from '@syncfusion/ej2-angular-navigations';
import { Accordion, ExpandEventArgs, TreeView } from '@syncfusion/ej2-navigations';
import { DocDB, DownloadDB, PicDB } from './datasource';
@Component({
  imports: [
    AccordionModule, TreeViewAllModule
  ],
  standalone: true,
  selector: 'app-container',
  template: `
    <ejs-accordion #element>
      <e-accordionitems>
        <e-accordionitem expanded='true'>
          <ng-template #header>
            <div>Documents</div>
          </ng-template>
          <ng-template #content>
            <ejs-treeview id="treeDoc" [fields]='docField'
[sortOrder]='sortOrder'></ejs-treeview>
          </ng-template>
```

```

        </e-accordionitem>
        <e-accordionitem>
          <ng-template #header>
            <div>Downloads</div>
          </ng-template>
          <ng-template #content>
            <ejs-treeview id="treeDownload" [fields]='downField'
[sortOrder]='sortOrder'></ejs-treeview>
          </ng-template>
        </e-accordionitem>
        <e-accordionitem>
          <ng-template #header>
            <div>Pictures</div>
          </ng-template>
          <ng-template #content>
            <ejs-treeview id="treePic" [fields]='picField'
[sortOrder]='sortOrder'></ejs-treeview>
          </ng-template>
        </e-accordionitem>
      </e-accordionitems>
    </ejs-accordion>
  })
  export class AppComponent {
    @ViewChild('element') acrdnInstance?: AccordionComponent;
    @ViewChild('treeDocRef') treeDocRef!: ElementRef<HTMLDivElement>;
    public docField: Object = { dataSource: DocDB, id: 'nodeId', text:
'nodeText', child: 'nodeChild', iconCss: 'icon', imageUrl: 'image' };
    public downField: Object = { dataSource: DownloadDB, id: 'nodeId', text:
'nodeText', child: 'nodeChild', iconCss: 'icon', imageUrl: 'image' };
    public picField: Object = { dataSource: PicDB, id: 'nodeId', text:
'nodeText', child: 'nodeChild', iconCss: 'icon', imageUrl: 'image' };
    public sortOrder: 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));

```

Ej1 api migration in Angular Accordion component

This article describes the API migration process of Accordion 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-accordion id="Accordion"
[allowKeyboardNavigation]="false" ></ej-accordion> | Not Applicable |

| Localization | **Not Applicable** | **Property** : locale
 <ejs-accordion id="Accordion" locale="fr-BE" ></ejs-accordion> |

| Right to left | **Property**: enableRTL
 <ej-accordion id="Accordion" [enableRTL]="true"></ej-accordion> | **Property**: enableRTL
 <ejs-accordion id="Accordion" [enableRTL]="true"> </ejs-accordion> |

AjaxSettings

<!-- markdownlint-disable MD033 -->

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

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

| Default | **Property** : ajaxSettings
 <ej-accordion id="accordion" [ajaxSettings.type]="GET"></ej-accordion> | **Not Applicable** |

| Asynchronous | **Property** : ajaxSettings.async
 <ej-accordion id="accordion" [ajaxSettings.async]="true "></ej-accordion> | **Not Applicable** |

| Browser Cache | **Property**: ajaxSettings.cache
 <ej-accordion id="accordion" [ajaxSettings.cache]="false "></ej-accordion> | **Not Applicable** |

| Request type | **Property** : ajaxSettings.contentType
 <ej-accordion id="accordion" [ajaxSettings.contentType]="html "></ej-accordion> | **Not Applicable** |

| Data | **Property** : ajaxSettings.data
 <ej-accordion id="accordion" [ajaxSettings.data]={}></ej-accordion> | **Not Applicable** |

| Response type | **Property** : ajaxSettings.dataType
 <ej-accordion id="accordion" [ajaxSettings.dataType]="html"></ej-accordion> | **Not Applicable** |

| HTTP request type | **Property**: ajaxSettings.type
 <ej-accordion id="accordion" [ajaxSettings.type]="GET"></ej-accordion> | **Not Applicable** |

| AjaxBeforeLoad | **Event**: ajaxBeforeLoad
 <ej-accordion id='accordion' (ajaxBeforeLoad)='onajaxBeforeLoad(\$event)'></ej-accordion>
 TS:
 onajaxBeforeLoad (event){} | **Not Applicable** |

| AjaxError | **Event**: ajaxError
 <ej-accordion id='accordion' (ajaxError)='onajaxError(\$event)'></ej-accordion>
 TS:
 onajaxError (event){} | **Not Applicable** |

| AjaxLoad | **Event**: ajaxLoad
 <ej-accordion id='accordion' (ajaxLoad)='onajaxLoad(\$event)'></ej-accordion>
 TS:
 onajaxLoad (event){} | **Not Applicable** |

| AjaxSuccess | **Event**: ajaxSuccess
 <ej-accordion id='accordion' (ajaxSuccess)='onajaxSuccess(\$event)'></ej-accordion>
 TS:
 onajaxSuccess (event){} | **Not Applicable** |

Animation

<!-- markdownlint-disable MD033 -->

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

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

| Default | Not Applicable | **Property** :animation
 <ejs-accordion id='accordion' [animation]="animation"></ejs-accordion>
 TS:
 public animation: Object[] = [{collapse: { effect: 'FlipXDownIn', duration: 600, easing: 'ease' }, expand: { effect: 'FlipXUpIn', duration: 600, easing: 'ease' }}] |

| EnableAnimation | **Property** :animation
 <ej-accordion id="Accordion" [enableAnimation]="false"></ej-accordion>
 | Not Applicable |

| Expand animation | Not Applicable | **Property** :animation.expand
 <ejs-accordion id='accordion' [animation]="animation"></ejs-accordion>
 TS:
public animation: Object[] = [{expand: { effect: 'SlideLeft', duration: 600, easing: 'ease-in' }}] |

| Collapse animation | Not Applicable | **Property** :animation.collapse
 <ejs-accordion id='accordion' [animation]="animation"></ejs-accordion>
 TS:
public animation: Object[] = [{collapse: { effect: 'SlideLeft', duration: 600, easing: 'ease-in' }}] |

| Duration [expand / collapse] | Not Applicable | **Property** :animation.collapse.duration
 <ejs-accordion id='accordion' [animation]="animation"></ejs-accordion>
 TS:
public animation: Object[] = [{expand: { duration: 600 }}] |

| Easing [expand / collapse] | Not Applicable | **Property** :animation.collapse.easing
 <ejs-accordion id='accordion' [animation]="animation"></ejs-accordion>
 TS:
public animation: Object[] = [{expand: { easing: 'ease-in' }}] |

| Effect [expand / collapse] | Not Applicable | **Property** :animation.collapse.effect
 <ejs-accordion id='accordion' [animation]="animation"></ejs-accordion>
 TS:
public animation: Object[] = [{expand: { effect: 'SlideLeft' }}] |

Items

<!-- markdownlint-disable MD033 -->

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

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

| Default | Not Applicable | **Property** : items
 <ejs-accordion id="Accordion" [items]="items"> </ejs-accordion> |

| Content | Not Applicable | **Property** : items[0].content
 <ejs-accordion id="Accordion" [items]="items"> </ejs-accordion>
 TS:
 public items: Object[] = [{ content: 'Contents'}];|

| Custom class | Not Applicable | **Property** : items[0].cssClass
 <ejs-accordion id="Accordion" [items]="items"> </ejs-accordion>
 TS:
 public items: Object[] = [{ cssClass: 'customClass'}];|

| Header | Not Applicable | **Property** : items[0].Header
 <ejs-accordion id="Accordion" [items]="items"> </ejs-accordion>
 TS:
 public items: Object[] = [{ header: 'Header1'}];|

| HeaderSize | **Property** : items[0].headerSize
 <ej-accordion id="Accordion" headerSize="50px" ></ej-accordion> | Not Applicable |

```
| Icon class |<b>Not Applicable</b>| Property : items[0].iconCss <br/> <ejs-accordion
id="Accordion" [items]="items"> </ejs-accordion><br/> TS:<br/> public items: Object[] = [{ iconCss:
'e-icons' }];|

| IsExpand |<b>Not Applicable</b>| Property : items[0].expanded <br/> <ejs-accordion
id="Accordion" [items]="items"> </ejs-accordion><br/> TS:<br/> public items: Object[] = [{
expanded: true }];|

| Collapse Item |Method : collapsePanel(index) <br/> <ej-accordion id="Accordion"
#Accordion></ej-accordion><br/> TS:<br/> @ViewChild('Accordion') public AccordionObj:
AccordionComponent;<br/>AccordionObj.collapsePanel(0);| Method : expandItem(index, false) <br/>
<ejs-accordion id="Accordion" [items]="items"> </ejs-accordion><br/> TS:<br/>
@ViewChild('Accordion') public AccordionObj: AccordionComponent;<br/> AccordionObj.expandItem(0,
false);|

| Expand Item |Method : expandPanel(index) <br/> <ej-accordion id="Accordion" #Accordion></ej-
accordion><br/> TS:<br/> @ViewChild('Accordion') public AccordionObj:
AccordionComponent;<br/>AccordionObj.expandPanel(0);| Method : expandItem(index, true) <br/>
<ejs-accordion id="Accordion" [items]="items"> </ejs-accordion><br/> TS:<br/>
@ViewChild('Accordion') public AccordionObj: AccordionComponent;<br/>AccordionObj.expandItem(0,
true);|

| CollapseAll |Method : collapseAll() <br/> <ej-accordion id="Accordion" #Accordion></ej-
accordion><br/> TS:<br/> @ViewChild('Accordion') public AccordionObj:
AccordionComponent;<br/>AccordionObj.collapseAll();| <b>Not Applicable</b>|

| ExpandAll |Method : expandAll() <br/> <ej-accordion id="Accordion" #Accordion></ej-
accordion><br/> TS:<br/> @ViewChild('Accordion') public AccordionObj:
AccordionComponent;<br/>AccordionObj.expandAll();| <b>Not Applicable</b>|

| Get ItemsCount |Method : getItemsCount() <br/> <ej-accordion id="Accordion" #Accordion></ej-
accordion><br/> TS:<br/> @ViewChild('Accordion') public AccordionObj:
AccordionComponent;<br/>AccordionObj.getItemsCount();| <b>Not Applicable</b>|

| AddItem |Method : addItem(text, content, index) <br/> <ej-accordion id="Accordion"
#Accordion></ej-accordion><br/> TS:<br/> @ViewChild('Accordion') public AccordionObj:
AccordionComponent;<br/> AccordionObj.addItem("New item", "The accordion content", 2);| Method :
addItem(items, index) <br/> <ejs-accordion id="Accordion" [items]="items"> </ejs-
accordion><br/> TS:<br/> @ViewChild('Accordion') public AccordionObj:
AccordionComponent;<br/>AccordionObj.addItem({ header: 'App', content: 'text' }, 0);|

| Remove Item |Method : removeItem(index) <br/> <ej-accordion id="Accordion" #Accordion></ej-
accordion><br/> TS:<br/> @ViewChild('Accordion') public AccordionObj: AccordionComponent;<br/>
AccordionObj.removeItem(0);| Method : removeItem(index) <br/> <ejs-accordion id="Accordion"
[items]="items"> </ejs-accordion><br/> TS:<br/> @ViewChild('Accordion') public AccordionObj:
AccordionComponent;<br/>AccordionObj.removeItem(index);|

| Disable Items |Property : disabledItems <br/> <ej-accordion id="Accordion" disabledItems="[0,
1]"></ej-accordion>| <b>Not Applicable</b>|
```

| Enable Items | **Property** : enabledItems
 <ej-accordion id="Accordion" enabledItems="[0, 1]"></ej-accordion> | Not Applicable |

| Disable Item | **Method** : disableItems([index])
 <ej-accordion id="Accordion" #Accordion></ej-accordion>
 TS:
 @ViewChild('Accordion') public AccordionObj: AccordionComponent;
 AccordionObj.disableItems([1]); | **Method** : enableItem(index, false)
 <ejs-accordion id="Accordion" #Accordion></ejs-accordion>
 TS:
 @ViewChild('Accordion') public AccordionObj: AccordionComponent;
 AccordionObj.enableItem(0, false); |

| Enable Item | **Method** : enableItems([index])
 <ej-accordion id="Accordion" #Accordion></ej-accordion>
 TS:
 @ViewChild('Accordion') public AccordionObj: AccordionComponent;
 AccordionObj.enableItems([1]); | **Method** : enableItem(index, true)
 <ejs-accordion id="Accordion" #Accordion></ejs-accordion>
 TS:
 @ViewChild('Accordion') public AccordionObj: AccordionComponent;
 AccordionObj.enableItem(0, true); |

| Hide Item | Not Applicable | **Method** : hideItem(index, true)
 <ejs-accordion id="Accordion" #Accordion></ejs-accordion>
 TS:
 @ViewChild('Accordion') public AccordionObj: AccordionComponent;
 AccordionObj.hideItem(0, true) |

| SelectedItemIndex | **Property** : selectedItemIndex
 <ej-accordion id="Accordion" #Accordion [selectedItemIndex]="false"></ej-accordion>
 | Not Applicable |

| Select | Not Applicable | **Method** : select(index)
 <ejs-accordion id="Accordion" #Accordion></ejs-accordion>
 TS:
 @ViewChild('Accordion') public AccordionObj: AccordionComponent;
 AccordionObj.select(0); |

| BeforeActivate | **Event**: beforeActivate
 <ej-accordion id='accordion' (beforeActivate)='onbeforeActivate(\$event)'></ej-accordion>
 TS:
 onbeforeActivate(event){} | **Event**: expanding
 <ejs-accordion id="Accordion" #Accordion (expanding)='onexpanding(\$event)'></ejs-accordion>
 TS:
 onexpanding(event){} |

| Activate | **Event**: activate
 <ej-accordion id='accordion' (activate)='onActivate(\$event)'></ej-accordion>
 TS:
 onActivate(event){} | **Event**: expanded
 <ejs-accordion id="Accordion" #Accordion (expanded)='onexpanded(\$event)'></ejs-accordion>
 TS:
 onexpanded(event){} |

| beforeInActivate | **Event**: beforeInactivate
 <ej-accordion id='accordion' (beforeInactivate)='onbeforeInactivate(\$event)'></ej-accordion>
 TS:
 onbeforeInactivate(event){} | Not Applicable |

| InActive | **Event**: inActivate
 <ej-accordion id='accordion' (inActivate)='onInActivate(\$event)'></ej-accordion>
 TS:
 onInActivate(event){} | Not Applicable |

| Clicked | **Event**: clicked
 <ejs-accordion id="Accordion" #Accordion (clicked)='onclicked(\$event)'></ejs-accordion>
 TS:
 onclicked (event){} | Not Applicable |

Common

<!-- markdownlint-disable MD033 -->

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

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

| Collapsible | **Property** : collapsible
 <ej-accordion id="Accordion" [collapsible]="false"> </ej-accordion>| Not Applicable |

| Collapse speed | **Property** : collapseSpeed
 <ej-accordion id="Accordion" collapseSpeed="500"> </ej-accordion>| Not Applicable |

| Custom class | **Property** : cssClass
 <ej-accordion id="Accordion" cssClass="custom" > </ej-accordion>| Not Applicable |

| CustomIcon class | **Property**: customIcon
 <ej-accordion id="Accordion" customIcon="custom" > </ej-accordion>
 TS:
 public custom: Object = {header: "header-close", selectedHeader: "header-expand"}; | Not Applicable |

| Enabled | **Property** : enabled
 <ej-accordion id="Accordion" enabled="false" > </ej-accordion>| Not Applicable |

| Events | **Property** : events
 <ej-accordion id="Accordion" events="mouseover" > </ej-accordion>| Not Applicable |

| Expand speed | **Property** : expandSpeed
 <ej-accordion id="Accordion" expandSpeed="300"> </ej-accordion>| Not Applicable |

| Height | **Property** : height
 <ej-accordion id="Accordion" height="400" > </ej-accordion>| **Property** : height
 <ejs-accordion id="Accordion" height="400" > </ejs-accordion> |

| HeightAdjustMode | **Property** : heightAdjustMode
 <ej-accordion id="Accordion" heightAdjustMode="content" > </ej-accordion>| Not Applicable |

| HtmlAttributes | **Property** : htmlAttributes
 <ej-accordion id="Accordion" [htmlAttributes]="attributes" > </ej-accordion>
 TS:
 public attributes: Object = {title: "Demo"};| Not Applicable |

| MultipleOpen | **Property** : enableMultipleOpen
 <ej-accordion id="Accordion" [enableMultipleOpen]="true" > </ej-accordion>| **Property** : expandMode
 <ejs-accordion id="Accordion" [expandMode]="Multiple" > </ejs-accordion> |

| Persistence | **Property** : enablePersistence
 <ej-accordion id="Accordion" [enablePersistence]="false" > </ej-accordion>| **Property** : enablePersistence
 <ejs-accordion id="Accordion" [enablePersistence]="true" > </ejs-accordion> |

| ShowRoundedCorner | **Property** : showRoundedCorner
 <ej-accordion id="Accordion" [showRoundedCorner]="false" > </ej-accordion>| Not Applicable |

| Width | **Property** : width
 <ej-accordion id="Accordion" [width]="600" > </ej-accordion>| **Property** : width
 <ejs-accordion id="Accordion" [width]="400" > </ejs-accordion> |

| Enable | **Method** : enable()
 <ej-accordion id="Accordion" #Accordion></ej-accordion>
 TS:
 @ViewChild('Accordion') public AccordionObj: AccordionComponent;
 AccordionObj.enable(); | Not Applicable |

| Disable | **Method** : disable()
 <ej-accordion id="Accordion" #Accordion></ej-accordion>
 TS:
 @ViewChild('Accordion') public AccordionObj: AccordionComponent;
 AccordionObj.disable(); | Not Applicable |

| Show | **Method** : show()
 <ej-accordion id="Accordion" #Accordion></ej-accordion>
 TS:
 @ViewChild('Accordion') public AccordionObj: AccordionComponent;
 AccordionObj.show(); | Not Applicable |

| Hide | **Method** : hide()
 <ej-accordion id="Accordion" #Accordion></ej-accordion>
 TS:
 @ViewChild('Accordion') public AccordionObj: AccordionComponent;
 AccordionObj.hide(); | Not Applicable |

| Destroy | **Method** : destroy()
 <ej-accordion id="Accordion" #Accordion></ej-accordion>
 TS:
 @ViewChild('Accordion') public AccordionObj: AccordionComponent;
 AccordionObj.destroy(); | **Method** : destroy()
 <ejs-accordion id="Accordion" #Accordion></ejs-accordion>
 TS:
 @ViewChild('Accordion') public AccordionObj: AccordionComponent;
 AccordionObj.destroy(); |

| Refresh | **Method** : refresh()
 <ej-accordion id="Accordion" #Accordion></ej-accordion>
 TS:
 @ViewChild('Accordion') public AccordionObj: AccordionComponent;
 AccordionObj.refresh(); | **Method** : refresh()
 <ejs-accordion id="Accordion" #Accordion></ejs-accordion>
 TS:
 @ViewChild('Accordion') public AccordionObj: AccordionComponent;
 AccordionObj.refresh(); |

| Created | **Event**: create
 <ej-accordion id="Accordion" #Accordion (create)='oncreate(\$event)'></ej-accordion>
 TS:
 oncreate(event) {} | **Event**: created
 <ejs-accordion id="Accordion" #Accordion (created)='oncreated(\$event)'></ejs-accordion>
 TS:
 oncreated(event) {} |

| Destroyed | **Event**: destroy
 <ej-accordion id="Accordion" #Accordion (destroy)='ondestroy(\$event)'></ej-accordion>
 TS:
 ondestroy(event) {} | **Event**: destroyed
 <ejs-accordion id="Accordion" #Accordion (destroyed)='ondestroyed(\$event)'></ejs-accordion>
 TS:
 ondestroyed(event) {} |

Accumulation Chart

<!-- markdownlint-disable MD036 -->

Getting started with Angular Accumulation chart component

This section explains you the steps required to create a simple Chart and demonstrate the basic usage of the AccumulationChart component in an Angular environment.

Setup Angular Environment

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

To install Angular CLI use the following command.

```
`bash`
```

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

```
,
```

Create an Angular Application

Start a new Angular application using below Angular CLI command.

```
`bash
```

```
ng new my-app
```

```
cd my-app
```

```
,
```

Installing Syncfusion AccumulationChart 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-charts](#) package to the application.

```
`bash
```

```
npm install @syncfusion/ej2-angular-charts --save
```

```
,
```

Angular compatibility compiled package(ngcc)

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

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

```
`bash
```

```
npm install @syncfusion/ej2-angular-charts@ngcc --save
```

```
,
```

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

```
`bash
```

```
@syncfusion/ej2-angular-charts:"20.2.38-ngcc"
```

```
,
```

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

Registering AccumulationChart Module

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

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

- Modify the template in `app.component.ts` file to render the `ej2-angular-charts` component

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

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

Now use the `<code>app-container</code>` in the index.html instead of default one.

```
<app-container></app-container>
```

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

```
npm start
```

Pie Series

By default pie series will be rendered on assigning JSON data to the series by using [dataSource](#) property. Map the field names in the JSON data to the [xName](#) and [yName](#) properties of the series.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { AccumulationChartModule } from '@syncfusion/ej2-angular-charts'
import { PieSeriesService, AccumulationLegendService,
  AccumulationTooltipService, AccumulationAnnotationService,
  AccumulationDataLabelService } from '@syncfusion/ej2-angular-charts'
import { Component, OnInit } from '@angular/core';
@Component({
  imports: [
    AccumulationChartModule
  ],
  providers: [PieSeriesService, AccumulationLegendService,
    AccumulationTooltipService, AccumulationDataLabelService,
    AccumulationAnnotationService],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-accumulationchart id="chart-container"
[legendSettings]='legendSettings'>
  <e-accumulation-series-collection>
    <e-accumulation-series [dataSource]='piedata' xName='x'
yName='y'></e-accumulation-series>
  </e-accumulation-series-collection>
</ejs-accumulationchart>`
})
export class AppComponent implements OnInit {
  public piedata?: Object[];
  public legendSettings?: Object;
  ngOnInit(): void {
    this.piedata = [
      { x: 'Jan', y: 3, text: 'Jan: 3' }, { x: 'Feb', y: 3.5, text:
'Feb: 3.5' },
      { x: 'Mar', y: 7, text: 'Mar: 7' }, { x: 'Apr', y: 13.5,
text: 'Apr: 13.5' },
      { x: 'May', y: 19, text: 'May: 19' }, { x: 'Jun', y: 23.5,
text: 'Jun: 23.5' },
      { x: 'Jul', y: 26, text: 'Jul: 26' }, { x: 'Aug', y: 25,
text: 'Aug: 25' },
```

```

        { x: 'Sep', y: 21, text: 'Sep: 21' }, { x: 'Oct', y: 15,
text: 'Oct: 15' },
        { x: 'Nov', y: 9, text: 'Nov: 9' }, { x: 'Dec', y: 3.5, text:
'Dec: 3.5' }]];
        this.legendSettings = {
            visible: false
        };
    }
}

```

MAIN.TS

```

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

```

Pie dough nut in Angular Accumulation chart component

Pie Chart

To render a pie series, use the series [type](#) as Pie and inject the [PieSeriesService](#) module into the [@NgModule.providers](#). If the [Link to the Video](#) module is not injected, this module will be loaded by default.

To know about circular and triangular charts, you can check on this video:

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { AccumulationChartModule } from '@syncfusion/ej2-angular-charts'
import { PieSeriesService, AccumulationLegendService,
AccumulationTooltipService, AccumulationAnnotationService,
    AccumulationDataLabelService } from '@syncfusion/ej2-angular-charts'
import { Component, OnInit } from '@angular/core';
import { pieData } from './datasource';
@Component({
    imports: [
        AccumulationChartModule
    ],
    providers: [PieSeriesService, AccumulationLegendService,
        AccumulationTooltipService, AccumulationDataLabelService,
        AccumulationAnnotationService],
    standalone: true,
    selector: 'app-container',
    template: `<ejs-accumulationchart id="chart-container"
[legendSettings]='legendSettings'>
        <e-accumulation-series-collection>
            <e-accumulation-series [dataSource]='piedata' xName='x' yName='y'
type='Pie'></e-accumulation-series>
        </e-accumulation-series-collection>
    </ejs-accumulationchart>`
})
export class AppComponent implements OnInit {
    public piedata?: Object[];
    public legendSettings?: Object;

```

```

ngOnInit(): void {
    this.piedata = pieData;
    this.legendSettings = {
        visible: false
    };
}
}

```

MAIN.TS

```

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

```

Radius Customization

By default, radius of the pie series will be 80% of the size (minimum of chart width and height).

You can customize this using [radius](#) property of the series.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { AccumulationChartModule } from '@syncfusion/ej2-angular-charts'
import { PieSeriesService, AccumulationLegendService,
    AccumulationTooltipService, AccumulationAnnotationService,
    AccumulationDataLabelService } from '@syncfusion/ej2-angular-charts'
import { Component, OnInit } from '@angular/core';
import { pieData } from './datasource';
@Component({
    imports: [
        AccumulationChartModule
    ],
    providers: [PieSeriesService, AccumulationLegendService,
        AccumulationTooltipService, AccumulationDataLabelService,
        AccumulationAnnotationService],
    standalone: true,
    selector: 'app-container',
    template: `<ejs-accumulationchart id="chart-container"
[legendSettings]='legendSettings'>
    <e-accumulation-series-collection>
        <e-accumulation-series [dataSource]='piedata' xName='x' yName='y'
radius="100%"></e-accumulation-series>
    </e-accumulation-series-collection>
</ejs-accumulationchart>`
})
export class AppComponent implements OnInit {
    public piedata?: Object[];
    public legendSettings?: Object;
    ngOnInit(): void {
        this.piedata = pieData;
        this.legendSettings = {
            visible: false
        };
    }
}

```

```
}

```

MAIN.TS

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

Pie Center

The center position of the pie can be changed by Center X and Center Y. By default, center value of the pie series x and y is 50%. You can customize this using [center](#) property of the series.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { AccumulationChartModule } from '@syncfusion/ej2-angular-charts'
import { PieSeriesService, AccumulationLegendService,
AccumulationTooltipService, AccumulationAnnotationService,
  AccumulationDataLabelService } from '@syncfusion/ej2-angular-charts'
import { Component, OnInit } from '@angular/core';
import { data } from './datasource';
@Component({
  imports: [
    AccumulationChartModule
  ],
  providers: [PieSeriesService, AccumulationLegendService,
AccumulationTooltipService, AccumulationDataLabelService,
  AccumulationAnnotationService],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-accumulationchart id="container" width='92%'
[legendSettings]="legendSettings" [title]="title" [enableAnimation]=
'enableAnimation' [center]='center'>
    <e-accumulation-series-collection>
      <e-accumulation-series name='Browser' [dataSource]='pieData'
xName='x' yName='y' [startAngle]="startAngle" [endAngle]="endAngle"
innerRadius="0%" radius="70%" [explode]='explode' explodeOffset='10%'
[explodeIndex]='0'>
        </e-accumulation-series>
      </e-accumulation-series-collection>
    </ejs-accumulationchart>`
})
export class AppComponent implements OnInit {
  public pieData?: Object[];
  public startAngle?: number;
  public endAngle?: number;
  public center?: Object ;
  public explode?: boolean ;
  public enableAnimation?: boolean ;
  public title?: string ;
  public legendSettings?: Object;
  ngOnInit(): void {
    this.pieData = data;
```

```

        this.legendSettings = {
            visible: false
        };
        this.center = {x: '60%', y: '60%'};
        this.startAngle = 0;
        this.endAngle = 360;
        this.explode = true;
        this.enableAnimation = false;
        this.title = 'Mobile Browser Statistics';
    }
}

```

MAIN.TS

```

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

```

Various Radius Pie Chart

You can use radius mapping to render the slice with different [radius](#) pie and also use [border](#), fill properties to customize the point. dataLabel is used to represent individual data and its value.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { AccumulationChartModule } from '@syncfusion/ej2-angular-charts'
import { PieSeriesService, AccumulationLegendService,
    AccumulationTooltipService, AccumulationAnnotationService,
    AccumulationDataLabelService } from '@syncfusion/ej2-angular-charts'
import { Component, OnInit } from '@angular/core';
import { variespiedata } from './datasource';
@Component({
    imports: [
        AccumulationChartModule
    ],
    providers: [PieSeriesService, AccumulationLegendService,
        AccumulationTooltipService, AccumulationDataLabelService,
        AccumulationAnnotationService],
    standalone: true,
    selector: 'app-container',
    template: `<ejs-accumulationchart id="container" width='92%'
[legendSettings]="legendSettings" [title]="title" [enableAnimation]=
'enableAnimation'>
        <e-accumulation-series-collection>
            <e-accumulation-series name='Browser' [dataSource]='pieData'
xName='x' yName='y' [startAngle]="startAngle" [endAngle]="endAngle"
innerRadius="20%" radius="r">
                </e-accumulation-series>
            </e-accumulation-series-collection>
        </ejs-accumulationchart>`
})
export class AppComponent implements OnInit {
    public pieData?: Object[];
}

```



```

    public startAngle?: number;
    public endAngle?: number;
    public center?: Object ;
    public explode?: boolean ;
    public enableAnimation?: boolean ;
    public title?: string ;
    public radius?: string ;
    public legendSettings?: Object;
    ngOnInit(): void {
        this.pieData = variespiedata;
        this.legendSettings = {
            visible: false
        };
        this.startAngle = 0;
        this.endAngle = 360;
        this.radius = 'r';
        this.enableAnimation = true;
        this.title = 'Mobile Browser Statistics';
    }
}

```

MAIN.TS

```

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

```

Doughnut Chart

To achieve a doughnut in pie series, customize the [innerRadius](#) property of the series. By setting value greater than 0%, a doughnut will appear. The [innerRadius](#) property takes value from 0% to 100% of the pie radius.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { AccumulationChartModule } from '@syncfusion/ej2-angular-charts'
import { PieSeriesService, AccumulationLegendService,
    AccumulationTooltipService, AccumulationAnnotationService,
    AccumulationDataLabelService } from '@syncfusion/ej2-angular-charts'
import { Component, OnInit } from '@angular/core';
import { pieData } from './datasource';
@Component({
    imports: [
        AccumulationChartModule
    ],
    providers: [PieSeriesService, AccumulationLegendService,
        AccumulationTooltipService, AccumulationDataLabelService,
        AccumulationAnnotationService],
    standalone: true,
    selector: 'app-container',
    template: `<ejs-accumulationchart id="chart-container"
    [legendSettings]='legendSettings'>
        <e-accumulation-series-collection>

```

```

        <e-accumulation-series [dataSource]='piedata' xName='x' yName='y'
innerRadius='40%'></e-accumulation-series>
    </e-accumulation-series-collection>
</ejs-accumulationchart>`
    })
    export class AppComponent implements OnInit {
        public piedata?: Object[];
        public legendSettings?: Object;
        ngOnInit(): void {
            this.piedata = pieData;
            this.legendSettings = {
                visible: false
            };
        }
    }
}

```

MAIN.TS

```

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

```

Start and End angles

You can customize the start and end angle of the pie series using the [startAngle](#) and [endAngle](#) properties. The default value of [startAngle](#) is 0 degree, and [endAngle](#) is 360 degrees. By customizing this, you can achieve a semi pie series.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { AccumulationChartModule } from '@syncfusion/ej2-angular-charts'
import { PieSeriesService, AccumulationLegendService,
AccumulationTooltipService, AccumulationAnnotationService,
    AccumulationDataLabelService } from '@syncfusion/ej2-angular-charts'
import { Component, OnInit } from '@angular/core';
import { pieData } from './datasource';
@Component({
    imports: [
        AccumulationChartModule
    ],
    providers: [PieSeriesService, AccumulationLegendService,
        AccumulationTooltipService, AccumulationDataLabelService,
        AccumulationAnnotationService],
    standalone: true,
    selector: 'app-container',
    template: `<ejs-accumulationchart id="chart-container"
[legendSettings]='legendSettings'>
    <e-accumulation-series-collection>
        <e-accumulation-series [dataSource]='piedata' xName='x' yName='y'
[startAngle]='startAngle' [endAngle]='endAngle' [dataLabel]='datalabel'></e-
accumulation-series>
    </e-accumulation-series-collection>
</ejs-accumulationchart>`
})

```

```

    })
    export class AppComponent implements OnInit {
        public piedata?: Object[];
        public startAngle?: number;
        public endAngle?: number;
        public datalabel?: Object;
        public legendSettings?: Object;
        ngOnInit(): void {
            this.startAngle = 270;
            this.endAngle = 90;
            this.datalabel = { visible: true, name: 'text', position: 'Outside' };

            this.piedata = pieData;
            this.legendSettings = {
                visible: false
            };
        }
    }
}

```

MAIN.TS

```

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

```

Color & Text Mapping

The fill color and the text in the data source can be mapped to the chart using `pointColorMapping` in series and `name` in datalabel respectively.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { AccumulationChartModule } from '@syncfusion/ej2-angular-charts';
import { PieSeriesService, AccumulationLegendService,
    AccumulationTooltipService, AccumulationAnnotationService,
    AccumulationDataLabelService } from '@syncfusion/ej2-angular-charts';
import { Component, OnInit } from '@angular/core';
import { dataMapping } from './datasource';
@Component({
    imports: [
        AccumulationChartModule
    ],
    providers: [PieSeriesService, AccumulationLegendService,
        AccumulationTooltipService, AccumulationDataLabelService,
        AccumulationAnnotationService],
    standalone: true,
    selector: 'app-container',
    template: `<ejs-accumulationchart id="chart-container"
[legendSettings]='legendSettings'>
    <e-accumulation-series-collection>
        <e-accumulation-series [dataSource]='piedata' xName='x' yName='y'
[pointColorMapping]= 'map' [dataLabel]='datalabel'></e-accumulation-series>
    </e-accumulation-series-collection>

```

```

    </ejs-accumulationchart>`
  })
  export class AppComponent implements OnInit {
    public piedata?: Object[];
    public map: Object = 'fill';
    public datalabel?: Object;
    public legendSettings?: Object;
    ngOnInit(): void {
      this.piedata = dataMapping;
      this.datalabel = { visible: true, name: 'text', position: 'Outside' };
    };

    this.legendSettings = {
      visible: false
    };
  }
}

```

MAIN.TS

```

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

```

Hide pie or doughnut border

By default, the border will appear in the pie/doughnut chart while mouse hover on the chart. You can disable the the border by setting `enableBorderOnMouseMove` property is `false`.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { AccumulationChartModule } from '@syncfusion/ej2-angular-charts'
import { PieSeriesService, AccumulationLegendService,
  AccumulationTooltipService, AccumulationAnnotationService,
  AccumulationDataLabelService } from '@syncfusion/ej2-angular-charts'
import { Component, OnInit } from '@angular/core';
import { dataMapping } from './datasource';
@Component({
  imports: [
    AccumulationChartModule
  ],
  providers: [PieSeriesService, AccumulationLegendService,
    AccumulationTooltipService, AccumulationDataLabelService,
    AccumulationAnnotationService],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-accumulationchart id="chart-container"
[legendSettings]='legendSettings'
[enableBorderOnMouseMove]='enableBorderOnMouseMove'>
  <e-accumulation-series-collection>
    <e-accumulation-series [dataSource]='piedata' xName='x' yName='y'
[dataLabel]='datalabel'></e-accumulation-series>
  </e-accumulation-series-collection>
</ejs-accumulationchart>`

```

```

    })
    export class AppComponent implements OnInit {
        public piedata?: Object[];
        public datalabel?: Object;
        public legendSettings?: Object;
        public enableBorderOnMouseMove?: boolean;
        ngOnInit(): void {
            this.enableBorderOnMouseMove = false;
            this.piedata = dataMapping;
            this.datalabel = { visible: true, name: 'text', position: 'Outside' };

            this.legendSettings = {
                visible: false
            };
        }
    }
}

```

MAIN.TS

```

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

```

Multi-level pie chart

You can achieve a multi-level drill down in pie and doughnut charts using [pointClick](#) event. If user clicks any point in the chart, that corresponding data will be shown in the next level and so on based on point clicked.

You can also achieve drill-up (back to the initial state) by using [chartMouseClicked](#) event. In below sample, you can drill-up by clicking back button in the center of the chart.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ChartModule } from '@syncfusion/ej2-angular-charts'
import { GridModule } from '@syncfusion/ej2-angular-grids'
import { PageService, SortService, FilterService, GroupService } from '@syncfusion/ej2-angular-grids'
import { AccumulationChartModule } from '@syncfusion/ej2-angular-charts'
import { PieSeriesService, AccumulationTooltipService, AccumulationDataLabelService } from '@syncfusion/ej2-angular-charts'
import { LineSeriesService, DateTimeService, DataLabelService, StackingColumnSeriesService, CategoryService, ChartShape, StepAreaSeriesService, SplineSeriesService, ChartAnnotationService, LegendService, TooltipService, StripLineService, SelectionService, ScatterSeriesService } from '@syncfusion/ej2-angular-charts'
import { Component, ViewChild } from '@angular/core';
import {
    AccumulationChartComponent,
    IMouseEventArgs,
    IAccTextRenderEventArgs,
    AccumulationChart,
}

```

```

} from '@syncfusion/ej2-angular-charts';
import { GridComponent } from '@syncfusion/ej2-angular-grids';
import { getElement } from '@syncfusion/ej2-charts';
/**
 * Sample for Drilldown in Pie chart
 */
@Component({
  imports: [
    ChartModule, AccumulationChartModule, GridModule
  ],
  providers: [ LineSeriesService, DateTimeService, DataLabelService,
    StackingColumnSeriesService, CategoryService,
    StepAreaSeriesService, SplineSeriesService,
    ChartAnnotationService, LegendService, TooltipService, StripLineService,
    PieSeriesService, AccumulationTooltipService,
    AccumulationDataLabelService, SelectionService, ScatterSeriesService
    , PageService ],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-accumulationchart id="container" #pie style='display:block;
width: 92%' [legendSettings]="legendSettings"
[enableSmartLabels]='false' [title]="title"
(textRender)="onTextRender($event)"
(chartMouseClicked)="onChartMouseClicked($event)"
(pointClick)="onPointClick($event)">
    <e-accumulation-series-collection>
      <e-accumulation-series [dataSource]='data' xName='x'
yName='y' [startAngle]="startAngle"
[endAngle]="endAngle" innerRadius="0%" radius="70%"
[dataLabel]="dataLabel" [explode]="explode"
explodeOffset='10%' [explodeIndex]='explodeIndex'>
        </e-accumulation-series>
      </e-accumulation-series-collection>
    </ejs-accumulationchart>
    <ejs-grid id='Grid' #grid [dataSource]='data'>
      <e-columns>
        <e-column field='x' headerText='Vehicle' type='string'></e-
column>
        <e-column field='y' headerText='Sales' type='string'></e-
column>
      </e-columns>
    </ejs-grid>`,
  ))
export class AppComponent {
  public data = [
    {
      x: 'SUV',
      y: 25,
      z: [
        {
          title: 'Automobile Sales in the SUV Segment',
          x: 'Toyota',
          y: 8,
          z: [
            { x: '2000', y: 20 },
            { x: '2001', y: 30 },
            { x: '2002', y: 40 },
          ]
        }
      ]
    }
  ]
}

```

```

        ],
        {
            x: 'Ford', y: 12 },
        { x: 'GM', y: 17 },
        { x: 'Renault', y: 6 },
    ],
},
{
    x: 'Car',
    y: 37,
    z: [
        { title: 'Automobile Sales in the Car Segment', x: 'Toyota', y: 7 },
        { x: 'Chrysler', y: 12 },
        { x: 'Nissan', y: 9 },
        { x: 'Ford', y: 15 },
    ],
},
{
    x: 'Pickup',
    y: 15,
    z: [
        { title: 'Automobile Sales in the Pickup Segment', x: 'Nissan', y: 9
        { x: 'Chrysler', y: 4 },
        { x: 'Ford', y: 7 },
        { x: 'Toyota', y: 20 },
    ],
},
{
    x: 'Minivan',
    y: 23,
    z: [
        {
            title: 'Automobile Sales in the Minivan Segment',
            x: 'Hummer',
            y: 11,
        },
        { x: 'Ford', y: 5 },
        { x: 'GM', y: 12 },
        { x: 'Chrysler', y: 3 },
    ],
},
];
@ViewChild('pie')
public pie?: AccumulationChartComponent | AccumulationChart;
@ViewChild('grid')
public grid?: GridComponent;
public pointIndex: number = -1;
//Initializing Legend
public legendSettings: Object = {
    visible: false,
};
//Initializing Datalabel
public dataLabel: Object = {
    visible: true,
    position: 'Inside',
    connectorStyle: { type: 'Curve', length: '5%' },

```

```

        font: { size: '14px', color: 'white' },
    };
    public explode: boolean = false;
    public content: string =
        '<div id="back" style="cursor:pointer;padding:3px;width:30px;
height:30px;">' +
        '';
    public startAngle: number = 0;
    public explodeIndex: number = 2;
    public endAngle: number = 360;
    public title: string = 'Automobile Sales by Category';
    public isparent: boolean = true;
    public onTextRender(args: IAccTextRenderEventArgs): void {
        args.text = args.point.x + ' ' + args.point.y + ' %';
    }
    public onChartMouseClicked(args: IMouseEventArgs): void {
        if (args.target.indexOf('back') > -1) {
            if (this.pie?.series[0].name === 'Level 3') {
                this.pie.series[0].dataSource = this.data[this.pointIndex].z;
                this.pie.series[0].name = 'Level 2';
                this.pie.title = this.data[this.pointIndex].z[0].title as any;
                this.pie.series[0].innerRadius = '30%';
                (this.grid as GridComponent).dataSource =
this.pie.series[0].dataSource;
                ((this.grid as GridComponent).columns[0] as any).headerText =
this.data[this.pointIndex].x;
                (this.grid as GridComponent).refresh();
                this.pie.refresh();
            } else if (this.pie?.series[0].name === 'Level 2') {
                this.pie.series[0].dataSource = this.data;
                this.pie.series[0].name = 'Level 1';
                this.pie.series[0].innerRadius = '0%';
                this.pie.title = 'Automobile Sales by Category';
                this.pie.annotations = null as any;
                this.pie.pointClick = this.onPointClick;
                (this.grid as GridComponent).dataSource =
this.pie.series[0].dataSource;
                ((this.grid as GridComponent).columns[0] as any).headerText =
'Vehicle';
                (this.grid as GridComponent).refresh();
                this.pie.refresh();
            }
        }
        (this.grid as GridComponent).dataSource = this.pie?.series[0].dataSource
as any;
    }
    public click(args: IMouseEventArgs | any) {
        if (this.pie?.series[0].name !== 'Level 3') {
            switch (args.pointIndex) {
                case 0:
                    this.pie!.series[0].dataSource = (this.data[0].z[0] as any).z;
                    this.pie!.title = 'SUV Sales by Years';
                    this.pie!.series[0].name = 'Level 3';
                    ((this.grid as GridComponent).columns[0] as any).headerText =
'Year';

```



```

        (this.grid as GridComponent ).refresh();
        this.pie?.refresh();
        break;
    }
    (this.grid as GridComponent ).dataSource =
this.pie?.series[0].dataSource as any;
    }
}
public onPointClick(args: IMouseEventArgs | any) {
    if (
        getElement(
            this.pie?.element.id +
            '_Series_' +
            args.seriesIndex +
            '_Point_' +
            args.pointIndex
        )
    ) {
        this.pie!.series[0].dataSource = this.data[args.pointIndex].z;
        this.pie!.title = this.data[args.pointIndex].z[0].title as any;
        this.pointIndex = args.pointIndex;
        this.pie!.series[0].name = 'Level 2';
        this.pie!.series[0].innerRadius = '30%';
        this.pie!.annotations = [
            {
                content:
                    '<div id="back" style="cursor:pointer;padding:3px;width:30px;
height:30px;">' +
                    '',
                region: 'Series',
                x: '50%',
                y: '50%',
            },
        ],
    };
    (this.grid as GridComponent ).dataSource = this.pie?.series[0].dataSource
as any;
    ((this.grid as GridComponent ).columns[0] as any).headerText =
this.data[args.pointIndex].x;
    (this.grid as GridComponent ).refresh();
    this.pie?.refresh();
}
constructor() {
    //code
}
}

```

MAIN.TS

```

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

```

Pyramid in Angular Accumulation chart component

To render a pyramid series, use the series [type](#) as `Pyramid` and inject `PyramidSeries` module into the [Link to the Video](#).

To know about pyramid, you can check on this video:

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { AccumulationChartModule } from '@syncfusion/ej2-angular-charts'
import { PyramidSeriesService, CategoryService, AccumulationDataLabelService }
from '@syncfusion/ej2-angular-charts'
import { Component, OnInit } from '@angular/core';
import { pyramidData } from './datasource';
@Component({
  imports: [
    AccumulationChartModule
  ],
  providers: [PyramidSeriesService, CategoryService,
  AccumulationDataLabelService],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-accumulationchart id="chart-container">
    <e-accumulation-series-collection>
      <e-accumulation-series type='Pyramid' [dataSource]='pyramidData'
xName='x' yName='y'></e-accumulation-series>
    </e-accumulation-series-collection>
  </ejs-accumulationchart>`
})
export class AppComponent implements OnInit {
  public pyramidData?: Object[];
  public enableSmartLabels?: boolean;
  ngOnInit(): void {
    this.pyramidData = pyramidData;
  }
}
```

MAIN.TS

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

Mode

The Pyramid chart supports linear and surface modes of rendering. The default type of the `pyramidMode` is `linear`.

APP.COMPONENT.TS

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

```

import { PyramidSeriesService, CategoryService, AccumulationDataLabelService }
  from '@syncfusion/ej2-angular-charts'
import { Component, OnInit } from '@angular/core';
import { pyramidData } from './datasource';
import {
  IAccPointRenderEventArgs,
} from '@syncfusion/ej2-charts';
@Component({
  imports: [
    AccumulationChartModule
  ],
  providers: [PyramidSeriesService, CategoryService,
    AccumulationDataLabelService],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-accumulationchart id="chart-container"
(pointRender)="onPointRender($event)">
    <e-accumulation-series-collection>
      <e-accumulation-series type='Pyramid' [dataSource]='pyramidData'
xName='x' yName='y' [gapRatio]="gapRatio"
    ></e-accumulation-series>
    </e-accumulation-series-collection>
  </ejs-accumulationchart>`,
})
export class AppComponent implements OnInit {
  public gapRatio: number = 0.2;
  public onPointRender = (args: IAccPointRenderEventArgs) => {
    if ((args.point.x as string).indexOf('Downloaded') > -1) {
      args.fill = '#D3D3D3';
    } else {
      args.fill = '#597cf9';
    }
  };
  public pyramidData = pyramidData;
  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));

```

Size

The size of the pyramid chart can be customized by using the **width** and **height** properties.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { AccumulationChartModule } from '@syncfusion/ej2-angular-charts'
import { PyramidSeriesService, CategoryService, AccumulationDataLabelService }
  from '@syncfusion/ej2-angular-charts'
import { Component, OnInit } from '@angular/core';

```

```
import { pyramidData } from './datasource';
@Component({
  imports: [
    AccumulationChartModule
  ],
  providers: [PyramidSeriesService, CategoryService,
    AccumulationDataLabelService],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-accumulationchart id="chart-container">
    <e-accumulation-series-collection>
      <e-accumulation-series width='60%' height='80%' type='Pyramid'
[dataSource]='pyramidData' xName='x' yName='y'></e-accumulation-series>
    </e-accumulation-series-collection>
  </ejs-accumulationchart>`
})
export class AppComponent implements OnInit {
  public pyramidData?: Object[];
  ngOnInit(): void {
    this.pyramidData = pyramidData;
  }
}
```

MAIN.TS

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

Gap Between the Segments

Pyramid chart provides options to customize the space between the segments by using the `gapRatio` property of the series. It ranges from 0 to 1.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { AccumulationChartModule } from '@syncfusion/ej2-angular-charts';
import { PyramidSeriesService, CategoryService, AccumulationDataLabelService }
from '@syncfusion/ej2-angular-charts';
import { Component, OnInit } from '@angular/core';
import { pyramidData } from './datasource';
@Component({
  imports: [
    AccumulationChartModule
  ],
  providers: [PyramidSeriesService, CategoryService,
    AccumulationDataLabelService],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-accumulationchart id="chart-container">
    <e-accumulation-series-collection>
      <e-accumulation-series type='Pyramid' [dataSource]='pyramidData'
xName='x' yName='y' [gapRatio]="gapRatio"></e-accumulation-series>
    </e-accumulation-series-collection>
  </ejs-accumulationchart>`
})
export class AppComponent implements OnInit {
  public pyramidData?: Object[];
  public gapRatio?: number;
  ngOnInit(): void {
    this.pyramidData = pyramidData;
    this.gapRatio = 0.5;
  }
}
```

```

        </e-accumulation-series-collection>
    </ejs-accumulationchart>`
    })
    export class AppComponent implements OnInit {
        public pyramidData?: Object[];
        public gapRatio: number = 0.2;
        ngOnInit(): void {
            this.pyramidData = pyramidData;
        }
    }
}

```

MAIN.TS

```

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

```

Explode

Points can be exploded on mouse click by setting the `explode` property to true. You can also explode the point on load using `explodeIndex`. Explode distance can be set by using `explodeOffset` property.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { AccumulationChartModule } from '@syncfusion/ej2-angular-charts'
import { PyramidSeriesService, CategoryService, AccumulationDataLabelService }
from '@syncfusion/ej2-angular-charts'
import { Component, OnInit } from '@angular/core';
import { pyramidData } from './datasource';
@Component({
    imports: [
        AccumulationChartModule
    ],
    providers: [PyramidSeriesService, CategoryService, AccumulationDataLabelService],
    standalone: true,
    selector: 'app-container',
    template: `<ejs-accumulationchart id="chart-container">
        <e-accumulation-series-collection>
            <e-accumulation-series type='Pyramid' [dataSource]='pyramidData'
xName='x' yName='y' [explode]='explode' [explodeAll]='explodeAll'
[explodeIndex]='explodeIndex'></e-accumulation-series>
        </e-accumulation-series-collection>
    </ejs-accumulationchart>`
})
export class AppComponent implements OnInit {
    public pyramidData?: Object[];
    public explode?: boolean;
    public explodeIndex?: number;
    public explodeAll?: boolean;
    explodeOffset?: string;
    ngOnInit(): void {
        this.explode = true;
    }
}

```

```

        this.explodeIndex = 3;
        this.explodeOffset = '30px';
        this.explodeAll = false;
        this.pyramidData = pyramidData;
    }
}

```

MAIN.TS

```

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

```

Customization

Individual points can be customized using the `pointRender` event.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { AccumulationChartModule } from '@syncfusion/ej2-angular-charts'
import { PyramidSeriesService, CategoryService, AccumulationDataLabelService }
from '@syncfusion/ej2-angular-charts'
import { Component, OnInit } from '@angular/core';
import { pyramidData } from './datasource';
import { IAccTextRenderEventArgs, IAccPointRenderEventArgs } from
 '@syncfusion/ej2-charts';
@Component({
  imports: [
    AccumulationChartModule
  ],
  providers: [PyramidSeriesService, CategoryService,
  AccumulationDataLabelService],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-accumulationchart id="chart-container"
(pointRender)="onPointRender($event)">
    <e-accumulation-series-collection>
      <e-accumulation-series type='Pyramid' [dataSource]='pyramidData'
xName='x' yName='y' [dataLabel]='datalabel' [gapRatio]="gapRatio"
    ></e-accumulation-series>
    </e-accumulation-series-collection>
  </ejs-accumulationchart>`
})
export class AppComponent implements OnInit {
  public pyramidData?: Object[];
  public gapRatio: number = 0.2;
  public onPointRender?: Function;
  ngOnInit(): void {
    this.onPointRender = (args: IAccPointRenderEventArgs) => {
      if ((args.point.x as string).indexOf('Downloaded') > -1) {
        args.fill = '#D3D3D3';
      }
      else {

```

```

        args.fill = '#597cf9';
    }
    }
    this.pyramidData = pyramidData;
}
}

```

MAIN.TS

```

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

```

Funnel in Angular Accumulation chart component

To render a funnel series, use the series [type](#) as **Funnel** and inject, the **FunnelSeries** module into the [Link to the Video](#).

To know about funnel charts, you can check on this video:

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { AccumulationChartModule } from '@syncfusion/ej2-angular-charts'
import { FunnelSeriesService, AccumulationTooltipService, CategoryService,
AccumulationDataLabelService } from '@syncfusion/ej2-angular-charts'
import { Component, OnInit } from '@angular/core';
import { funnelData } from './datasource';
@Component({
  imports: [
    AccumulationChartModule
  ],
  providers: [FunnelSeriesService, AccumulationTooltipService, CategoryService,
AccumulationDataLabelService ],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-accumulationchart id="chart-container">
    <e-accumulation-series-collection>
      <e-accumulation-series width='330px' type='Funnel'
[dataSource]='funneldata' xName='x' yName='y'></e-accumulation-series>
    </e-accumulation-series-collection>
  </ejs-accumulationchart>`
})
export class AppComponent implements OnInit {
  public funneldata?: Object[];
  ngOnInit(): void {
    this.funneldata = funnelData;
  }
}

```

MAIN.TS

```

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

```

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

Size

The size of the funnel chart can be customized by using the `width` and `height` properties.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { AccumulationChartModule } from '@syncfusion/ej2-angular-charts'
import { FunnelSeriesService, AccumulationTooltipService, CategoryService,
AccumulationDataLabelService } from '@syncfusion/ej2-angular-charts'
import { Component, OnInit } from '@angular/core';
import { funnelData } from './datasource';
@Component({
  imports: [
    AccumulationChartModule
  ],
  providers: [FunnelSeriesService, AccumulationTooltipService, CategoryService,
AccumulationDataLabelService ],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-accumulationchart id="chart-container">
    <e-accumulation-series-collection>
      <e-accumulation-series width='60%' height='80%' type='Funnel'
[dataSource]='funneldata' xName='x' yName='y' [dataLabel]='datalabel'></e-
accumulation-series>
    </e-accumulation-series-collection>
  </ejs-accumulationchart>`
})
export class AppComponent implements OnInit {
  public funneldata?: Object[];
  datalabel?: Object;
  ngOnInit(): void {
    this.datalabel = { visible: true, name: 'text', position: 'Inside' };
    this.funneldata = funnelData;
  }
}
```

MAIN.TS

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

Neck Size

The funnel's neck size can be customized by using the `neckWidth` and `neckHeight` properties.

APP.COMPONENT.TS

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



```

import { FunnelSeriesService, AccumulationTooltipService, CategoryService,
AccumulationDataLabelService } from '@syncfusion/ej2-angular-charts'
import { Component, OnInit } from '@angular/core';
import { funnelData } from './datasource';
@Component({
  imports: [
    AccumulationChartModule
  ],
  providers: [FunnelSeriesService, AccumulationTooltipService, CategoryService,
AccumulationDataLabelService ],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-accumulationchart id="chart-container">
    <e-accumulation-series-collection>
      <e-accumulation-series type='Funnel' [dataSource]='funneldata'
xName='x' yName='y' [dataLabel]='datalabel' [neckWidth]='neckWidth'
[neckHeight]='neckHeight'></e-accumulation-series>
    </e-accumulation-series-collection>
  </ejs-accumulationchart>`
})
export class AppComponent implements OnInit {
  public funneldata?: Object[];
  public neckWidth: string = '25%';
  public neckHeight?: '5%'
  datalabel: any;
  ngOnInit(): void {
    this.funneldata = funnelData;
  }
}

```

MAIN.TS

```

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

```

Gap between the segments

Funnel chart provides options to customize the space between the segments by using the **gapRatio** property of the series. It ranges from 0 to 1.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { AccumulationChartModule } from '@syncfusion/ej2-angular-charts'
import { FunnelSeriesService, AccumulationTooltipService, CategoryService,
AccumulationDataLabelService } from '@syncfusion/ej2-angular-charts'
import { Component, OnInit } from '@angular/core';
import { funnelData } from './datasource';
@Component({
  imports: [
    AccumulationChartModule
  ],

```

```

providers: [FunnelSeriesService, AccumulationTooltipService, CategoryService,
AccumulationDataLabelService ],
standalone: true,
selector: 'app-container',
template: `<ejs-accumulationchart id="chart-container">
  <e-accumulation-series-collection>
    <e-accumulation-series type='Funnel' [dataSource]='funneldata'
xName='x' yName='y' [dataLabel]='datalabel' [gapRatio]="gapRatio"></e-
accumulation-series>
  </e-accumulation-series-collection>
</ejs-accumulationchart>`
})
export class AppComponent implements OnInit {
  public funneldata?: Object[];
  public gapRatio: number = 0.08;
  datalabel: any;
  ngOnInit(): void {
    this.funneldata = funnelData;
  }
}

```

MAIN.TS

```

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

```

Explode

Points can be exploded on mouse click by setting the **explode** property to true. You can also explode the point on load using **explodeIndex**. Explode distance can be set by using **explodeOffset** property.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { AccumulationChartModule } from '@syncfusion/ej2-angular-charts'
import { FunnelSeriesService, AccumulationTooltipService, CategoryService,
AccumulationDataLabelService } from '@syncfusion/ej2-angular-charts'
import { Component, OnInit } from '@angular/core';
import { funnelData } from './datasource';
@Component({
  imports: [
    AccumulationChartModule
  ],
  providers: [FunnelSeriesService, AccumulationTooltipService, CategoryService,
AccumulationDataLabelService ],
standalone: true,
selector: 'app-container',
template: `<ejs-accumulationchart id="chart-container">
  <e-accumulation-series-collection>
    <e-accumulation-series type='Funnel' [dataSource]='funnelData'
xName='x' yName='y' [explode]='explode' [explodeAll]='explodeAll'
[explodeIndex]='explodeIndex'></e-accumulation-series>
  </e-accumulation-series-collection>

```

```

    </ejs-accumulationchart>`
  })
  export class AppComponent implements OnInit {
    public funnelData?: Object[];
    public explode?: boolean;
    public explodeIndex?: number;
    public explodeAll?: boolean;
    explodeOffset?: string;
    ngOnInit(): void {
      this.explode = true;
      this.explodeIndex = 3;
      this.explodeOffset = '30px';
      this.explodeAll = false;
      this.funnelData = funnelData;
    }
  }
}

```

MAIN.TS

```

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

```

Smart Data Label

It provides the data label smart arrangement of the funnel and pyramid series. The overlap data label will be placed on left side of the funnel/pyramid series.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { AccumulationChartModule } from '@syncfusion/ej2-angular-charts'
import { FunnelSeriesService, AccumulationTooltipService, CategoryService,
AccumulationDataLabelService } from '@syncfusion/ej2-angular-charts'
import { Component, OnInit } from '@angular/core';
import { funnelDataSource } from './datasource';
@Component({
  imports: [
    AccumulationChartModule
  ],
  providers: [FunnelSeriesService, AccumulationTooltipService, CategoryService,
AccumulationDataLabelService ],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-accumulationchart id="chart-container"
[legendSettings]="legendSettings" [tooltip]="tooltip" title="Top population
countries in the world 2017">
    <e-accumulation-series-collection>
      <e-accumulation-series type='Funnel' [dataSource]='funnelData'
xName='x' yName='y' neckWidth='15%'
      neckHeight='18%' [dataLabel]='dataLabel' name="2017 Population"
explode="false"></e-accumulation-series>
    </e-accumulation-series-collection>
  </ejs-accumulationchart>`
})

```

```

})
export class AppComponent implements OnInit {
  public funnelData?: Object[];
  public dataLabel?: Object;
  public legendSettings?: Object;
  public tooltip?: Object;
  ngOnInit(): void {
    this.funnelData = funnelDataSource;
    this.dataLabel = {
      visible: true, position: 'Outside',
      connectorStyle: { length: '6%' }, name: 'text',
    };
    this.legendSettings = {visible: false};
    this.tooltip = {
      enable: true, format: '${point.x} : <b>${point.y}</b>'
    };
  }
}

```

MAIN.TS

```

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

```

Customization

Individual points can be customized using the `pointRender` event.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { AccumulationChartModule } from '@syncfusion/ej2-angular-charts'
import { FunnelSeriesService, AccumulationTooltipService, CategoryService,
AccumulationDataLabelService } from '@syncfusion/ej2-angular-charts'
import { Component, OnInit } from '@angular/core';
import { funnelData } from './datasource';
import { IAccTextRenderEventArgs, IAccPointRenderEventArgs } from
'@syncfusion/ej2-charts';
@Component({
  imports: [
    AccumulationChartModule
  ],
  providers: [FunnelSeriesService, AccumulationTooltipService, CategoryService,
AccumulationDataLabelService ],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-accumulationchart id="chart-container"
(pointRender)="onPointRender($event)">
  <e-accumulation-series-collection>
    <e-accumulation-series type='Funnel' [dataSource]='funneldata'
xName='x' yName='y' [dataLabel]='dataLabel' [gapRatio]="gapRatio"
    ></e-accumulation-series>
  </e-accumulation-series-collection>

```

```

    </ejs-accumulationchart>`
  })
  export class AppComponent implements OnInit {
    public funneldata?: Object[];
    public gapRatio: number = 0.08;
    public onPointRender: Function | any;
    datalabel: any;
    ngOnInit(): void {
      this.onPointRender = (args: IAccPointRenderEventArgs) => {
        if ((args.point.x as string).indexOf('Downloaded') > -1) {
          args.fill = '#D3D3D3';
        }
        else {
          args.fill = '#597cf9';
        }
      }
      this.funneldata = funnelData;
    }
  }
}

```

MAIN.TS

```

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

```

See Also

- [Data label](#)
- [Grouping](#)

Data label in Angular Accumulation chart component

Data label can be added to a chart series by enabling the [visible](#) option in the dataLabel property.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { AccumulationChartModule } from '@syncfusion/ej2-angular-charts'
import { PieSeriesService, AccumulationDataLabelService } from '@syncfusion/ej2-angular-charts'
import { Component, OnInit } from '@angular/core';
import { labelData } from './datasource';
@Component({
  imports: [
    AccumulationChartModule
  ],
  providers: [PieSeriesService, AccumulationDataLabelService],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-accumulationchart id="chart-container">
    <e-accumulation-series-collection>

```

```

        <e-accumulation-series [dataSource]='piedata' xName='x' yName='y'
[dataLabel]='datalabel'></e-accumulation-series>
    </e-accumulation-series-collection>
</ejs-accumulationchart>`
    })
    export class AppComponent implements OnInit {
        public piedata?: Object[];
        public datalabel?: Object;
        ngOnInit(): void {
            this.datalabel = { visible: true };
            this.piedata = labelData;
        }
    }
}

```

MAIN.TS

```

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

```

Note: To use the data label feature, inject the **DataLabel** into the **@NgModule.providers**.

Positioning

Accumulation chart provides support for placing the data label either **inside** or **outside** the chart.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { AccumulationChartModule } from '@syncfusion/ej2-angular-charts'
import { PieSeriesService, AccumulationDataLabelService } from
 '@syncfusion/ej2-angular-charts'
import { Component, OnInit } from '@angular/core';
import { labelData } from './datasource';
@Component({
    imports: [
        AccumulationChartModule
    ],
    providers: [PieSeriesService, AccumulationDataLabelService],
    standalone: true,
    selector: 'app-container',
    template: `<ejs-accumulationchart id="chart-container">
        <e-accumulation-series-collection>
            <e-accumulation-series [dataSource]='piedata' xName='x' yName='y'
[dataLabel]='datalabel'></e-accumulation-series>
        </e-accumulation-series-collection>
    </ejs-accumulationchart>`
})
export class AppComponent implements OnInit {
    public piedata?: Object[];
    public datalabel?: Object;
    ngOnInit(): void {
        this.datalabel = { visible: true, position: 'Outside' };
        this.piedata = labelData;
    }
}

```

```
}

```

MAIN.TS

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

```

DataLabel rotation

Using **angle** property, you can rotate the data label by its given angle.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { AccumulationChartModule } from '@syncfusion/ej2-angular-charts'
import { PieSeriesService, AccumulationDataLabelService } from
 '@syncfusion/ej2-angular-charts'
import { Component, OnInit } from '@angular/core';
import { labelData } from './datasource';
@Component({
  imports: [
    AccumulationChartModule
  ],
  providers: [PieSeriesService, AccumulationDataLabelService],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-accumulationchart id="chart-container">
    <e-accumulation-series-collection>
      <e-accumulation-series [dataSource]='piedata' xName='x' yName='y'
[dataLabel]='datalabel'></e-accumulation-series>
    </e-accumulation-series-collection>
  </ejs-accumulationchart>`
})
export class AppComponent implements OnInit {
  public piedata?: Object[];
  public datalabel?: Object;
  ngOnInit(): void {
    this.datalabel = { visible: true, angle: 90, enableRotation: true },
    this.piedata = labelData;
  }
}
```

MAIN.TS

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

```

Note: when the **enableRotation** is true, the data label is rotated along the slice.

Smart labels

Data labels will be arranged smartly without overlapping with each other. You can enable or disable this feature using the [enableSmartLabels](#) property.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { AccumulationChartModule } from '@syncfusion/ej2-angular-charts'
import { PieSeriesService, AccumulationDataLabelService } from
 '@syncfusion/ej2-angular-charts'
import { Component, OnInit } from '@angular/core';
import { labelData } from './datasource';
@Component({
  imports: [
    AccumulationChartModule
  ],
  providers: [PieSeriesService, AccumulationDataLabelService],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-accumulationchart id="chart-container"
[enableSmartLabels]='enableSmartLabels'>
  <e-accumulation-series-collection>
    <e-accumulation-series [dataSource]='piedata' xName='x' yName='y'
[dataLabel]='datalabel'></e-accumulation-series>
  </e-accumulation-series-collection>
</ejs-accumulationchart>`
})
export class AppComponent implements OnInit {
  public piedata?: Object[];
  public datalabel?: Object;
  public enableSmartLabels?: boolean;
  ngOnInit(): void {
    this.datalabel = { visible: true, name: 'text', position: 'Outside'
  };

    this.enableSmartLabels = true;
    this.piedata = labelData;
  }
}
```

MAIN.TS

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

Format

Data label for the accumulation chart can be formatted using [format](#) property. You can use the global formatting options, such as 'n', 'p', and 'c'.

APP.COMPONENT.TS

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



```

import { AccumulationChartModule } from '@syncfusion/ej2-angular-charts'
import { PieSeriesService, AccumulationDataLabelService } from
 '@syncfusion/ej2-angular-charts'
import { Component, OnInit } from '@angular/core';
import { labelData } from './datasource';
@Component({
  imports: [
    AccumulationChartModule
  ],
  providers: [PieSeriesService, AccumulationDataLabelService],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-accumulationchart id="chart-container"
[enableSmartLabels]='enableSmartLabels'>
    <e-accumulation-series-collection>
      <e-accumulation-series [dataSource]='piedata' xName='x' yName='y'
[dataLabel]='datalabel'></e-accumulation-series>
    </e-accumulation-series-collection>
  </ejs-accumulationchart>`
})
export class AppComponent implements OnInit {
  public piedata?: Object[];
  public datalabel?: Object;
  public enableSmartLabels?: boolean;
  ngOnInit(): void {
    this.datalabel = { visible: true, format: 'n2' };
    this.enableSmartLabels = true;
    this.piedata = labelData;
  }
}

```

MAIN.TS

```

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

```

Value	Format	Resultant Value	Description
1000	n1	1000.0	The number is rounded to 1 decimal place.
1000	n2	1000.00	The number is rounded to 2 decimal places.
1000	n3	1000.000	The number is rounded to 3 decimal place.
0.01	p1	1.0%	The number is converted to percentage with 1 decimal place.
0.01	p2	1.00%	The number is converted to percentage with 2 decimal place.
0.01	p3	1.000%	The number is converted to percentage with 3 decimal place.
1000	c1	\$1000.0	The currency symbol is appended to number and number is rounded to 1 decimal place.

1000	c2	\$1000.00	The currency symbol is appended to number and number is rounded to 2 decimal place.
------	----	-----------	---

DataLabel template

Label content can be formatted by using the template option. Inside the template, you can add the placeholder text `${point.x}` and `${point.y}` to display corresponding data points x & y value. Using [template](#) property, you can set data label template in chart.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { AccumulationChartModule } from '@syncfusion/ej2-angular-charts'
import { PieSeriesService, AccumulationDataLabelService } from
 '@syncfusion/ej2-angular-charts'
import { Component, OnInit } from '@angular/core';
import { labelData } from './datasource';
@Component({
  imports: [
    AccumulationChartModule
  ],
  providers: [PieSeriesService, AccumulationDataLabelService],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-accumulationchart id="chart-container"
[enableSmartLabels]='enableSmartLabels'>
  <e-accumulation-series-collection>
    <e-accumulation-series [dataSource]='piedata' xName='x' yName='y'
[dataLabel]='datalabel'></e-accumulation-series>
  </e-accumulation-series-collection>
</ejs-accumulationchart>`
})
export class AppComponent implements OnInit {
  public piedata?: Object[];
  public datalabel?: Object;
  public enableSmartLabels?: boolean;
  ngOnInit(): void {
    this.datalabel = { visible: true, name: 'text', position: 'Outside',
template: '<div>${point.x}</div><div>${point.y}</div>' };
    this.enableSmartLabels = true;
    this.piedata = labelData;
  }
}
```

MAIN.TS

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

Connector Line

Connector line will be visible when the data label is placed **outside** the chart.

The connector line can be customized using the **type**, **color**, **width**, **length** and **dashArray** properties

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { AccumulationChartModule } from '@syncfusion/ej2-angular-charts'
import { PieSeriesService, AccumulationDataLabelService } from
 '@syncfusion/ej2-angular-charts'
import { Component, OnInit } from '@angular/core';
import { labelData } from './datasource';
@Component({
  imports: [
    AccumulationChartModule
  ],
  providers: [PieSeriesService, AccumulationDataLabelService],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-accumulationchart id="chart-container"
[enableSmartLabels]='enableSmartLabels'>
  <e-accumulation-series-collection>
    <e-accumulation-series [dataSource]='piedata' xName='x' yName='y'
[dataLabel]='datalabel'></e-accumulation-series>
  </e-accumulation-series-collection>
</ejs-accumulationchart>`
})
export class AppComponent implements OnInit {
  public piedata?: Object[];
  public datalabel?: Object;
  public enableSmartLabels?: boolean;
  ngOnInit(): void {
    this.datalabel = { visible: true, name: 'text', position: 'Outside',
      connectorStyle:{
        //Length of the connector line in pixels
        length: '50px',
        //Width of the connector line in pixels
        width: 2,
        //dashArray of the connector line
        dashArray: '5,3',
        //Color of the connector line
        color: '#f4429e',
        //Specifies the type of the connector line either Line or
Curve
        type: 'Curve'
      }
    };
    this.enableSmartLabels = true;
    this.piedata = labelData;
  }
}

```

MAIN.TS

```

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

```

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

Text Mapping

The fill color and the text in the data source can be mapped to the chart using `pointColorMapping` in series and `name` in data label respectively.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { AccumulationChartModule } from '@syncfusion/ej2-angular-charts'
import { PieSeriesService, AccumulationLegendService,
  AccumulationTooltipService, AccumulationAnnotationService,
  AccumulationDataLabelService } from '@syncfusion/ej2-angular-charts'
import { Component, OnInit } from '@angular/core';
import { dataMapping } from './datasource';
@Component({
  imports: [
    AccumulationChartModule
  ],
  providers: [PieSeriesService, AccumulationLegendService,
    AccumulationTooltipService, AccumulationDataLabelService,
    AccumulationAnnotationService],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-accumulationchart id="chart-container">
    <e-accumulation-series-collection>
      <e-accumulation-series [dataSource]='piedata' xName='x' yName='y'
[dataLabel]='datalabel'></e-accumulation-series>
    </e-accumulation-series-collection>
  </ejs-accumulationchart>`
})
export class AppComponent implements OnInit {
  public piedata?: Object[];
  public datalabel?: Object;
  ngOnInit(): void {
    this.piedata = dataMapping;
    this.datalabel = { visible: true, name: 'text', position: 'Outside'
  };
}
```

MAIN.TS

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

Customization

Individual text can be customized using the `textRender` event.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { AccumulationChartModule } from '@syncfusion/ej2-angular-charts'
import { PieSeriesService, AccumulationDataLabelService } from
 '@syncfusion/ej2-angular-charts'
import { Component, OnInit } from '@angular/core';
import { labelData } from '../datasource';
import { IAccTextRenderEventArgs } from '@syncfusion/ej2-charts';
@Component({
  imports: [
    AccumulationChartModule
  ],
  providers: [PieSeriesService, AccumulationDataLabelService],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-accumulationchart id="chart-container"
(textRender)="onTextRender($event)">
    <e-accumulation-series-collection>
      <e-accumulation-series [dataSource]='piedata' xName='x' yName='y'
[dataLabel]='datalabel'></e-accumulation-series>
    </e-accumulation-series-collection>
  </ejs-accumulationchart>`
})
export class AppComponent implements OnInit {
  public piedata?: Object[];
  public datalabel?: Object;
  public onTextRender(args: IAccTextRenderEventArgs): void {
    if (args.text === '13.5') {
      args.color = 'red';
      args.border.width = 1;
    }
  }
  ngOnInit(): void {
    this.datalabel = { visible: true };
    this.piedata = labelData;
  }
}

```

MAIN.TS

```

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

```

Text wrap

When the data label text exceeds the container, the text can be wrapped by using [textWrap](#) property. End user can also wrap the data label text based on [maxWidth](#) property.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { AccumulationChartModule } from '@syncfusion/ej2-angular-charts'

```

```
import { PieSeriesService, AccumulationDataLabelService } from
 '@syncfusion/ej2-angular-charts'
import { Component, OnInit } from '@angular/core';
import { labelData } from './datasource';
@Component({
  imports: [
    AccumulationChartModule
  ],
  providers: [PieSeriesService, AccumulationDataLabelService],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-accumulationchart id="chart-container">
    <e-accumulation-series-collection>
      <e-accumulation-series [dataSource]='piedata' startAngle="270"
endAngle="90" innerRadius="40%" radius="100%" xName='x' yName='y'
[dataLabel]='datalabel'></e-accumulation-series>
    </e-accumulation-series-collection>
  </ejs-accumulationchart>`
})
export class AppComponent implements OnInit {
  public piedata: Object[] = [];
  public datalabel?: Object;
  ngOnInit(): void {
    this.datalabel = { visible: true, position: 'Inside', maxWidth: 100
, textWrap: 'Wrap', name: 'text', enableRotation: true };
    this.piedata = labelData;
  }
}
```

MAIN.TS

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

Show percentages in data labels of pie chart

You can show the percentages in data labels of pie chart using `textRender` event and `template` option.

Using `textRender` event

You can customize the data label of pie chart using `textRender` event as follows to show percentage.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { AccumulationChartModule } from '@syncfusion/ej2-angular-charts'
import { PieSeriesService, AccumulationDataLabelService } from
 '@syncfusion/ej2-angular-charts'
import { Component, OnInit } from '@angular/core';
import { labelData } from './datasource';
import { IAccTextRenderEventArgs } from '@syncfusion/ej2-angular-charts';
@Component({
  imports: [
    AccumulationChartModule
  ],
```

```

providers: [PieSeriesService, AccumulationDataLabelService],
standalone: true,
  selector: 'app-container',
  template: `<ejs-accumulationchart id="chart-container"
(textRender)="onTextRender($event)">
    <e-accumulation-series-collection>
      <e-accumulation-series [dataSource]='piedata' xName='x' yName='y'
[dataLabel]='datalabel'></e-accumulation-series>
    </e-accumulation-series-collection>
  </ejs-accumulationchart>`
})
export class AppComponent implements OnInit {
  public piedata?: Object[];
  public datalabel?: Object;
  onTextRender: Function | any;
  ngOnInit(): void {
    this.datalabel = { visible: true };
    this.piedata = labelData;
    this.onTextRender = (args: IAccTextRenderEventArgs) => {
      args.text = args.point.percentage + "%";
    }
  }
}

```

MAIN.TS

```

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

```

Using template

You can display the percentage values in data label of pie chart using **template** option

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { AccumulationChartModule } from '@syncfusion/ej2-angular-charts'
import { PieSeriesService, AccumulationDataLabelService } from
'@syncfusion/ej2-angular-charts'
import { Component, OnInit } from '@angular/core';
import { labelData } from './datasource';
@Component({
  imports: [
    AccumulationChartModule
  ],
  providers: [PieSeriesService, AccumulationDataLabelService],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-accumulationchart id="chart-container">
    <e-accumulation-series-collection>
      <e-accumulation-series [dataSource]='piedata' xName='x' yName='y'
[dataLabel]='datalabel'></e-accumulation-series>
    </e-accumulation-series-collection>
  </ejs-accumulationchart>`
})

```

```

    })
    export class AppComponent implements OnInit {
        public piedata?: Object[];
        public datalabel?: Object;
        ngOnInit(): void {
            this.datalabel = { visible: true, template: "<div
id='dataLabelTemplate'>${point.percentage}%</div>" };
            this.piedata = labelData;
        }
    }
}

```

MAIN.TS

```

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

```

Grouping in Angular Accumulation chart component

You can club/group few points of the series based on [groupBy](#) property. For example, if the club value is 11, then the points with value less than 11 is grouped together and will be showed as a single point with label **others**. The property also takes value in percentage (percentage of total data points value).

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { AccumulationChartModule } from '@syncfusion/ej2-angular-charts'
import { PieSeriesService, AccumulationDataLabelService } from
 '@syncfusion/ej2-angular-charts'
import { Component, OnInit } from '@angular/core';
@Component({
    imports: [
        AccumulationChartModule
    ],
    providers: [PieSeriesService, AccumulationDataLabelService],
    standalone: true,
    selector: 'app-container',
    template: `<ejs-accumulationchart id="chart-container" >
        <e-accumulation-series-collection>
            <e-accumulation-series [dataSource]='piedata' xName='x' yName='y'
groupTo='11' [dataLabel]='datalabel'></e-accumulation-series>
        </e-accumulation-series-collection>
    </ejs-accumulationchart>`
})
export class AppComponent implements OnInit {
    public piedata?: Object[];
    public datalabel?: Object;
    ngOnInit(): void {
        this.datalabel = { visible: true, name: 'text', position: 'Outside'
    };

        this.piedata = [
            { x: 'Jan', y: 3, text: 'Jan: 3' }, { x: 'Feb', y: 3.5, text:
'Feb: 3.5' },

```



```

        { x: 'Mar', y: 7, text: 'Mar: 7' }, { x: 'Apr', y: 13.5,
text: 'Apr: 13.5' },
        { x: 'May', y: 19, text: 'May: 19' }, { x: 'Jun', y: 23.5,
text: 'Jun: 23.5' },
        { x: 'Jul', y: 26, text: 'Jul: 26' }, { x: 'Aug', y: 25,
text: 'Aug: 25' },
        { x: 'Sep', y: 21, text: 'Sep: 21' }, { x: 'Oct', y: 15,
text: 'Oct: 15' }]];
    }
}

```

MAIN.TS

```

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

```

Broken slice

You can visualize all points available in club/group points by clicking on the grouped point. For example, if 5 points are grouped together it will be showed as a single slice with label **others**. If we click on **others** slice it will explode and broke into 5 separate slices.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { AccumulationChartModule } from '@syncfusion/ej2-angular-charts'
import { PieSeriesService, AccumulationDataLabelService } from
'@syncfusion/ej2-angular-charts'
import { Component, OnInit } from '@angular/core';
@Component({
  imports: [
    AccumulationChartModule
  ],
  providers: [PieSeriesService, AccumulationDataLabelService],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-accumulationchart id="chart-container" >
    <e-accumulation-series-collection>
      <e-accumulation-series [dataSource]='piedata' xName='x' yName='y'
groupTo='11' [dataLabel]='datalabel' [explode]='explode'
[explodeOffset]='explodeOffset'></e-accumulation-series>
    </e-accumulation-series-collection>
  </ejs-accumulationchart>`
})
export class AppComponent implements OnInit {
  public piedata?: Object[];
  public datalabel?: Object;
  public explode?: boolean;
  public explodeOffset?: string;
  ngOnInit(): void {
    this.datalabel = { visible: true, name: 'text', position: 'Outside'
  };
  this.piedata = [

```

```

        { x: 'Jan', y: 3, text: 'Jan: 3' }, { x: 'Feb', y: 3.5, text:
'Feb: 3.5' },
        { x: 'Mar', y: 7, text: 'Mar: 7' }, { x: 'Apr', y: 13.5,
text: 'Apr: 13.5' },
        { x: 'May', y: 19, text: 'May: 19' }, { x: 'Jun', y: 23.5,
text: 'Jun: 23.5' },
        { x: 'Jul', y: 26, text: 'Jul: 26' }, { x: 'Aug', y: 25,
text: 'Aug: 25' },
        { x: 'Sep', y: 21, text: 'Sep: 21' }, { x: 'Oct', y: 15,
text: 'Oct: 15' },
        { x: 'Nov', y: 9, text: 'Nov: 9' }, { x: 'Dec', y: 3.5, text:
'Dec: 3.5' }
    ];
    this.explode = true;
    this.explodeOffset = '10%';
}
}

```

MAIN.TS

```

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

```

Group Mode

Slice can also be grouped based on number of points by specifying the `groupMode` to Point. For example, if the group to value is 11, accumulation chart will show 1st 11 points and will group remaining entries in the collection as a single point.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { AccumulationChartModule } from '@syncfusion/ej2-angular-charts'
import { PieSeriesService, AccumulationDataLabelService } from
'@syncfusion/ej2-angular-charts'
import { Component, OnInit } from '@angular/core';
import { IAccTextRenderEventArgs, IAccPointRenderEventArgs } from
'@syncfusion/ej2-charts';
import { groupData } from './datasource';
@Component({
  imports: [
    AccumulationChartModule
  ],
  providers: [PieSeriesService, AccumulationDataLabelService],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-accumulationchart id="chart-container"
(textRender)="onTextRender($event)" (pointRender)="onPointRender($event)">
    <e-accumulation-series-collection>
      <e-accumulation-series [dataSource]='piedata' xName='x' yName='y'
[dataLabel]='datalabel' groupTo='4' groupMode='Point'></e-accumulation-
series>
    </e-accumulation-series-collection>
`
})

```

```

    </ejs-accumulationchart>`
  })
  export class AppComponent implements OnInit {
    public piedata?: Object[];
    public datalabel?: Object;
    public onTextRender: Function | any;;
    public onPointRender: Function | any;;
    ngOnInit(): void {
      this.datalabel = { visible: true, name: 'text', position: 'Outside'
    };

    this.onTextRender = (args: IAccTextRenderEventArgs) => {
      if (args.text.indexOf('Others') > -1) {
        args.color = 'red';
        args.border.width = 1;
      }
    }
    this.onPointRender = (args: IAccPointRenderEventArgs) => {
      if ((args.point.x as string).indexOf('Others') > -1) {
        args.fill = '#D3D3D3';
      }
    }
    this.piedata = groupData;
  }
}

```

MAIN.TS

```

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

```

Customization

You can customize the grouped point and its data label using `pointRender` and `textRender` event.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { AccumulationChartModule } from '@syncfusion/ej2-angular-charts'
import { PieSeriesService, AccumulationDataLabelService } from
 '@syncfusion/ej2-angular-charts'
import { Component, OnInit } from '@angular/core';
import { IAccTextRenderEventArgs, IAccPointRenderEventArgs } from
 '@syncfusion/ej2-charts';
import { groupData } from './datasource';
@Component({
  imports: [
    AccumulationChartModule
  ],
  providers: [PieSeriesService, AccumulationDataLabelService],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-accumulationchart id="chart-container"
(textRender)="onTextRender($event)" (pointRender)="onPointRender($event)">

```

```

        <e-accumulation-series-collection>
            <e-accumulation-series [dataSource]='piedata' xName='x' yName='y'
groupTo='11' [dataLabel]='datalabel'></e-accumulation-series>
        </e-accumulation-series-collection>
    </ejs-accumulationchart>`
    })
    export class AppComponent implements OnInit {
        public piedata?: Object[];
        public datalabel?: Object;
        public onTextRender: Function | any;;
        public onPointRender: Function | any;;
        ngOnInit(): void {
            this.datalabel = { visible: true, name: 'text', position: 'Outside'
        };

            this.onTextRender = (args: IAccTextRenderEventArgs) => {
                if (args.text.indexOf('Others') > -1) {
                    args.color = 'red';
                    args.border.width = 1;
                }
            }
            this.onPointRender = (args: IAccPointRenderEventArgs) => {
                if ((args.point.x as string).indexOf('Others') > -1) {
                    args.fill = '#D3D3D3';
                }
            }
            this.piedata = groupData;
        }
    }
}

```

MAIN.TS

```

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

```

Empty points in Angular Accumulation chart component

The data points those uses the **null** or **undefined** as value are considered as empty points. The empty data points are ignored and not plotted in the chart. You can customize those points, using the **emptyPointSettings** property in series. The default mode of the empty point is **Gap**. Other supported modes are **Average** and **Zero**.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { AccumulationChartModule, ExportService } from '@syncfusion/ej2-angular-charts'
import { PieSeriesService, AccumulationDataLabelService } from
 '@syncfusion/ej2-angular-charts'
import { Component, OnInit } from '@angular/core';
@Component({
    imports: [
        AccumulationChartModule
    ],

```

```

providers: [PieSeriesService, AccumulationDataLabelService, ExportService],
standalone: true,
  selector: 'app-container',
  template: `<ejs-accumulationchart id="chart-container">
    <e-accumulation-series-collection>
      <e-accumulation-series [dataSource]='piedata' xName='x' yName='y'
radius="100%"
        [emptyPointSettings]='empty'></e-accumulation-series>
    </e-accumulation-series-collection>
  </ejs-accumulationchart>`
})
export class AppComponent implements OnInit {
  public piedata?: Object[];
  public empty?: Object;
  ngOnInit(): void {
    this.piedata = [
      { x: 'Jan', y: 3, text: 'Jan: 3' }, { x: 'Feb', y: 3.5, text:
'Feb: 3.5' },
      { x: 'Mar', y: null, text: 'Mar: 7' }, { x: 'Apr', y: 13.5, text:
'Apr: 13.5' },
      { x: 'May', y: 19, text: 'May: 19' }, { x: 'Jun', y: 23.5, text:
'Jun: 23.5' },
      { x: 'Jul', y: 26, text: 'Jul: 26' }, { x: 'Aug', y: undefined,
text: 'Aug: 25' },
      { x: 'Sep', y: 21, text: 'Sep: 21' }, { x: 'Oct', y: 15, text:
'Oct: 15' }
    ];
    this.empty = {
      mode: 'Zero', fill: 'pink', border: { width: 2, color: 'black' }
    }
  }
}

```

MAIN.TS

```

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

```

Customization

Specific color for an empty point can be set by using the **fill** property in **emptyPointSettings** and the border for an empty point can be set by using the **border** property.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { AccumulationChartModule, ExportService } from '@syncfusion/ej2-
angular-charts'
import { PieSeriesService, AccumulationDataLabelService } from
'@syncfusion/ej2-angular-charts'
import { Component, OnInit } from '@angular/core';
@Component({

```

```

imports: [
    AccumulationChartModule
],
providers: [PieSeriesService, AccumulationDataLabelService, ExportService],
standalone: true,
selector: 'app-container',
template: `<ejs-accumulationchart id="chart-container">
    <e-accumulation-series-collection>
        <e-accumulation-series [dataSource]='piedata' xName='x' yName='y'
[emptyPointSettings]='empty'></e-accumulation-series>
    </e-accumulation-series-collection>
</ejs-accumulationchart>`
})
export class AppComponent implements OnInit {
    public piedata?: Object[];
    public empty?: Object;
    ngOnInit(): void {
        this.piedata = [
            { x: 'Jan', y: 3, text: 'Jan: 3' }, { x: 'Feb', y: 3.5, text:
'Feb: 3.5' },
            { x: 'Mar', y: null, text: 'Mar: 7' }, { x: 'Apr', y: 13.5,
text: 'Apr: 13.5' },
            { x: 'May', y: 19, text: 'May: 19' }, { x: 'Jun', y: 23.5,
text: 'Jun: 23.5' },
            { x: 'Jul', y: 26, text: 'Jul: 26' }, { x: 'Aug', y:
undefined, text: 'Aug: 25' },
            { x: 'Sep', y: 21, text: 'Sep: 21' }, { x: 'Oct', y: 15,
text: 'Oct: 15' }];
        this.empty = {
            mode: 'Average', fill: 'pink', border: { width: 2, color:
'black' }
        }
    }
}

```

MAIN.TS

```

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

```

Annotation in Angular Accumulation chart component

The annotations are used to mark the specific area of interest in the chart area with texts, shapes or images.

<!-- markdownlint-disable MD033 -->

By using the `<code>content</code>`

 option of annotation property, you can specify the Id of the element that needs to be displayed in the chart area.**APP.COMPONENT.TS**

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { AccumulationChartModule } from '@syncfusion/ej2-angular-charts'

```

```

import { PieSeriesService, AccumulationLegendService,
  AccumulationTooltipService, AccumulationAnnotationService,
  AccumulationDataLabelService } from '@syncfusion/ej2-angular-charts'
import { Component, OnInit } from '@angular/core';
import { pieData } from './datasource';
@Component({
  imports: [
    AccumulationChartModule
  ],
  providers: [PieSeriesService, AccumulationLegendService,
    AccumulationTooltipService, AccumulationDataLabelService,
    AccumulationAnnotationService],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-accumulationchart id="chart-container">
    <e-accumulation-annotations>
      <e-accumulation-annotation content='<div style="border: 1px solid
black; background-color:#f5f5f5; padding: 5px 5px 5px 5px">3.5</div>'
      region='Series' coordinateUnits='Point' x='Feb' y=3.5>
    </e-accumulation-annotation>
    </e-accumulation-annotations>
    <e-accumulation-series-collection>
      <e-accumulation-series [dataSource]='piedata' xName='x'
yName='y'></e-accumulation-series>
    </e-accumulation-series-collection>
  </ejs-accumulationchart>`
})
export class AppComponent implements OnInit {
  public piedata?: Object[];
  ngOnInit(): void {
    this.piedata = pieData;
  }
}

```

MAIN.TS

```

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

```

Note: To use annotation feature in accumulation chart, we need to inject `AccumulationAnnotationService` into the `@NgModule.providers`.

Region

The annotation can be placed with respect to either `Series` or `Chart`.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { AccumulationChartModule } from '@syncfusion/ej2-angular-charts'
import { PieSeriesService, AccumulationLegendService,
  AccumulationTooltipService, AccumulationAnnotationService,
  AccumulationDataLabelService } from '@syncfusion/ej2-angular-charts'

```

```
import { Component, OnInit } from '@angular/core';
import { pieData } from './datasource';
@Component({
  imports: [
    AccumulationChartModule
  ],
  providers: [PieSeriesService, AccumulationLegendService,
    AccumulationTooltipService, AccumulationDataLabelService,
    AccumulationAnnotationService],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-accumulationchart id="chart-container">
    <e-accumulation-annotations>
      <e-accumulation-annotation content='<div style="border: 1px solid
black;background-color:#f5f5f5; padding: 5px 5px 5px 5px">13.5</div>'
        region='Chart' coordinateUnits='Pixel' x=150 y=150>
    </e-accumulation-annotation>
    </e-accumulation-annotations>
    <e-accumulation-series-collection>
      <e-accumulation-series [dataSource]='piedata' xName='x'
yName='y'></e-accumulation-series>
    </e-accumulation-series-collection>
  </ejs-accumulationchart>`
})
export class AppComponent implements OnInit {
  public piedata?: Object[];
  ngOnInit(): void {
    this.piedata = pieData;
  }
}
```

MAIN.TS

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

Co-ordinate Units

Specifies the coordinate units of an annotation either in **Pixel** or **Point**.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { AccumulationChartModule } from '@syncfusion/ej2-angular-charts'
import { PieSeriesService, AccumulationLegendService,
  AccumulationTooltipService, AccumulationAnnotationService,
  AccumulationDataLabelService } from '@syncfusion/ej2-angular-charts'
import { Component, OnInit } from '@angular/core';
import { pieData } from './datasource';
@Component({
  imports: [
    AccumulationChartModule
  ],
```



```

providers: [PieSeriesService, AccumulationLegendService,
AccumulationTooltipService, AccumulationDataLabelService,
  AccumulationAnnotationService],
standalone: true,
selector: 'app-container',
template: `<ejs-accumulationchart id="chart-container">
  <e-accumulation-annotations>
    <e-accumulation-annotation content='<div style="border: 1px solid
black;background-color:#f5f5f5; padding: 5px 5px 5px 5px">Jan : 3</div>'
    region='Series' coordinateUnits='Point' x='Jan' y=3>
  </e-accumulation-annotation>
</e-accumulation-annotations>
  <e-accumulation-series-collection>
    <e-accumulation-series [dataSource]='piedata' xName='x'
yName='y'></e-accumulation-series>
  </e-accumulation-series-collection>
</ejs-accumulationchart>`
})
export class AppComponent implements OnInit {
  public piedata?: Object[];
  ngOnInit(): void {
    this.piedata = pieData;
  }
}

```

MAIN.TS

```

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

```

Alignment

The annotations provides the `verticalAlignment` or `horizontalAlignment` properties.

The `verticalAlignment` property can be customized via `Top`, `Bottom` or `Middle` and the `horizontalAlignment` property can be customized via `Near`, `Far` or `Center`.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { AccumulationChartModule } from '@syncfusion/ej2-angular-charts'
import { PieSeriesService, AccumulationLegendService,
AccumulationTooltipService, AccumulationAnnotationService,
  AccumulationDataLabelService } from '@syncfusion/ej2-angular-charts'
import { Component, OnInit } from '@angular/core';
import { pieData } from './datasource';
@Component({
  imports: [
    AccumulationChartModule
  ],
  providers: [PieSeriesService, AccumulationLegendService,
AccumulationTooltipService, AccumulationDataLabelService,
  AccumulationAnnotationService],

```

```

standalone: true,
  selector: 'app-container',
  template: `<ejs-accumulationchart id="chart-container">
    <e-accumulation-annotations>
      <e-accumulation-annotation content='<div style="border: 1px solid
black;background-color:#f5f5f5; padding: 5px 5px 5px 5px">Jan : 3</div>'
        region='Series' coordinateUnits='Point' x='Jan' y=3
verticalAlignment='Top' horizontalAlignment='Near'>
      </e-accumulation-annotation>
    </e-accumulation-annotations>
    <e-accumulation-series-collection>
      <e-accumulation-series [dataSource]='piedata' xName='x'
yName='y'></e-accumulation-series>
    </e-accumulation-series-collection>
  </ejs-accumulationchart>`
})
export class AppComponent implements OnInit {
  public piedata?: Object[];
  ngOnInit(): void {
    this.piedata = pieData;
  }
}

```

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 Accumulation chart component

Tooltip for the accumulation chart can be enabled by using the [enable](#) property.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { AccumulationChartModule } from '@syncfusion/ej2-angular-charts'
import { PieSeriesService, AccumulationLegendService,
AccumulationTooltipService, AccumulationAnnotationService,
  AccumulationDataLabelService } from '@syncfusion/ej2-angular-charts'
import { Component, OnInit } from '@angular/core';
import { pieData } from './datasource';
@Component({
  imports: [
    AccumulationChartModule
  ],
  providers: [PieSeriesService, AccumulationLegendService,
    AccumulationTooltipService, AccumulationDataLabelService,
    AccumulationAnnotationService],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-accumulationchart id="chart-container"
[tooltip]='tooltip'>
    <e-accumulation-series-collection>

```

```

        <e-accumulation-series [dataSource]='piedata' xName='x'
yName='y'></e-accumulation-series>
    </e-accumulation-series-collection>
</ejs-accumulationchart>`
    })
    export class AppComponent implements OnInit {
        public piedata?: Object[];
        public tooltip?: Object;
        ngOnInit(): void {
            this.piedata = pieData;
            this.tooltip = {
                enable: true
            }
        };
    }
}

```

MAIN.TS

```

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

```

Note: To use tooltip feature, inject the `AccumulationTooltipService` into the `@NgModule.providers`.

Header

We can specify header for the tooltip using `header` property.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { AccumulationChartModule } from '@syncfusion/ej2-angular-charts'
import { PieSeriesService, AccumulationLegendService,
AccumulationTooltipService, AccumulationAnnotationService,
    AccumulationDataLabelService } from '@syncfusion/ej2-angular-charts'
import { Component, OnInit } from '@angular/core';
import { pieData } from './datasource';
@Component({
    imports: [
        AccumulationChartModule
    ],
    providers: [PieSeriesService, AccumulationLegendService,
        AccumulationTooltipService, AccumulationDataLabelService,
        AccumulationAnnotationService],
    standalone: true,
    selector: 'app-container',
    template: `<ejs-accumulationchart id="chart-container"
[tooltip]='tooltip'>
        <e-accumulation-series-collection>
            <e-accumulation-series [dataSource]='piedata' xName='x'
yName='y'></e-accumulation-series>
        </e-accumulation-series-collection>
    </ejs-accumulationchart>`
})
export class AppComponent implements OnInit {

```

```

public piedata?: Object[];
public tooltip?: Object;
ngOnInit(): void {
    this.piedata = pieData;
    this.tooltip = {
        enable: true, header: "Pie Chart"
    };
}

```

MAIN.TS

```

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

```

Format

By default, tooltip shows information of x and y value in points. In addition to that, you can show more information in tooltip. For example the format `${series.name} ${point.x}` shows series name and point x value.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { AccumulationChartModule } from '@syncfusion/ej2-angular-charts'
import { PieSeriesService, AccumulationLegendService,
    AccumulationTooltipService, AccumulationAnnotationService,
    AccumulationDataLabelService } from '@syncfusion/ej2-angular-charts'
import { Component, OnInit } from '@angular/core';
import { pieData } from './datasource';
@Component({
    imports: [
        AccumulationChartModule
    ],
    providers: [PieSeriesService, AccumulationLegendService,
        AccumulationTooltipService, AccumulationDataLabelService,
        AccumulationAnnotationService],
    standalone: true,
    selector: 'app-container',
    template: `<ejs-accumulationchart id="chart-container"
[tooltip]='tooltip'>
    <e-accumulation-series-collection>
        <e-accumulation-series [dataSource]='piedata' xName='x'
yName='y'></e-accumulation-series>
    </e-accumulation-series-collection>
</ejs-accumulationchart>`
})
export class AppComponent implements OnInit {
    public piedata?: Object[];
    public tooltip?: Object;
    ngOnInit(): void {
        this.piedata = pieData;
        this.tooltip = {

```

```

        enable: true, format: '${point.x} : <b>${point.y}%</b>'
      }
    };
  }
}

```

MAIN.TS

```

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

```

Tooltip template

Any HTML element can be displayed in the tooltip by using the [template](#) property.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { AccumulationChartModule } from '@syncfusion/ej2-angular-charts'
import { PieSeriesService, AccumulationLegendService,
  AccumulationTooltipService, AccumulationAnnotationService,
  AccumulationDataLabelService } from '@syncfusion/ej2-angular-charts'
import { Component, OnInit } from '@angular/core';
import { pieData } from './datasource';
@Component({
  imports: [
    AccumulationChartModule
  ],
  providers: [PieSeriesService, AccumulationLegendService,
    AccumulationTooltipService, AccumulationDataLabelService,
    AccumulationAnnotationService],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-accumulationchart id="chart-container"
[tooltip]='tooltip'>
  <e-accumulation-series-collection>
    <e-accumulation-series [dataSource]='piedata' xName='x'
yName='y'></e-accumulation-series>
  </e-accumulation-series-collection>
</ejs-accumulationchart>`
})
export class AppComponent implements OnInit {
  public piedata?: Object[];
  public tooltip?: Object;
  ngOnInit(): void {
    this.piedata = pieData;
    this.tooltip = {
      enable: true, template: "<div id='templateWrap'
style='background-color:#bd18f9;border-radius: 3px; float: right;padding:
2px;line-height: 20px;text-align: center;'>"+
      "<img
src='https://ej2.syncfusion.com/demos/src/chart/images/sunny.png' />" +
      "<div style='color:white; font-family:Roboto; font-style: medium;
font-size:14px;float: right;padding: 2px;line-height: 20px;text-align:
center;padding-right:6px'><span>${y}</span></div></div>"

```

```

    }
  };
}

```

MAIN.TS

```

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

```

Fixed tooltip

By default, tooltip track the mouse movement, but you can set a fixed position for the tooltip by using the [location](#) property.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { AccumulationChartModule } from '@syncfusion/ej2-angular-charts'
import { PieSeriesService, AccumulationLegendService,
AccumulationTooltipService, AccumulationAnnotationService,
  AccumulationDataLabelService } from '@syncfusion/ej2-angular-charts'
import { Component, OnInit } from '@angular/core';
import { pieData } from './datasource';
@Component({
  imports: [
    AccumulationChartModule
  ],
  providers: [PieSeriesService, AccumulationLegendService,
    AccumulationTooltipService, AccumulationDataLabelService,
    AccumulationAnnotationService],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-accumulationchart id="chart-container"
[tooltip]='tooltip'>
  <e-accumulation-series-collection>
    <e-accumulation-series [dataSource]='piedata' xName='x'
yName='y'></e-accumulation-series>
  </e-accumulation-series-collection>
</ejs-accumulationchart>`
})
export class AppComponent implements OnInit {
  public piedata?: Object[];
  public tooltip?: Object;
  ngOnInit(): void {
    this.piedata = pieData;
    this.tooltip = {
      enable: true,
      location: { x: 200, y: 20 }
    }
  };
}

```

MAIN.TS

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

Customization

The [fill](#) and [border](#) properties are used to customize the background color and border of the tooltip respectively. The [textStyle](#) property in the tooltip is used to customize the font of the tooltip text. The [highlightColor](#) property can be used to change the color of the data point when hovering.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { AccumulationChartModule } from '@syncfusion/ej2-angular-charts'
import { PieSeriesService, AccumulationLegendService,
  AccumulationTooltipService, AccumulationAnnotationService,
  AccumulationDataLabelService } from '@syncfusion/ej2-angular-charts'
import { Component, OnInit } from '@angular/core';
import { pieData } from './datasource';
@Component({
  imports: [
    AccumulationChartModule
  ],
  providers: [PieSeriesService, AccumulationLegendService,
    AccumulationTooltipService, AccumulationDataLabelService,
    AccumulationAnnotationService],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-accumulationchart id="chart-container"
[tooltip]='tooltip' highlightColor="red" [legendSettings]='legendSettings'>
  <e-accumulation-series-collection>
    <e-accumulation-series [dataSource]='piedata' xName='x'
yName='y'></e-accumulation-series>
  </e-accumulation-series-collection>
</ejs-accumulationchart>`
})
export class AppComponent implements OnInit {
  public piedata: Object[] = [];
  public tooltip?: Object;
  public legendSettings?: Object;
  ngOnInit(): void {
    this.piedata = pieData;
    this.tooltip = {
      enable: true };
    this.legendSettings = { visible: false}
  };
}
```

MAIN.TS

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

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

Tooltip mapping name

By default, tooltip shows information of x and y value in points. You can show more information from datasource in tooltip by using the [tooltipMappingName](#) property of the tooltip.

You can use the ``${point.tooltip}`` as place holders to display the specified tooltip content.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { AccumulationChartModule } from '@syncfusion/ej2-angular-charts'
import { PieSeriesService, AccumulationLegendService,
  AccumulationTooltipService, AccumulationAnnotationService,
  AccumulationDataLabelService } from '@syncfusion/ej2-angular-charts'
import { Component, OnInit } from '@angular/core';
@Component({
  imports: [
    AccumulationChartModule
  ],
  providers: [PieSeriesService, AccumulationLegendService,
    AccumulationTooltipService, AccumulationDataLabelService,
    AccumulationAnnotationService],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-accumulationchart id="chart-container"
[tooltip]='tooltip'>
  <e-accumulation-series-collection>
    <e-accumulation-series [dataSource]='piedata' xName='x' yName='y'
tooltipMappingName='text'></e-accumulation-series>
  </e-accumulation-series-collection>
</ejs-accumulationchart>`
})
export class AppComponent implements OnInit {
  public piedata?: Object[];
  public tooltip?: Object;
  ngOnInit(): void {
    this.piedata = [{ x: 'Jan', y: 13, text: 'Jan: 13' }, { x: 'Feb', y:
13, text: 'Feb: 13' },
    { x: 'Mar', y: 17, text: 'Mar: 17' }, { x: 'Apr', y:
13.5, text: 'Apr: 13.5' }
  ];
  this.tooltip = {
    enable: true, format: `${point.tooltip}`
  };
}
```

MAIN.TS

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


To customize individual tooltip

Using [tooltipRender](#) event, you can customize a tooltip for particular point. event, you can customize a tooltip for particular point.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { AccumulationChartModule } from '@syncfusion/ej2-angular-charts'
import { PieSeriesService, AccumulationLegendService,
  AccumulationTooltipService, AccumulationAnnotationService,
  AccumulationDataLabelService } from '@syncfusion/ej2-angular-charts'
import { Component, OnInit } from '@angular/core';
import { pieData } from './datasource';
import { IAccTooltipRenderEventArgs } from '@syncfusion/ej2-charts';
import { FontModel } from '@syncfusion/ej2-angular-charts';
@Component({
  imports: [
    AccumulationChartModule
  ],
  providers: [PieSeriesService, AccumulationLegendService,
    AccumulationTooltipService, AccumulationDataLabelService,
    AccumulationAnnotationService],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-accumulationchart id="chart-container"
[tooltip]='tooltip' (tooltipRender)="ontooltipRender($event)">
  <e-accumulation-series-collection>
    <e-accumulation-series [dataSource]='piedata' xName='x'
yName='y'></e-accumulation-series>
  </e-accumulation-series-collection>
</ejs-accumulationchart>`
})
export class AppComponent implements OnInit {
  public piedata?: Object[];
  public tooltip?: Object;
  public ontooltipRender?: Function;
  ngOnInit(): void {
    this.piedata = pieData;
    this.tooltip = {
      enable: true
    }
    this.ontooltipRender = (args: IAccTooltipRenderEventArgs) => {
      if (args.point.index === 3) {
        args.text = args.point.x + ' ' + ':' + args.point.y + ' ' + ' '
+ 'customtext';
        (args.textStyle as FontModel).color = '#f48042';
      }
    }
  };
}
```

MAIN.TS

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

Legend in Angular Accumulation chart component

As like a chart, the legend is also available for accumulation charts, which gives information about the points.

By default, the legend will be placed on the right, if the width of the chart is high or will be placed on the bottom, if the height of the chart is high. Other customization features regarding the legend element are same as the [chart legend](#).

Here, the legend for a point can be collapsed by giving the empty string to the x value of the point.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { AccumulationChartModule } from '@syncfusion/ej2-angular-charts'
import { PieSeriesService, AccumulationLegendService } from '@syncfusion/ej2-angular-charts'
import { Component, OnInit } from '@angular/core';
import { pieData } from './datasource';
@Component({
  imports: [
    AccumulationChartModule
  ],
  providers: [PieSeriesService, AccumulationLegendService],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-accumulationchart id="chart-container"
[legendSettings]='legendSettings'>
  <e-accumulation-series-collection>
    <e-accumulation-series [dataSource]='piedata' xName='x'
yName='y'></e-accumulation-series>
  </e-accumulation-series-collection>
</ejs-accumulationchart>`
})
export class AppComponent implements OnInit {
  public piedata?: Object[];
  public legendSettings?: Object;
  ngOnInit(): void {
    this.legendSettings = {
      visible: true
    };
    this.piedata = pieData;
  }
}
```

MAIN.TS

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

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

Position and Alignment

By using the position property, you can position the legend at the **left**, **right**, **top** or **bottom** of the chart.

You can also align the legend to **center**, **far** or **near** of the chart using the alignment property.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { AccumulationChartModule } from '@syncfusion/ej2-angular-charts'
import { PieSeriesService, AccumulationLegendService,
  AccumulationTooltipService, AccumulationAnnotationService,
  AccumulationDataLabelService } from '@syncfusion/ej2-angular-charts'
import { Component, OnInit } from '@angular/core';
import { pieData } from './datasource';
@Component({
  imports: [
    AccumulationChartModule
  ],
  providers: [PieSeriesService, AccumulationLegendService,
    AccumulationTooltipService, AccumulationDataLabelService,
    AccumulationAnnotationService],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-accumulationchart id="chart-container"
[legendSettings]='legendSettings'>
  <e-accumulation-series-collection>
    <e-accumulation-series [dataSource]='piedata' xName='x'
yName='y'></e-accumulation-series>
  </e-accumulation-series-collection>
</ejs-accumulationchart>`
})
export class AppComponent implements OnInit {
  public piedata?: Object[];
  public legendSettings?: Object;
  ngOnInit(): void {
    this.legendSettings = {
      position: 'Top', alignment: 'Near'
    };
    this.piedata = pieData;
  }
}
```

MAIN.TS

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

Legend Reverse

You can reverse the order of the legend items by using the [reverse](#) property. By default, legend for the first series in the collection will be placed first.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { AccumulationChartModule } from '@syncfusion/ej2-angular-charts'
import { PieSeriesService, AccumulationLegendService,
  AccumulationTooltipService, AccumulationAnnotationService,
  AccumulationDataLabelService } from '@syncfusion/ej2-angular-charts'
import { Component, OnInit } from '@angular/core';
import { pieData } from './datasource';
@Component({
  imports: [
    AccumulationChartModule
  ],
  providers: [PieSeriesService, AccumulationLegendService,
    AccumulationTooltipService, AccumulationDataLabelService,
    AccumulationAnnotationService],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-accumulationchart id="chart-container"
[legendSettings]='legendSettings'>
  <e-accumulation-series-collection>
    <e-accumulation-series [dataSource]='piedata' xName='x' yName='y'
legendShape='Rectangle'></e-accumulation-series>
  </e-accumulation-series-collection>
</ejs-accumulationchart>`
})
export class AppComponent implements OnInit {
  public piedata?: Object[];
  public legendSettings?: Object;
  ngOnInit(): void {
    this.legendSettings = {
      visible: true,
      reverse: true
    };
    this.piedata = pieData;
  }
}
```

MAIN.TS

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

Legend Shape

To change the legend icon shape, use the `legendShape` property in the `series`. By default, legend icon shape is `seriesType`.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { AccumulationChartModule } from '@syncfusion/ej2-angular-charts'
import { PieSeriesService, AccumulationLegendService,
  AccumulationTooltipService, AccumulationAnnotationService,
  AccumulationDataLabelService } from '@syncfusion/ej2-angular-charts'
import { Component, OnInit } from '@angular/core';
import { pieData } from './datasource';
@Component({
  imports: [
    AccumulationChartModule
  ],
  providers: [PieSeriesService, AccumulationLegendService,
    AccumulationTooltipService, AccumulationDataLabelService,
    AccumulationAnnotationService],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-accumulationchart id="chart-container"
[legendSettings]='legendSettings'>
  <e-accumulation-series-collection>
    <e-accumulation-series [dataSource]='piedata' xName='x' yName='y'
legendShape='Rectangle'></e-accumulation-series>
  </e-accumulation-series-collection>
</ejs-accumulationchart>`
})
export class AppComponent implements OnInit {
  public piedata?: Object[];
  public legendSettings?: Object;
  ngOnInit(): void {
    this.legendSettings = {
      visible: true
    };
    this.piedata = pieData;
  }
}

```

MAIN.TS

```

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

```

Legend Size

The legend size can be changed by using the **width** and **height** properties of the **legendSettings**.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { AccumulationChartModule } from '@syncfusion/ej2-angular-charts'
import { PieSeriesService, AccumulationLegendService,
  AccumulationTooltipService, AccumulationAnnotationService,
  AccumulationDataLabelService } from '@syncfusion/ej2-angular-charts'
import { Component, OnInit } from '@angular/core';

```

```

import { pieData } from './datasource';
@Component({
  imports: [
    AccumulationChartModule
  ],
  providers: [PieSeriesService, AccumulationLegendService,
    AccumulationTooltipService, AccumulationDataLabelService,
    AccumulationAnnotationService],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-accumulationchart id="chart-container"
[legendSettings]='legendSettings'>
  <e-accumulation-series-collection>
    <e-accumulation-series [dataSource]='piedata' xName='x' yName='y'
legendShape='Circle'></e-accumulation-series>
  </e-accumulation-series-collection>
</ejs-accumulationchart>`
})
export class AppComponent implements OnInit {
  public piedata?: Object[];
  public legendSettings?: Object;
  ngOnInit(): void {
    this.legendSettings = {
      width: '150', height: '100', border: { width: 1, color: 'pink' }
    };
    this.piedata = pieData;
  }
}

```

MAIN.TS

```

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

```

Legend Item Size

You can customize the size of the legend items by using the `shapeHeight` and `shapeWidth` properties.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { AccumulationChartModule } from '@syncfusion/ej2-angular-charts'
import { PieSeriesService, AccumulationLegendService,
  AccumulationTooltipService, AccumulationAnnotationService,
  AccumulationDataLabelService } from '@syncfusion/ej2-angular-charts'
import { Component, OnInit } from '@angular/core';
import { pieData } from './datasource';
@Component({
  imports: [
    AccumulationChartModule
  ],
  providers: [PieSeriesService, AccumulationLegendService,
    AccumulationTooltipService, AccumulationDataLabelService,

```

```

    AccumulationAnnotationService],
    standalone: true,
    selector: 'app-container',
    template: `<ejs-accumulationchart id="chart-container"
[legendSettings]='legendSettings'>
        <e-accumulation-series-collection>
            <e-accumulation-series [dataSource]='piedata' xName='x' yName='y'
legendShape='Circle'></e-accumulation-series>
        </e-accumulation-series-collection>
    </ejs-accumulationchart>`
  })
  export class AppComponent implements OnInit {
    public piedata?: Object[];
    public legendSettings?: Object;
    ngOnInit(): void {
      this.legendSettings = {
        shapeHeight: 15, shapeWidth: 15
      };
      this.piedata = pieData;
    }
  }
}

```

MAIN.TS

```

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

```

Enable Animation

You can customize the animation while clicking legend by setting enableAnimation as true or false using enableAnimation property in Accumulation Chart.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { AccumulationChartModule } from '@syncfusion/ej2-angular-charts'
import { PieSeriesService, AccumulationLegendService,
AccumulationTooltipService, AccumulationAnnotationService,
    AccumulationDataLabelService } from '@syncfusion/ej2-angular-charts'
import { Component, OnInit } from '@angular/core';
import { pieData } from './datasource';
@Component({
  imports: [
    AccumulationChartModule
  ],
  providers: [PieSeriesService, AccumulationLegendService,
AccumulationTooltipService, AccumulationDataLabelService,
    AccumulationAnnotationService],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-accumulationchart id="chart-container"
[legendSettings]='legendSettings' [enableAnimation]='enableAnimation'
[enableSmartLabels]='enableSmartLabels'

```

```

    [title]='title'>
      <e-accumulation-series-collection>
        <e-accumulation-series [dataSource]='piedata' xName='x' yName='y'
groupTo='11' [dataLabel]='datalabel' [explode]='explode'
[explodeOffset]='explodeOffset'
          [startAngle]='startAngle' [endAngle]='endAngle' radius='70%'
innerRadius='40%'></e-accumulation-series>
        </e-accumulation-series-collection>
      </ejs-accumulationchart>`
  })
  export class AppComponent implements OnInit {
    public piedata?: Object[];
    public legendSettings?: Object;
    public datalabel?: Object;
    public startAngle?: number;
    public endAngle?: number;
    public explode?: boolean;
    public explodeOffset?: string;
    public enableSmartLabels?: boolean;
    public enableAnimation?: boolean;
    public title: Object = 'Project Cost Breakdown';
    ngOnInit(): void {
      this.datalabel = { visible: true,
        name: 'text',
        position: 'Inside',
        font: {
          fontWeight: '600',
          color: '#ffffff'
        }
      };
      this.legendSettings = {
        visible: true, position: 'Top'
      };
      this.startAngle = 0;
      this.endAngle = 360;
      this.explode = true;
      this.explodeOffset = '10%';
      this.enableSmartLabels = true;
      this.enableAnimation = true;
      this.piedata = [{ x: 'Labour', y: 18, text: '18%' }, { x: 'Legal', y:
8, text: '8%' },
        { x: 'Production', y: 15, text: '15%' }, { x: 'License', y:
11, text: '11%' },
        { x: 'Facilities', y: 18, text: '18%' }, { x: 'Taxes', y: 14,
text: '14%' },
        { x: 'Insurance', y: 16, text: '16%' }]];
    }
  }

```

MAIN.TS

```

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

```


Paging for Legend

Paging will be enabled by default, when the legend items exceeds the legend bounds. You can view the each legend item by navigating between the pages using the navigation buttons.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { AccumulationChartModule } from '@syncfusion/ej2-angular-charts'
import { PieSeriesService, AccumulationLegendService,
  AccumulationTooltipService, AccumulationAnnotationService,
  AccumulationDataLabelService } from '@syncfusion/ej2-angular-charts'
import { Component, OnInit } from '@angular/core';
import { pieData } from './datasource';
@Component({
  imports: [
    AccumulationChartModule
  ],
  providers: [PieSeriesService, AccumulationLegendService,
    AccumulationTooltipService, AccumulationDataLabelService,
    AccumulationAnnotationService],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-accumulationchart id="chart-container"
[legendSettings]='legendSettings'>
  <e-accumulation-series-collection>
    <e-accumulation-series [dataSource]='piedata' xName='x' yName='y'
legendShape='Circle'></e-accumulation-series>
  </e-accumulation-series-collection>
</ejs-accumulationchart>`
})
export class AppComponent implements OnInit {
  public piedata?: Object[];
  public legendSettings?: Object;
  ngOnInit(): void {
    this.legendSettings = {
      height: '150', width: '80'
    };
    this.piedata = pieData;
  }
}
```

MAIN.TS

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

Legend Text Wrap

When the legend text exceeds the container, the text can be wrapped by using `textWrap` Property. End user can also wrap the legend text based on the `maximumLabelWidth` property.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { AccumulationChartModule } from '@syncfusion/ej2-angular-charts'
import { PieSeriesService, AccumulationLegendService,
  AccumulationTooltipService, AccumulationAnnotationService,
  AccumulationDataLabelService } from '@syncfusion/ej2-angular-charts'
import { Component, OnInit } from '@angular/core';
import { piechart } from './datasource';
@Component({
  imports: [
    AccumulationChartModule
  ],
  providers: [PieSeriesService, AccumulationLegendService,
    AccumulationTooltipService, AccumulationDataLabelService,
    AccumulationAnnotationService],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-accumulationchart id="chart-container"
[legendSettings]='legendSettings'>
  <e-accumulation-series-collection>
    <e-accumulation-series [dataSource]='piechart' xName='x'
yName='y' legendShape='Circle'></e-accumulation-series>
  </e-accumulation-series-collection>
</ejs-accumulationchart>`
})
export class AppComponent implements OnInit {
  public piechart?: Object[];
  public legendSettings?: Object;
  ngOnInit(): void {
    this.legendSettings = {
      visible:true, position:'Right', textWrap:'Wrap',
      maximumLabelWidth:60, height:'44%', width:'64%'
    };
    this.piechart = piechart;
  }
}

```

MAIN.TS

```

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

```

Legend Title

You can set title for legend using title property in legendSettings. You can also customize the fontStyle, size, fontWeight, color, textAlignment, fontFamily, opacity and textOverflow of legend title. titlePosition is used to set the legend position in Top, Left and Right position. maximumTitleWidth is used to set the width of the legend title. By default, it will be 100px.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { AccumulationChartModule } from '@syncfusion/ej2-angular-charts'

```

```

import { PieSeriesService, AccumulationLegendService,
  AccumulationTooltipService, AccumulationAnnotationService,
  AccumulationDataLabelService } from '@syncfusion/ej2-angular-charts'
import { Component, OnInit } from '@angular/core';
import { pieData } from './datasource';
@Component({
  imports: [
    AccumulationChartModule
  ],
  providers: [PieSeriesService, AccumulationLegendService,
    AccumulationTooltipService, AccumulationDataLabelService,
    AccumulationAnnotationService],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-accumulationchart id="chart-container"
[legendSettings]='legendSettings'>
  <e-accumulation-series-collection>
    <e-accumulation-series [dataSource]='piedata' xName='x' yName='y'
legendShape='Circle'></e-accumulation-series>
  </e-accumulation-series-collection>
</ejs-accumulationchart>`
})
export class AppComponent implements OnInit {
  public piedata?: Object[];
  public legendSettings?: Object;
  ngOnInit(): void {
    this.legendSettings = {
      title: 'Months', position: 'Bottom'
    };
    this.piedata = pieData;
  }
}

```

MAIN.TS

```

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

```

Arrow Page Navigation

By default, the page number will be enabled while legend paging. Now, you can disable that page number and also you can get left and right arrows for page navigation. You have to set `false` value to `enablePages` to get this support.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { AccumulationChartModule } from '@syncfusion/ej2-angular-charts'
import { PieSeriesService, AccumulationLegendService,
  AccumulationTooltipService, AccumulationAnnotationService,
  AccumulationDataLabelService } from '@syncfusion/ej2-angular-charts'
import { Component, OnInit } from '@angular/core';
import { pieData } from './datasource';

```

```

@Component({
  imports: [
    AccumulationChartModule
  ],
  providers: [PieSeriesService, AccumulationLegendService,
    AccumulationTooltipService, AccumulationDataLabelService,
    AccumulationAnnotationService],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-accumulationchart id="chart-container"
[legendSettings]='legendSettings'>
    <e-accumulation-series-collection>
      <e-accumulation-series [dataSource]='piedata' xName='x' yName='y'
legendShape='Circle'></e-accumulation-series>
    </e-accumulation-series-collection>
  </ejs-accumulationchart>`
})
export class AppComponent implements OnInit {
  public piedata?: Object[];
  public legendSettings?: Object;
  ngOnInit(): void {
    this.legendSettings = {
      width: '260px', height: '50px', enablePages: false, position:
'Bottom'
    };
    this.piedata = pieData;
  }
}

```

MAIN.TS

```

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

```

Legend Item Padding

The [itemPadding](#) property can be used to adjust the space between the legend items.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { AccumulationChartModule } from '@syncfusion/ej2-angular-charts'
import { PieSeriesService, AccumulationLegendService,
  AccumulationTooltipService, AccumulationAnnotationService,
  AccumulationDataLabelService } from '@syncfusion/ej2-angular-charts'
import { Component, OnInit } from '@angular/core';
import { pieData } from './datasource';
@Component({
  imports: [
    AccumulationChartModule
  ],
  providers: [PieSeriesService, AccumulationLegendService,
    AccumulationTooltipService, AccumulationDataLabelService,
    AccumulationAnnotationService],

```

```

standalone: true,
  selector: 'app-container',
  template: `<ejs-accumulationchart id="chart-container"
[legendSettings]='legendSettings'>
    <e-accumulation-series-collection>
      <e-accumulation-series [dataSource]='piedata' xName='x' yName='y'
legendShape='Circle'></e-accumulation-series>
    </e-accumulation-series-collection>
  </ejs-accumulationchart>`
})
export class AppComponent implements OnInit {
  public piedata?: Object[];
  public legendSettings?: Object;
  ngOnInit(): void {
    this.legendSettings = {
      width: '260px', height: '50px', position: "Bottom", itemPadding: 30
    };
    this.piedata = pieData;
  }
}

```

MAIN.TS

```

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

```

Centerlabel

Using [centerLabel](#) it is now possible to place a label at the center of a pie or donut chart. To configure the default text rendered on the center label for the pie and doughnut charts, use the [text](#) property in the [centerLabel](#).

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { AccumulationChartModule } from '@syncfusion/ej2-angular-charts'
import { PieSeriesService, AccumulationLegendService,
AccumulationTooltipService, AccumulationAnnotationService,
  AccumulationDataLabelService } from '@syncfusion/ej2-angular-charts'
import { Component, OnInit } from '@angular/core';
@Component({
  imports: [
    AccumulationChartModule
  ],
  providers: [PieSeriesService, AccumulationLegendService,
AccumulationTooltipService, AccumulationDataLabelService,
  AccumulationAnnotationService],
  standalone: true,
  selector: 'app-container',
  template:
    `<ejs-accumulationchart id="chart-container"
[legendSettings]='legendSettings' [centerLabel]='centerLabel'>
    <e-accumulation-series-collection>

```

```

        <e-accumulation-series [dataSource]='centerLabelData' xName='x'
yName='y' innerRadius='65%'></e-accumulation-series>
    </e-accumulation-series-collection>
</ejs-accumulationchart>`
    })
    export class AppComponent implements OnInit {
        public piedata?: Object[];
        public legendSettings?: Object;
        public centerLabel?: Object;
        public centerLabelData: Object = [
            { x: 'Chrome', y: 61.3, text: 'Chrome: 61.3%' },
            { x: 'Safari', y: 24.6, text: 'Safari: 24.6%' },
            { x: 'Edge', y: 5.0, text: 'Edge: 5.00%' },
            { x: 'Samsung Internet', y: 2.7, text: 'Samsung Internet: 2.7%' },
            { x: 'Firefox', y: 2.6, text: 'Firefox: 2.6%' },
            { x: 'Others', y: 3.6, text: 'Others: 3.6%' }
        ];
        ngOnInit(): void {
            this.legendSettings = {
                visible: false
            };
            this.centerLabel = {
                text: 'Mobile<br>Browsers<br>Statistics'
            }
        }
    }
}

```

MAIN.TS

```

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

```

Hover text

The default text in the center label can be changed when the mouse pointer hovers over the pie and doughnut charts slice using the [hoverTextFormat](#) property.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { AccumulationChartModule } from '@syncfusion/ej2-angular-charts'
import { PieSeriesService, AccumulationLegendService,
AccumulationTooltipService, AccumulationAnnotationService,
    AccumulationDataLabelService } from '@syncfusion/ej2-angular-charts'
import { Component, OnInit } from '@angular/core';
@Component({
    imports: [
        AccumulationChartModule
    ],
    providers: [PieSeriesService, AccumulationLegendService,
        AccumulationTooltipService, AccumulationDataLabelService,
            AccumulationAnnotationService],
    standalone: true,

```

```

    selector: 'app-container',
    template:
      `<ejs-accumulationchart id="chart-container"
[legendSettings]='legendSettings' [centerLabel]='centerLabel'>
      <e-accumulation-series-collection>
        <e-accumulation-series [dataSource]='centerLabelData' xName='x'
yName='y' innerRadius='65%'></e-accumulation-series>
      </e-accumulation-series-collection>
    </ejs-accumulationchart>`
  ))
export class AppComponent implements OnInit {
  public piedata?: Object[];
  public legendSettings?: Object;
  public centerLabel?: Object;
  public centerLabelData: Object = [
    { x: 'Chrome', y: 61.3, text: 'Chrome: 61.3%' },
    { x: 'Safari', y: 24.6, text: 'Safari: 24.6%' },
    { x: 'Edge', y: 5.0, text: 'Edge: 5.00%' },
    { x: 'Samsung Internet', y: 2.7, text: 'Samsung Internet: 2.7%' },
    { x: 'Firefox', y: 2.6, text: 'Firefox: 2.6%' },
    { x: 'Others', y: 3.6, text: 'Others: 3.6%' }
  ];
  ngOnInit(): void {
    this.legendSettings = {
      visible: false
    };
    this.centerLabel = {
      text: 'Mobile<br>Browsers<br>Statistics',
      hoverTextFormat: `${point.x} <br> Browser Share <br> ${point.y}%`
    }
  }
}

```

MAIN.TS

```

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

```

Customization

Customize the center label text using the [textStyle](#) property.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { AccumulationChartModule } from '@syncfusion/ej2-angular-charts'
import { PieSeriesService, AccumulationLegendService,
AccumulationTooltipService, AccumulationAnnotationService,
  AccumulationDataLabelService } from '@syncfusion/ej2-angular-charts'
import { Component, OnInit } from '@angular/core';
@Component({
  imports: [
    AccumulationChartModule
  ],

```

```

providers: [PieSeriesService, AccumulationLegendService,
AccumulationTooltipService, AccumulationDataLabelService,
  AccumulationAnnotationService],
standalone: true,
  selector: 'app-container',
  template: `<ejs-accumulationchart id="chart-container"
[legendSettings]='legendSettings' [centerLabel]='centerLabel'>
  <e-accumulation-series-collection>
    <e-accumulation-series [dataSource]='centerLabelData' xName='x'
yName='y' innerRadius='65%' ></e-accumulation-series>
  </e-accumulation-series-collection>
</ejs-accumulationchart>`
})
export class AppComponent implements OnInit {
  public piedata?: Object[];
  public legendSettings?: Object;
  public centerLabel?: Object;
  public centerLabelData: Object= [
    { x: 'Chrome', y: 61.3, text: 'Chrome: 61.3%' },
    { x: 'Safari', y: 24.6, text: 'Safari: 24.6%' },
    { x: 'Edge', y: 5.0, text: 'Edge: 5.00%' },
    { x: 'Samsung Internet', y: 2.7, text: 'Samsung Internet: 2.7%' },
    { x: 'Firefox', y: 2.6, text: 'Firefox: 2.6%' },
    { x: 'Others', y: 3.6, text: 'Others: 3.6%' }
  ];
  ngOnInit(): void {
    this.legendSettings = {
      visible: false
    };
    this.centerLabel = {
      text : 'Mobile<br>Browsers<br>Statistics',
      hoverTextFormat: '${point.x} <br> Browser Share <br>
${point.y}%',
      textStyle:{
        fontWeight: '900',
        size: '15px',
        color: 'grey',
        fontFamily: 'Roboto',
        fontStyle: 'Italic'
      }
    }
  }
}

```

MAIN.TS

```

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

```

Title and sub title in Angular Accumulation chart component

Accumulation Chart can be given a title using [title](#) property, to show the information about the data plotted.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { AccumulationChartModule } from '@syncfusion/ej2-angular-charts'
import { PieSeriesService, AccumulationLegendService,
  AccumulationTooltipService, AccumulationAnnotationService,
  AccumulationDataLabelService } from '@syncfusion/ej2-angular-charts'
import { Component, OnInit } from '@angular/core';
@Component({
  imports: [
    AccumulationChartModule
  ],
  providers: [PieSeriesService, AccumulationLegendService,
    AccumulationTooltipService, AccumulationDataLabelService,
    AccumulationAnnotationService],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-accumulationchart id="chart-container"
[legendSettings]='legendSettings' [title]='title'>
  <e-accumulation-series-collection>
    <e-accumulation-series [dataSource]='piedata' xName='x' yName='y'
type='Pie'></e-accumulation-series>
  </e-accumulation-series-collection>
</ejs-accumulationchart>`
})
export class AppComponent implements OnInit {
  public piedata?: Object[];
  public legendSettings?: Object;
  public title?: string;
  public titleStyle?: Object;
  ngOnInit(): void {
    this.piedata = [{ x: 'Saudi Arabia', y: 58, text: '58%' },
      { x: 'Persian Gulf', y: 15, text: '15%' },
      { x: 'Canada', y: 13, text: '13%' },
      { x: 'Venezuela', y: 8, text: '8%' },
      { x: 'Mexico', y: 3, text: '3%' },
      { x: 'Russia', y: 2, text: '2%' },
      { x: 'Miscellaneous', y: 1, text: '1%' }];
    this.legendSettings = {
      visible: false
    };
    this.title = 'Oil and other liquid imports in USA';
  }
}

```

MAIN.TS

```

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

```

Title Customization

Accumulation Chart can be customizing a title using [titleStyle](#) property.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { AccumulationChartModule } from '@syncfusion/ej2-angular-charts'
import { PieSeriesService, AccumulationLegendService,
  AccumulationTooltipService, AccumulationAnnotationService,
  AccumulationDataLabelService } from '@syncfusion/ej2-angular-charts'
import { Component, OnInit } from '@angular/core';
import { pieData } from './datasource';
@Component({
  imports: [
    AccumulationChartModule
  ],
  providers: [PieSeriesService, AccumulationLegendService,
    AccumulationTooltipService, AccumulationDataLabelService,
    AccumulationAnnotationService],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-accumulationchart id="chart-container"
[legendSettings]='legendSettings' [title]='title' [titleStyle]='titleStyle'>
  <e-accumulation-series-collection>
    <e-accumulation-series [dataSource]='piedata' xName='x' yName='y'
type='Pie'></e-accumulation-series>
  </e-accumulation-series-collection>
</ejs-accumulationchart>`
})
export class AppComponent implements OnInit {
  public piedata?: Object[];
  public legendSettings?: Object;
  public title?: string;
  public titleStyle?: Object;
  ngOnInit(): void {
    this.piedata = [{ x: 'Saudi Arabia', y: 58, text: '58%' },
      { x: 'Persian Gulf', y: 15, text: '15%' },
      { x: 'Canada', y: 13, text: '13%' },
      { x: 'Venezuela', y: 8, text: '8%' },
      { x: 'Mexico', y: 3, text: '3%' },
      { x: 'Russia', y: 2, text: '2%' },
      { x: 'Miscellaneous', y: 1, text: '1%' }];
    this.legendSettings = {
      visible: false
    };
    this.title = 'Oil and other liquid imports in USA';
    this.titleStyle = {
      fontFamily: 'Arial',
      fontStyle: 'italic',
      fontWeight: 'regular',
      color: '#E27F2D',
      size: '23px'
    }
  }
}

```

MAIN.TS

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

SubTitle

Accumulation Chart can be given a subtitle using [subTitle](#) property, to show the information about the data plotted.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { AccumulationChartModule } from '@syncfusion/ej2-angular-charts'
import { PieSeriesService, AccumulationLegendService,
AccumulationTooltipService, AccumulationAnnotationService,
AccumulationDataLabelService } from '@syncfusion/ej2-angular-charts'
import { Component, OnInit } from '@angular/core';
import { pieData } from './datasource';
@Component({
imports: [
AccumulationChartModule
],
providers: [PieSeriesService, AccumulationLegendService,
AccumulationTooltipService, AccumulationDataLabelService,
AccumulationAnnotationService],
standalone: true,
selector: 'app-container',
template: `<ejs-accumulationchart id="chart-container"
[legendSettings]='legendSettings' [title]='title' [subTitle]='subTitle'>
<e-accumulation-series-collection>
<e-accumulation-series [dataSource]='piedata' xName='x' yName='y'
type='Pie'></e-accumulation-series>
</e-accumulation-series-collection>
</ejs-accumulationchart>`
})
export class AppComponent implements OnInit {
public piedata?: Object[];
public legendSettings?: Object;
public title?: string;
public SubTitle?: string;
public subTitleStyle?: Object;
subTitle?: string;
ngOnInit(): void {
this.piedata = [{ x: 'Saudi Arabia', y: 58, text: '58%' },
{ x: 'Persian Gulf', y: 15, text: '15%' },
{ x: 'Canada', y: 13, text: '13%' },
{ x: 'Venezuela', y: 8, text: '8%' },
{ x: 'Mexico', y: 3, text: '3%' },
{ x: 'Russia', y: 2, text: '2%' },
{ x: 'Miscellaneous', y: 1, text: '1%' }];
this.legendSettings = {
visible: false
};
this.title = 'Oil and other liquid imports in USA';
```

```

        this.subTitle = 'In the year 2014 - 2015';
    }
}

```

MAIN.TS

```

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

```

SubTitle Customization

Accumulation Chart can be customizing a subtitle using [subTitleStyle](#) property.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { AccumulationChartModule } from '@syncfusion/ej2-angular-charts';
import { PieSeriesService, AccumulationLegendService,
    AccumulationTooltipService, AccumulationAnnotationService,
    AccumulationDataLabelService } from '@syncfusion/ej2-angular-charts';
import { Component, OnInit } from '@angular/core';
import { pieData } from './datasource';
@Component({
    imports: [
        AccumulationChartModule
    ],
    providers: [PieSeriesService, AccumulationLegendService,
        AccumulationTooltipService, AccumulationDataLabelService,
        AccumulationAnnotationService],
    standalone: true,
    selector: 'app-container',
    template: `<ejs-accumulationchart id="chart-container"
[legendSettings]='legendSettings' [title]='title' [subTitle]='subTitle'
[subTitleStyle]='subTitleStyle'>
        <e-accumulation-series-collection>
            <e-accumulation-series [dataSource]='piedata' xName='x' yName='y'
type='Pie'></e-accumulation-series>
        </e-accumulation-series-collection>
    </ejs-accumulationchart>`
})
export class AppComponent implements OnInit {
    public piedata?: Object[];
    public legendSettings?: Object;
    public title?: string;
    public SubTitle?: string;
    public subTitleStyle?: Object;
    subTitle?: string;
    ngOnInit(): void {
        this.piedata = [{ x: 'Saudi Arabia', y: 58, text: '58%' },
            { x: 'Persian Gulf', y: 15, text: '15%' },
            { x: 'Canada', y: 13, text: '13%' },
            { x: 'Venezuela', y: 8, text: '8%' },
            { x: 'Mexico', y: 3, text: '3%' },
            { x: 'Russia', y: 2, text: '2%' },

```

```

        { x: 'Miscellaneous', y: 1, text: '1%' }]];
    this.legendSettings = {
        visible: false
    };
    this.title = 'Oil and other liquid imports in USA';
    this.subTitle = 'In the year 2014 - 2015';
    this.subTitleStyle = {
        fontFamily: "Arial",
        fontStyle: 'italic',
        fontWeight: 'regular',
        color: "#E27F2D"
    }
}
}
}

```

MAIN.TS

```

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

```

Chart print in Angular Accumulation chart component

Print

The rendered chart can be printed directly from the browser by calling the public method print.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { AccumulationChartModule, ExportService } from '@syncfusion/ej2-angular-charts'
import { PieSeriesService, AccumulationDataLabelService } from '@syncfusion/ej2-angular-charts'
import { Component, OnInit, ViewChild } from '@angular/core';
import { ChartComponent } from '@syncfusion/ej2-angular-charts';
@Component({
    imports: [
        AccumulationChartModule
    ],
    providers: [PieSeriesService, AccumulationDataLabelService, ExportService],
    standalone: true,
    selector: 'app-container',
    template: `<div class="col-md-8">
        <button ej-button id='print' (click)='print()'>Print</button>
        <ejs-accumulationchart #chart id="chart-container">
            <e-accumulation-series-collection>
                <e-accumulation-series [dataSource]='piedata' xName='x' yName='y'
radius="100%"></e-accumulation-series>
            </e-accumulation-series-collection>
        </ejs-accumulationchart>
    </div> `
})
export class AppComponent implements OnInit {
    public piedata?: Object[];
}

```

```

    @ViewChild('chart')
    public chartObj?: ChartComponent;
    ngOnInit(): void {
        this.piedata = [
            { x: 'Jan', y: 3, text: 'Jan: 3' }, { x: 'Feb', y: 3.5, text:
'Feb: 3.5' },
            { x: 'Mar', y: 7, text: 'Mar: 7' }, { x: 'Apr', y: 13.5,
text: 'Apr: 13.5' },
            { x: 'May', y: 19, text: 'May: 19' }, { x: 'Jun', y: 23.5,
text: 'Jun: 23.5' },
            { x: 'Jul', y: 26, text: 'Jul: 26' }, { x: 'Aug', y: 25,
text: 'Aug: 25' },
            { x: 'Sep', y: 21, text: 'Sep: 21' }, { x: 'Oct', y: 15,
text: 'Oct: 15' }];
        }
        print() {
            this.chartObj?.print();
        }
    }
}

```

MAIN.TS

```

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

```

Export

The rendered chart can be exported to **Image**(jpeg or png) or **SVG** or **PDF** format by using the export method.

Input parameters for this method are **Export Type** for **format** and **fileName** of result.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { AccumulationChartModule, ExportService } from '@syncfusion/ej2-
angular-charts'
import { PieSeriesService, AccumulationDataLabelService } from
'@syncfusion/ej2-angular-charts'
import { Component, OnInit, ViewChild } from '@angular/core';
import { ChartComponent, IAccLoadedEventArgs } from '@syncfusion/ej2-angular-
charts';
@Component({
  imports: [
    AccumulationChartModule
  ],
  providers: [PieSeriesService, AccumulationDataLabelService, ExportService],
  standalone: true,
  selector: 'app-container',
  template: `<div class="col-md-8">
    <button ej-button id='print' (click)='export()'>Export</button>
    <ejs-accumulationchart #chart id="chart-container">
      <e-accumulation-series-collection>

```

```

        <e-accumulation-series [dataSource]='piedata' xName='x' yName='y'
radius="100%"></e-accumulation-series>
    </e-accumulation-series-collection>
</ejs-accumulationchart>
</div>
})
export class AppComponent implements OnInit {
    public piedata?: Object[];
    public loaded: Function | any;
    @ViewChild('chart')
    public chartObj?: ChartComponent;
    ngOnInit(): void {
        this.piedata = [
            { x: 'Jan', y: 3, text: 'Jan: 3' }, { x: 'Feb', y: 3.5, text:
'Feb: 3.5' },
            { x: 'Mar', y: 7, text: 'Mar: 7' }, { x: 'Apr', y: 13.5,
text: 'Apr: 13.5' },
            { x: 'May', y: 19, text: 'May: 19' }, { x: 'Jun', y: 23.5,
text: 'Jun: 23.5' },
            { x: 'Jul', y: 26, text: 'Jul: 26' }, { x: 'Aug', y: 25,
text: 'Aug: 25' },
            { x: 'Sep', y: 21, text: 'Sep: 21' }, { x: 'Oct', y: 15,
text: 'Oct: 15' }];
    }
    export() {
        this.chartObj?.export('PNG', 'export');
    }
}

```

MAIN.TS

```

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

```

Appearance in Angular Accumulation chart component

Custom Color Palette

You can customize the default color of series or points by providing a custom color palette of your choice by using the [palettes](#) property.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { AccumulationChartModule } from '@syncfusion/ej2-angular-charts'
import { PieSeriesService, AccumulationLegendService,
AccumulationTooltipService, AccumulationAnnotationService,
    AccumulationDataLabelService } from '@syncfusion/ej2-angular-charts'
import { Component, OnInit } from '@angular/core';
import { data } from './datasource';
@Component({
    imports: [
        AccumulationChartModule
    ],

```

```

providers: [PieSeriesService, AccumulationLegendService,
AccumulationTooltipService, AccumulationDataLabelService,
  AccumulationAnnotationService],
standalone: true,
  selector: 'app-container',
  template: `<ejs-accumulationchart id="chart-container"
[legendSettings]='legendSettings'>
    <e-accumulation-series-collection>
      <e-accumulation-series [dataSource]='piedata' xName='x' yName='y'
type='Pie' [palettes]='palette'></e-accumulation-series>
    </e-accumulation-series-collection>
  </ejs-accumulationchart>`
})
export class AppComponent implements OnInit {
  public piedata?: Object[];
  public legendSettings?: Object;
  public palette?: string[];
  ngOnInit(): void {
    this.piedata = data;
    this.legendSettings = {
      visible: false
    };
    this.palette = ["#E94649", "#F6B53F", "#6FAAB0",
"#FF33F3", "#228B22", "#3399FF"]
  }
}

```

MAIN.TS

```

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

```

Animation**Fluid Animation**

Fluid animation used to animate series with updated dataSource continues animation rather than animation whole series. You can customize animation for a particular series using `animate` method.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { AccumulationChartModule, ExportService } from '@syncfusion/ej2-
angular-charts'
import { PieSeriesService, AccumulationDataLabelService } from
 '@syncfusion/ej2-angular-charts'
import { Component, ViewChild, OnInit } from '@angular/core';
import {
  AccumulationChart, AccumulationChartComponent,
  IAccLoadedEventArgs } from '@syncfusion/ej2-angular-charts';
@Component({
  imports: [
    AccumulationChartModule
  ],

```



```

providers: [PieSeriesService, AccumulationDataLabelService, ExportService],
standalone: true,
  selector: 'app-container',
  template:
    `<ejs-accumulationchart id="donut-container" #pie [title]="title"
    (loaded)="loaded($event)">
      <e-accumulation-series-collection>
        <e-accumulation-series name='Revenue' [dataSource]='data'
        xName='x' yName='y' [startAngle]="startAngle" [endAngle]="endAngle"
        innerRadius="40%" [dataLabel]="dataLabel"> </e-accumulation-series>
      </e-accumulation-series-collection>
    </ejs-accumulationchart>`
  })
export class AppComponent implements OnInit {
  public data: any;
  public dataLabel?: Object[];
  public count?: number ;
  public startAngle?: number ;
  public endAngle?: number ;
  public title?: string ;
  @ViewChild('pie')
  public pie?: AccumulationChartComponent | AccumulationChart;
  public loaded(args: IAccLoadedEventArgs): void {
    if (this.execute) {
      return;
    }
  }
  execute: any;
  ngOnInit(): void {
    this.data = [
      { 'x': 'Net-tution and Fees', y: 21, text: '21%' },
      { 'x': 'Self-supporting Operations', y: 21, text: '21%' },
      { 'x': 'Private Gifts', y: 8, text: '8%' },
      { 'x': 'All Other', y: 8, text: '8%' },
      { 'x': 'Local Revenue', y: 4, text: '4%' },
      { 'x': 'State Revenue', y: 21, text: '21%' },
      { 'x': 'Federal Revenue', y: 16, text: '16%' }
    ];
    this.dataLabel = {
      visible: true,
      position: 'Inside',
      name: '${point.y}',
      font: {
        color: 'white',
        fontWeight: 'Bold',
        size: '14px'
      }
    }
  } as any;
  let pieinterval = setInterval( () => {
    if (document.getElementById('donut-container')) {
      if (this.count === 0) {
        this.pie!.series[0].dataSource = [{ 'x': 'Net-tution and
Fees', y: 10 },
        { 'x': 'Self-supporting Operations', y: 10 },
        { 'x': 'Private Gifts', y: 13 }, { 'x': 'All Other', y:
14 },

```

```

y: 13 },
        { 'x': 'Local Revenue', y: 9 }, { 'x': 'State Revenue',
        { 'x': 'Federal Revenue', y: 8 }
    ];
    this.execute = true;
    this.pie?.animate();
    this.count++;
    } else if (this.count === 1) {
        this.pie!.series[0].dataSource = [
            { 'x': 'Net-tution and Fees', y: 120 }, { 'x': 'Self-
supporting Operations', y: 31 },
            { 'x': 'Private Gifts', y: 6 }, { 'x': 'All Other',
y: 12 },
            { 'x': 'Local Revenue', y: 25 }, { 'x': 'State
Revenue', y: 11 },
            { 'x': 'Federal Revenue', y: 12 }
        ];
        this.execute = true;
        this.pie?.animate();
        this.count++;
    } else if (this.count === 2) {
        this.pie!.series[0].dataSource = [
            { 'x': 'Net-tution and Fees', y: 6 }, { 'x': 'Self-
supporting Operations', y: 22 },
            { 'x': 'Private Gifts', y: 11 }, { 'x': 'All Other',
y: 15 },
            { 'x': 'Local Revenue', y: 13 }, { 'x': 'State
Revenue', y: 10 },
            { 'x': 'Federal Revenue', y: 8 }
        ];
        this.execute = true;
        this.pie?.animate();
        this.count++;
    } else if (this.count === 3) {
        this.pie!.series[0].dataSource = [
            { 'x': 'Net-tution and Fees', y: 15 }, { 'x': 'Self-
supporting Operations', y: 10 },
            { 'x': 'Private Gifts', y: 18 }, { 'x': 'All Other',
y: 20 },
            { 'x': 'Local Revenue', y: 30 }, { 'x': 'State
Revenue', y: 20 },
            { 'x': 'Federal Revenue', y: 25 }
        ];
        this.execute = true;
        this.pie?.animate();
        this.count++;
    } else if (this.count === 4) {
        this.pie!.series[0].dataSource = [
            { 'x': 'Net-tution and Fees', y: 21 }, { 'x': 'Self-
supporting Operations', y: 10 },
            { 'x': 'Private Gifts', y: 15 }, { 'x': 'All Other',
y: 15 },
            { 'x': 'Local Revenue', y: 11 }, { 'x': 'State
Revenue', y: 20 },
            { 'x': 'Federal Revenue', y: 60 }
        ];
        this.execute = true;
    }

```

```

        this.pie?.animate();
        this.count = 0;
    }
    } else {
        clearInterval(pieinterval);
    }
}, 3000);
this.count = 0;
this.startAngle = 0;
this.endAngle = 360;
this.title = 'Education Institutional Revenue';
}
}

```

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 Accumulation chart component

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

The accessibility compliance for the Accumulation chart component is outlined below.

| Accessibility Criteria | Compatibility |

| -- | -- |

| [WCAG 2.2](#) Support | |

| [Section 508](#) Support | |

| Screen Reader Support | |

| Right-To-Left Support | |

| Color Contrast | |

| Mobile Device Support | |

| Keyboard Navigation Support | |

| [Accessibility Checker](#) Validation | |

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

WAI-ARIA attributes

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

- `img` (role)
- `button` (role)
- `region` (role)
- `aria-label` (attribute)
- `aria-hidden` (attribute)
- `aria-pressed` (attribute)

Keyboard interaction

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

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

| --- | --- |

| **Alt + J** | Moves the focus to the Accumulation chart element. |

| **Tab** | Moves the focus to the next element in the Accumulation chart. |

| **Shift + Tab** | Moves the focus to the previous element in the Accumulation chart. |

| **Down Arrow** | Moves the focus to the data point left side from the selected point. |

| **Up Arrow** | Moves the focus to the data point right side from the selected point. |

| **Down/Left Arrow** | Moves the focus to the legend left side from the selected legend. |

| **Up/Right Arrow** | Moves the focus to the legend right side from the selected legend. |

| Enter/Space | Toggles the visibility of the corresponding series. |

| Ctrl + P | Prints the Accumulation chart. |

Ensuring accessibility

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

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

See also

- [Accessibility in Syncfusion Angular components](#)

<!-- markdownlint-disable MD033 -->

<!-- markdownlint-disable MD038 -->

Ej1 api migration in Angular Accumulation chart component

This article describes the API migration process of {Component Name} component from Essential JS 1 to Essential JS 2.

Accumulation Chart

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

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

| Annotation | **Property:** *annotation*

 <ej-chart>
<e-annotations>
<e-annotation>
<e-annotation>
</e-annotations>
</ej-chart> | **Property:** *e-annotation*

 <ejs-accumulationchart>
<e-annotations>
<e-annotation>
<e-annotation>
</e-annotations>
</ejs-accumulationchart> |

| Background | **Property:** *background*

 <ej-chart background='red'>
</ej-chart> | **Property:** *background*

 <ejs-accumulationchart background='red'>
</ejs-accumulationchart> |

| border of the chart | **Property:** *border*

 <ej-chart [border]='border'>
</ej-chart>

 this.border = { width: 2, color: 'red'} | **Property:** *border*

 <ejs-accumulationchart [border]='border'>
</ejs-accumulationchart>

 this.border = { width: 2, color: 'red'} |

| dataSource | **Property:** *Not applicable* | **Property:** *dataSource*

 <ejs-accumulationchart [dataSource]='data'>
</ejs-accumulationchart>

 this.data = 'data' |

| Persisting component's state between page reloads. | **Property:** *Not applicable* | **Property:** *enablePersistence*

 <ejs-accumulationchart [enablePersistence]='enablepersistence'>
</ejs-accumulationchart>

 this.enablepersistence = true |

| Animation after legend click | **Property:** *Not applicable* | **Property:** *enableAnimation*

 <ejs-accumulationchart [enableAnimation]= 'enableAnimation'>
</ejs-accumulationchart>

 this.enableAnimation = true |

| Enabling smart labels | **Property:** *enableSmartLabels*

 <ej-chart [enableSmartLabels]= 'enableSmartLabels' >
 </ej-chart>

 this.enableSmartLabels = true | **Property:** *enableSmartLabels*

 <ejs-accumulationchart [enableSmartLabels]= 'enablesmartlabels'>
 </ejs-accumulationchart>

 this.enablesmartlabels = true |

| Height of chart | **Property:** *size.height*

 <ej-chart [size]= 'size'>
</ej-chart>

 this.size = { height: '300' } | **Property:** *height*

 <ejs-accumulationchart [height]= 'height'>
 </ejs-accumulationchart>

 this.height = '300' |

| Width of chart | **Property:** *size.width*

 <ej-chart [size]= 'size'>
</ej-chart>

 this.size = { width: '300' } | **Property:** *width*

 <ejs-accumulationchart [width]= 'width'>
 </ejs-accumulationchart>

 this.width = '300' |

| Multi selection of chart | Not Applicable | **Property:** *isMultiSelect*

 <ejs-accumulationchart [isMultiSelect]= 'select'>
</ejs-accumulationchart>

 this.select = true |

| legend Settings | **Property:** *legend.visible*

 <ej-chart [legend]= 'legend'>
</ej-chart>

 this.legend = { visible: true } | **Property:** *legend.visible*

 <ejs-accumulationchart [legend]= 'legend'>
</ejs-accumulationchart>

 this.legend = { visible: true }

| Margin for the chart | **Property:** *margin*

 <ej-chart [margin]= 'margin'>
</ej-chart>

 this.margin = { top: 20, bottom: 23, right: 15, left: 10 } | **Property:** *margin*

 <ejs-accumulationchart [margin]= 'margin'>
</ejs-accumulationchart>

 this.margin = { top: 20, bottom: 23, right: 15, left: 10 }

| Selected Data Indexes | **Property:** *selectedDataPointIndexes*

 <ej-chart [selectedDataPointIndexes]= 'selectData'>
</ej-chart>

 this.selectedData = [{ seriesIndex: 2, pointIndex: 2}] | **Property:** *selectedDataIndexes*

 <ejs-accumulationchart [selectedDataIndexes]= 'selectData'>
</ejs-accumulationchart>

 this.selectData = [{ series: 0, point: 1}] |

| Selection Mode | **Property:** *commonSeriesOptions.selectionSettings.mode*

 <ej-chart [commonSeriesOptions.selectionSettings.mode]= 'selectData'>
</ej-chart>

 this.selectData = 'Point' | **Property:** *selectionMode*

 <ejs-accumulationchart [selectionMode]= 'selectionmode'>
</ejs-accumulationchart>

 this.selectionmode = 'Point' |

| Series | **Property:** *series*

 <ej-chart>
 <e-series-collection>
 <e-series>
 </e-series>
 </e-series-collection>
 </ej-chart> | **Property:** *e-accumulation-series*

 <ejs-accumulationchart>
 <e-accumulation-series-collection>
 <e-accumulation-series>
 </e-accumulation-series>
 </e-accumulation-series-collection>
 </ejs-accumulationchart> |

| Title text | **Property:** *title.text*

 <ej-chart title.text="Expenditures">
</ej-chart> |
Property: *title*

 <ejs-accumulationchart [title]='title'>
</ejs-accumulationchart>

this.title = 'Chart title' |

| SubTitle text | **Property:** *title.subTitle.text*

 <ej-chart title.subTitle.text="subTitle">

</ej-chart> | **Property:** *title*

 <ejs-accumulationchart [subTitle]='title'>
</ejs-accumulationchart>

 this.title = 'Chart subtitle' |

| tooltip | **Property:** *tooltip*

 <ej-chart>
 <e-series-collection>
 <e-series
 [tooltip]='tooltip'>
 </e-series>
 </e-series-collection>
 </ej-chart>

 this.tooltip = {} | **Property:** *tooltip*

 <ejs-accumulationchart [tooltip]='tooltip'>

</ejs-accumulationchart>

 this.tooltip = {} |

Annotations

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

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

| Annotation | **Property:** *annotation*

 <ej-chart>
 <e-annotations>
 <e-
 annotation [visible]='visible'>
<e-annotation>
</e-annotations>
</ej-chart>

 this.visible = true | **Property:** *e-annotation*

 <ejs-accumulationchart>
 <e-
 accumulation-annotations>
<e-accumulation-annotation>
 </e-accumulation-
 annotation>
 </e-accumulation-annotations>
</ejs-accumulationchart> |

| Annotation content | **Property:** *annotation.content*

 <ej-chart>
<e-
 annotations>
<e-annotation>
<e-annotation content='<div>Chart</div>'>
</e-
 annotations>
</ej-chart> | **Property:** *annotation.content*

 <ejs-accumulationchart>

<e-accumulation-annotations>
 <e-annotation content='<div>Chart</div>'>
 </e-
 accumulation-annotation>
 </e-accumulation-annotations>
</ejs-accumulationchart> |

| Coordinate unit of annotation | **Property:** *annotation.coordinateUnit*

 <ej-chart>
<e-
 annotations>
<e-annotation>
<e-annotation coordinateUnit='Pixel'>
</e-
 annotations>
</ej-chart> | **Property:** *annotation.coordinateUnits*

 <ejs-
 accumulationchart>
<e-accumulation-annotations>
 <e-accumulation-annotation
 coordinateUnits='Pixel'>
</e-accumulation-annotation>
 </e-accumulation-
 annotations>
 </ejs-accumulationchart> |

| Horizontal alignment of annotation | **Property:** *annotation.horionalAlignment*

 <ej-
 chart>
<e-annotations>
<e-annotation>
<e-annotation
 horionalAlignment='near'>
</e-annotations>
</ej-chart> | **Property:**
annotation.horionalAlignment

 <ejs-accumulationchart>
<e-accumulation-
 annotations>
 <e-accumulation-annotation horionalAlignment='Near'>
 </e-
 accumulation-annotation>
 </e-accumulation-annotations>
 </ejs-accumulationchart> |

| Margin for annotation | **Property:** *annotation.margin*

 <ej-chart>
<e-
 annotations>
<e-annotation>
<e-annotation [margin]='margin'>
</e-
 annotations>
</ej-chart>
 this.margin = { } | **Property:** Not Applicable.

| Opacity for annotation | **Property:** *annotation.opacity*

 <ej-chart>
<e-annotations>
<e-annotation>
<e-annotation [opacity]='opacity'>
</e-annotations>
</ej-chart>
 this.opacity = 2 | **Property:** Not Applicable.

| region of annotation | **Property:** *annotation.region*

 <ej-chart>
<e-annotations>
<e-annotation>
<e-annotation region='Chart'>
</e-annotations>
</ej-chart> | **Property:** *annotation.region*

 <ejs-accumulationchart>
<e-accumulation-annotations>
 <e-accumulation-annotation region='Chart'>
 </e-accumulation-annotation>
 </e-accumulation-annotations>
 </ejs-accumulationchart> |

| Vertical alignment of annotation | **Property:** *annotation.verticalAlignment*

 <ej-chart>
<e-annotations>
<e-annotation>
<e-annotation verticalAlignment='Top'>
</e-annotations>
</ej-chart> | **Property:** *annotation.verticalAlignment*

 <ejs-accumulationchart>
<e-accumulation-annotations>
 <e-accumulation-annotation verticalAlignment='Top'>
 </e-accumulation-annotation>
 </e-accumulation-annotations>
 </ejs-accumulationchart> |

| X offset for annotation | **Property:** *annotation.x*

 <ej-chart>
<e-annotations>
<e-annotation>
<e-annotation [x]='xvalue'>
</e-annotations>
</ej-chart>

 this.xvalue = 2 | **Property:** *annotation.x*

 <ejs-accumulationchart>
<e-accumulation-annotations>
 <e-accumulation-annotation x='xvalue'>
</e-annotations>
</ejs-accumulationchart>

 this.xvalue = 2 |

| Y offset for annotation | **Property:** *annotation.y*

 <ej-chart>
<e-annotations>
<e-annotation>
<e-annotation [Y]='yvalue'>
 </e-annotations>
 </ej-chart>
 this.yvalue = 'axis' | **Property:** *annotation.y*

 <ejs-accumulationchart>
<e-accumulation-annotations>
 <e-accumulation-annotations [Y]='yvalue'>
 <e-accumulation-annotation>
 </e-accumulation-annotations>
 </ejs-accumulationchart>

 this.yvalue = 'axis' |

Series

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

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

| Series | **Property:** *series*

 <ej-chart>
<e-series-collection>
<e-series>
</e-series>
</e-series-collection>
 </ej-chart> | **Property:** *series*

 <ejs-accumulationchart>
<e-accumulation-series-collection>
<e-accumulation-series>
 </e-accumulation-series>
 </e-accumulation-series-collection>
 </ejs-accumulationchart> |

| Enable animation for series | **Property:** *series.enableAnimation*

 <ej-chart>
<e-series-collection>
<e-series [enableAnimation]='animation'>
 </e-series>
 </e-series-collection>
 </ej-chart>
 this.animation = true | **Property:** *series.animation.enable*

 <ejs-accumulationchart>
<e-accumulation-series-collection>
<e-accumulation-series [animation]='animation'>
 </e-accumulation-series>
 </e-accumulation-series-collection>
 </ejs-accumulationchart>

 this.animation = { enable: true } |

| Duration animation for series | **Property:** *series.animationDuration*

 <ej-chart>
 <e-series-collection>
 <e-series [animationDuration]='animation' >
 </e-series>
 </e-series-collection>
</ej-chart>

 this.animation = 100 | **Property:** *series.animation.duration*

 <ejs-accumulationchart>
<e-accumulation-series-collection>
<e-accumulation-series [animation]='animation' >
 </e-accumulation-series>
 </e-accumulation-series-collection>
</ejs-accumulationchart>

 this.animation = { duration: 100} |

| Animation delay for series | **Property:** Not Applicable | **Property:** *series.animation.delay*

 <ejs-accumulationchart>
<e-accumulation-series-collection>
<e-accumulation-series [animation]='animation' >
</e-accumulation-series>
 </e-accumulation-series-collection>
</ejs-accumulationchart>

 this.animation = { delay: 100 } |

| border of the series | **Property:** *border*

 <ej-chart>
<e-series-collection>
<e-series [border]='border' >
</e-series>
</e-series-collection>
</ej-chart>

 this.border = { width: 2, color: 2} | **Property:** *border*

 <ejs-accumulationchart>
 <e-accumulation-series-collection>
 <e-accumulation-series border='border'>
 </e-accumulation-series>
 </e-accumulation-series-collection>
 </ejs-accumulationchart>

 this.border = { width: 2, color: 'red' } |

| DataLabel of series | **Property:** *series.marker.dataLabel*

 <ej-chart>
<e-series-collection>
<e-series [marker]='marker'>
</e-series>
</e-series-collection>
</ej-chart>

 this.marker = { dataLabel: { visible: true } } | **Property:** *series.marker.dataLabel*

 <ejs-accumulationchart>
<e-accumulation-series-collection>
<e-accumulation-series [marker]='marker'>
</e-accumulation-series>
</e-accumulation-series-collection>
</ejs-accumulationchart>

 this.marker = { dataLabel: { visible: true } } |

| dataSource for series | **Property:** *series.dataSource*

 <ej-chart>
<e-series-collection>
<e-series [dataSource]='data' >
</e-series>
</e-series-collection>
</ej-chart>

 this.data = [] | **Property:** *series.dataSource*

 <ejs-accumulationchart>
<e-accumulation-series-collection>
 <e-accumulation-series [dataSource]='data' >
 </e-accumulation-series>
 </e-accumulation-series-collection>
 </ejs-accumulationchart>

 this.data = [] |

| enableTooltip for series | **Property:** *tooltip.visible*

 <ej-chart>
 <e-series-collection>
 <e-series [tooltip.visible]='true' >
 </e-series>
 </e-series-collection>
 </ej-chart> | **Property:** *tooltip*

 <ejs-accumulationchart>
<e-accumulation-series-collection>
 <e-accumulation-series [tooltip]='tooltip' >
 </e-accumulation-series>
 </e-accumulation-series-collection>
 </ejs-accumulationchart>

 this.tooltip = { enable:true } |

| start angle | **Property:** *startAngle*

 <ej-chart>
 <e-series-collection>
 <e-series [startAngle]=270 >
 </e-series>
 </e-series-collection>
 </ej-chart> | **Property:** *startAngle*

 <ejs-accumulationchart>
<e-accumulation-series-collection>
 <e-accumulation-series [startAngle]='startAngle' >
 </e-accumulation-series>
 </e-accumulation-series-collection>
 </ejs-accumulationchart>

 this.startAngle = 270 |

| end angle | **Property:** *endAngle*

 <ej-chart>
 <e-series-collection>
 <e-series [endAngle]=270 >
 </e-series>
 </e-series-collection>
 </ej-chart> | **Property:** *endAngle*

 <ejs-accumulationchart>
<e-accumulation-series-collection>
 <e-accumulation-series [endAngle]='endAngle' >
 </e-accumulation-series>
 </e-accumulation-series-collection>
 </ejs-accumulationchart>

 this.endAngle = 270 |

| explode | **Property:** *explode*

 <ej-chart>
 <e-series-collection>
 <e-series [explode]="true" >
 </e-series>
 </e-series-collection>
 </ej-chart> | **Property:** *explode*

 <ejs-accumulationchart>
<e-accumulation-series-collection>
 <e-accumulation-series [explode]='explode' >
 </e-accumulation-series>
 </e-accumulation-series-collection>
 </ejs-accumulationchart>

 this.explode = true |

| explodeAll | **Property:** *explodeAll*

 <ej-chart>
 <e-series-collection>
 <e-series [explodeAll]="true" >
 </e-series>
 </e-series-collection>
 </ej-chart> | **Property:** *explodeAll*

 <ejs-accumulationchart>
<e-accumulation-series-collection>
 <e-accumulation-series [explodeAll]='explodeAll' >
 </e-accumulation-series>
 </e-accumulation-series-collection>
 </ejs-accumulationchart>

 this.explodeAll = true |

| explodeIndex | **Property:** *explodeIndex*

 <ej-chart>
 <e-series-collection>
 <e-series [explodeIndex]=1 >
 </e-series>
 </e-series-collection>
 </ej-chart> | **Property:** *explodeIndex*

 <ejs-accumulationchart>
<e-accumulation-series-collection>
 <e-accumulation-series [explodeIndex]='explodeIndex' >
 </e-accumulation-series>
 </e-accumulation-series-collection>
 </ejs-accumulationchart>

 this.explodeIndex = 0 |

| explodeOffset | **Property:** *explodeOffset*

 <ej-chart>
 <e-series-collection>
 <e-series [explodeOffset]=30% >
 </e-series>
 </e-series-collection>
 </ej-chart> | **Property:** *explodeOffset*

 <ejs-accumulationchart>
<e-accumulation-series-collection>
 <e-accumulation-series [explodeOffset]='explodeOffset' >
 </e-accumulation-series>
 </e-accumulation-series-collection>
 </ejs-accumulationchart>

 this.explodeOffset = 30% |

| gapRatio | **Property:** *gapRatio*

 <ej-chart>
 <e-series-collection>
 <e-series [gapRatio]=0.1 >
 </e-series>
 </e-series-collection>
 </ej-chart> | **Property:** *gapRatio*

 <ejs-accumulationchart>
<e-accumulation-series-collection>
 <e-accumulation-series [gapRatio]='gapRatio' >
 </e-accumulation-series>
 </e-accumulation-series-collection>
 </ejs-accumulationchart>

 this.gapRatio = 0.1 |

| gapWidth | **Property:** *gapWidth*

 <ej-chart>
 <e-series-collection>
 <e-series [gapWidth]=0.1 >
 </e-series>
 </e-series-collection>
 </ej-chart> | **Property:** *gapWidth*

 <ejs-accumulationchart>
<e-accumulation-series-collection>
 <e-accumulation-series [gapWidth]='gapWidth' >
 </e-accumulation-series>
 </e-accumulation-series-collection>
 </ejs-accumulationchart>
 this.gapWidth = 1 |

| inner radius of the accumulation chart | **Property:** *doughnutCoefficient*

 <ej-chart>
 <e-series-collection>
 <e-series [doughnutCoefficient]=0.1 >
 </e-series>
 </e-series-collection>
 </ej-chart> | **Property:** *innerRadius*

 <ejs-accumulationchart>
<e-accumulation-series-collection>
 <e-accumulation-series innerRadius='40%'>


```
</e-accumulation-series> <br> </e-accumulation-series-collection> <br> </ejs-accumulationchart>|
```

```
| Legend shape of the series | Property: Not applicable | Property: legendShape <br/> <br/> <ejs-accumulationchart> <br><e-accumulation-series-collection> <br> <e-accumulation-series-legendShape='Rectangle' ><br> </e-accumulation-series> <br> </e-accumulation-series-collection> <br> </ejs-accumulationchart> |
```

```
| name of the series | Property: name <br/> <br/> <ej-chart> <br> <e-series-collection> <br> <e-series [name]='name' > <br> </e-series> <br> </e-series-collection> <br> </ej-chart> | Property: name <br/> <br/> <ejs-accumulationchart> <br><e-accumulation-series-collection> <br> <e-accumulation-series name='name'><br> </e-accumulation-series> <br> </e-accumulation-series-collection> <br> </ejs-accumulationchart>|
```

```
| neck height for funnel series | Property: funnelHeight <br/> <br/> <ej-chart> <br> <e-series-collection> <br> <e-series [funnelHeight]="20%" > <br> </e-series> <br> </e-series-collection> <br> </ej-chart> | Property: neckHeight <br/> <br/> <ejs-accumulationchart> <br><e-accumulation-series-collection> <br> <e-accumulation-series [neckHeight]]='neckHeight'><br> </e-accumulation-series> <br> </e-accumulation-series-collection> <br> </ejs-accumulationchart> <br> this.neckHeight='30%' |
```

```
| neck width for funnel series | Property: funnelWidth <br/> <br/> <ej-chart> <br> <e-series-collection> <br> <e-series [funnelWidth]="20%" > <br> </e-series> <br> </e-series-collection> <br> </ej-chart> | Property: neckWidth <br/> <br/> <ejs-accumulationchart> <br><e-accumulation-series-collection> <br> <e-accumulation-series [neckWidth]]='neckWidth'><br> </e-accumulation-series> <br> </e-accumulation-series-collection> <br> </ejs-accumulationchart> <br> this.neckWidth='30%' |
```

```
| opacity for series | Property: opacity <br/> <br/> <ej-chart> <br> <e-series-collection> <br> <e-series [opacity]="0.4" > <br> </e-series> <br> </e-series-collection> <br> </ej-chart> | Property: opacity <br/> <br/> <ejs-accumulationchart> <br><e-accumulation-series-collection> <br> <e-accumulation-series [opacity]]='opacity'><br> </e-accumulation-series> <br> </e-accumulation-series-collection> <br> </ejs-accumulationchart> <br> this.opacity='0.4' |
```

```
| palettes for series | Property: Not applicable | Property: palettes <br/> <br/> <ejs-accumulationchart> <br><e-accumulation-series-collection> <br> <e-accumulation-series [palettes]]='palettes'><br> </e-accumulation-series> <br> </e-accumulation-series-collection> <br> </ejs-accumulationchart> <br> this.palettes=[] |
```

```
| point color mapping name for series | Property: pointColorMappingName <br/> <br/> <ej-chart> <br> <e-series-collection> <br> <e-series [pointColorMappingName]="color" > <br> </e-series> <br> </e-series-collection> <br> </ej-chart> | Property: opacity <br/> <br/> <ejs-accumulationchart> <br><e-accumulation-series-collection> <br> <e-accumulation-series [pointColorMappingName]]='pointColorMappingName'><br> </e-accumulation-series> <br> </e-accumulation-series-collection> <br> </ejs-accumulationchart> <br> this.pointColorMappingName='color' |
```

```
| Mode of pyramid series | Property: pyramidMode <br/> <br/> <ej-chart> <br> <e-series-collection> <br> <e-series [pyramidMode]="Surface" > <br> </e-series> <br> </e-series-
```

```
collection> <br> </ej-chart> | Property: pyramidMode <br/> <br/> <ejs-accumulationchart>
<br><e-accumulation-series-collection> <br> <e-accumulation-series
pyramidMode='Surface'><br> </e-accumulation-series> <br> </e-accumulation-series-
collection> <br> </ejs-accumulationchart> |
```

```
| query for datasource for series | Property: query <br/> <br/> <ej-chart> <br> <e-series-collection>
<br> <e-series [query]='"' > <br> </e-series> <br> </e-series-collection> <br> </ej-chart> |
Property: query <br/> <br/> <ejs-accumulationchart> <br><e-accumulation-series-collection> <br>
<e-accumulation-series pyramidMode=""><br> </e-accumulation-series> <br> </e-accumulation-
series-collection> <br> </ejs-accumulationchart> |
```

```
| Radius of Pie series | Property: pieCoefficient <br/> <br/> <ej-chart> <br> <e-series-collection>
<br> <e-series [pieCoefficient]=0.4 > <br> </e-series> <br> </e-series-collection> <br> </ej-chart>
| Property: radius <br/> <br/> <ejs-accumulationchart> <br><e-accumulation-series-collection>
<br> <e-accumulation-series radius="100%"><br> </e-accumulation-series> <br> </e-
accumulation-series-collection> <br> </ejs-accumulationchart> |
```

```
| Selection Style of Accumulation chart | Property: Not applicable | Property: selectionStyle <br/>
<br/> <ejs-accumulationchart> <br><e-accumulation-series-collection> <br> <e-accumulation-
series [selectionStyle]='selectionStyle'><br> </e-accumulation-series> <br> </e-accumulation-
series-collection> <br> </ejs-accumulationchart> <br> this.selectionStyle={} |
```

```
| tooltip Mapping name | Property: Not applicable | Property: tooltipMappingName <br/> <br/>
<ejs-accumulationchart> <br><e-accumulation-series-collection> <br> <e-accumulation-series
[tooltipMappingName]='tooltipMappingName'><br> </e-accumulation-series> <br> </e-
accumulation-series-collection> <br> </ejs-accumulationchart> <br>
this.tooltipMappingName='text' |
```

```
| Type of series | Property: type <br/> <br/> <ej-chart> <br> <e-series-collection> <br> <e-series
type='Funnel' > <br> </e-series> <br> </e-series-collection> <br> </ej-chart> | Property: type
<br/> <br/> <ejs-accumulationchart> <br><e-accumulation-series-collection> <br> <e-
accumulation-series type='Funnel'><br> </e-accumulation-series> <br> </e-accumulation-
series-collection> <br> </ejs-accumulationchart> |
```

```
| Name of the property in the datasource that contains x value for the series | Property: xName <br/>
<br/> <ej-chart> <br> <e-series-collection> <br> <e-series xName='x' > <br> </e-series> <br> </e-
series-collection> <br> </ej-chart> | Property: xName <br/> <br/> <ejs-accumulationchart>
<br><e-accumulation-series-collection> <br> <e-accumulation-series xName='x'><br> </e-
accumulation-series> <br> </e-accumulation-series-collection> <br> </ejs-accumulationchart> |
```

```
| Name of the property in the datasource that contains y value for the series | Property: yName <br/>
<br/> <ej-chart> <br> <e-series-collection> <br> <e-series yName='y' > <br> </e-series> <br> </e-
series-collection> <br> </ej-chart> | Property: yName <br/> <br/> <ejs-accumulationchart>
<br><e-accumulation-series-collection> <br> <e-accumulation-series yName='y'><br> </e-
accumulation-series> <br> </e-accumulation-series-collection> <br> </ejs-accumulationchart> |
```

```
| Grouping | Property: Not applicable | Property: groupTo <br/> <br/> <ejs-accumulationchart>
<br><e-accumulation-series-collection> <br> <e-accumulation-series [groupTo]='11'><br> </e-
accumulation-series> <br> </e-accumulation-series-collection> <br> </ejs-accumulationchart> |
```

| GroupMode | **Property:** *Not applicable* | **Property:** *groupMode*

 <ejs-accumulationchart>
<e-accumulation-series-collection>
 <e-accumulation-series [groupMode]='Point'>
 </e-accumulation-series>
 </e-accumulation-series-collection>
 </ejs-accumulationchart> |

DataLabel

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

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

| DataLabel of series| **Property:** *series.marker.dataLabel*

 <ej-chart>
<e-series-collection>
<e-series [marker]='marker'>
</e-series>
</e-series-collection>
</ej-chart>

 this.marker = { dataLabel: { visible: true } } | **Property:** *series.marker.dataLabel*

 <ejs-accumulationchart>
<e-accumulation-series-collection>
<e-accumulation-series [marker]='marker'>
</e-accumulation-series>
</e-accumulation-series-collection>
</ejs-accumulationchart>

 this.marker = { dataLabel: { visible: true } } |

| Border of datalabel| **Property:** *marker.dataLabel.border*

 <ej-chart>
<e-series-collection>
<e-series [marker]='marker'>
</e-series>
</e-series-collection>
</ej-chart>

 this.marker = { dataLabel: { border: { color: 'red', width: 2 } } } | **Property:** *marker.dataLabel.border*

 <ejs-accumulationchart>
<e-accumulation-series-collection>
<e-accumulation-series [marker]='marker'>
 </e-accumulation-series>
 </e-accumulation-series-collection>
 </ejs-accumulationchart>

 this.marker = { dataLabel: { border: { width: 2, color: 'blue' } } } |

| Border of datalabel| **Property:** *marker.dataLabel.border*

 <ej-chart>
<e-series-collection>
<e-series [marker]='marker'>
</e-series>
</e-series-collection>
</ej-chart>

 this.marker = { dataLabel :{ connectorLine: { type: 'Curve', width: 2 } } } | **Property:** *marker.dataLabel.border*

 <ejs-accumulationchart>
 <e-accumulation-series-collection>
 <e-accumulation-series [marker]='marker'>
 </e-accumulation-series>
 </e-accumulation-series-collection>
 </ejs-accumulationchart>

 this.marker = { dataLabel : { connectorStyle: { type: 'Curve', width: 2 } } } |

| Fill color of datalabel| **Property:** *marker.dataLabel.fill*

 <ej-chart>
<e-series-collection>
<e-series [marker]='marker'>
</e-series>
</e-series-collection>
</ej-chart>

 this.marker = { dataLabel: { fill: 'red' } } | **Property:** *marker.dataLabel.fill*

 <ejs-accumulationchart>
<e-accumulation-series-collection>
<e-accumulation-series [marker]='marker'>
 </e-accumulation-series>
 </e-accumulation-series-collection>
 </ejs-accumulationchart>

 this.marker = { dataLabel: { fill: 'red' } } |

| font for datalabel| **Property:** *marker.dataLabel.font*

 <ej-chart>
<e-series-collection>
<e-series [marker]='marker'>
</e-series>
</e-series-collection>
</ej-chart>

 this.marker = { dataLabel: { font:{} } } | **Property:** *marker.dataLabel.font*

 <ejs-accumulationchart>
<e-accumulation-series-collection>
 <e-accumulation-series [marker]='marker'>
 </e-accumulation-series>
 </e-accumulation-series-collection>
 </ejs-accumulationchart>

 this.marker = { dataLabel: { font:{} } } |

| position | **Property:** *marker.dataLabel.position*

 <ej-chart>
 <e-series-collection>

 <e-series [marker]='marker'>
</e-series>
 </e-series-collection>
 </ej-chart>

 this.marker = { dataLabel: { position:'Inside' } } | **Property:** *marker.dataLabel.position*

 <ejs-accumulationchart>
<e-accumulation-series-collection>
 <e-accumulation-series [marker]='marker'>

 </e-accumulation-series>
 </e-accumulation-series-collection>
 </ejs-accumulationchart>

 this.marker = { dataLabel: { position:'Inside' } } |

| Rounded corner radius X | **Property:** *Not Applicable* | **Property:** *rx*

 <ejs-accumulationchart>

<e-accumulation-series-collection>
 <e-accumulation-series rx="10">
 </e-accumulation-series>

 </e-accumulation-series-collection>
 </ejs-accumulationchart> |

| Rounded corner radius Y | **Property:** *Not Applicable* | **Property:** *ry*

 <ejs-accumulationchart>

<e-accumulation-series-collection>
 <e-accumulation-series ry="10">
 </e-accumulation-series>

 </e-accumulation-series-collection>
 </ejs-accumulationchart> |

| HTML template in dataLabel | **Property:** *marker.dataLabel.template*

 <ej-chart>
 <e-series-collection>

 <e-series [marker]='marker'>
</e-series>
 </e-series-collection>
 </ej-chart>

 this.marker = { dataLabel: { template:"" } } | **Property:** *marker.dataLabel.template*

 <ejs-accumulationchart>
<e-accumulation-series-collection>
 <e-accumulation-series [marker]='marker'>

 </e-accumulation-series>
 </e-accumulation-series-collection>
 </ejs-accumulationchart>

 this.marker = { dataLabel: { template:"" } } |

Legend

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

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

| Visibility of legend | **Property:** *legend.visible*

 <ej-chart [legend]='legend'>
</ej-chart>

 this.legend = { visible: true } | **Property:** *legend.visible*

 < ej-accumulationchart
 [legend]='legend'>
</ ej-accumulationchart>

 this.legend = { visible: true } |

| Height of legend | **Property:** *legend.size.height*

 <ej-chart [legend]='legend'>
</ej-chart>

 this.legend = { size: { height: 20 } } | **Property:** *legend.height*

 < ej-accumulationchart
 [legend]='legend'>
 </ej-accumulationchart>

 this.legend = { height: '40' } |

| Width of legend | **Property:** *legend.size.width*

 <ej-chart [legend]='legend'>
</ej-chart>

 this.legend = { size: { width: 20 } } | **Property:** *legend.height*

 < ej-accumulationchart
 [legend]='legend'>
 </ej-accumulationchart>

 this.legend = { width: '40' } |

| Location of legend | **Property:** *legend.location*

 <ej-chart [legend]='legend'>
</ej-chart>

 this.legend = { location: { x: 10, y: 30 } } | **Property:** *legend.location*

 < ej-accumulationchart
 [legend]='legend'>
 </ej-accumulationchart>

 this.legend = { location: { x: 10, y: 30 } } |

| Padding of legend | **Property:** *legend.padding*

 <ej-chart [legend]='legend'>
</ej-chart>

 this.legend = { padding: 20 } | **Property:** *legend.padding*

 <ejs-accumulationchart [legend]='legend'>
</ ej-accumulationchart>

 this.legend = { padding: 8 } |

| Alignment of legend | **Property:** *legend.alignment*

 <ej-chart [legend]='legend'>
</ej-chart>

 this.legend = { alignment: 'near' } | **Property:** *legend.alignment*

 <ejs-accumulationchart [legend]='legend'>
</ ej-accumulationchart>

 this.legend = { alignment: 'near' } |

| Text style of legend | **Property:** *legend.font*

 <ej-chart [legend]='legend'>
</ej-chart>

 this.legend = { font: { size: '12px', color: 'red' } } | **Property:** *legend.textStyle*

 <ejs-accumulationchart [legend]='legend'>
</ ej-accumulationchart>

 this.legend = { textStyle: { size: '12px', color: 'red' } } |

| Shape height of legend | **Property:** *legend.itemStyle.height*

 <ej-chart [legend]='legend'>
</ej-chart>

 this.legend = { itemStyle: { height: 20 } } | **Property:** *legend.shapeHeight*

 <ejs-accumulationchart [legend]='legend'>
</ ej-accumulationchart>

 this.legend = { shapeHeight: 20 } |

| Shape width of legend | **Property:** *legend.itemStyle.width*

 <ej-chart [legend]='legend'>
</ej-chart>

 this.legend = { itemStyle: { width: 20 } } | **Property:** *legend.shapeWidth*

 <ejs-accumulationchart [legend]='legend'>
</ ej-accumulationchart>

 this.legend = { shapeWidth: 20 } |

| Shape border of legend | **Property:** *legend.itemStyle.border*

 <ej-chart [legend]='legend'>
</ej-chart>

 this.legend = { itemStyle: { border: { width: 2, color: 'red' } } } | **Property:** *legend.shapeBorder*

 <ejs-accumulationchart [legend]='legend'>
</ ej-accumulationchart>

 this.legend = { shapeBorder: { color: 'red', width: 2 } } |

| Shape padding of legend | **Property:** *legend.itemPadding*

 <ej-chart [legend]='legend'>
</ej-chart>

 this.legend = { itemPadding: 10 } | **Property:** *legend.shapePadding*

 <ejs-accumulationchart [legend]='legend'>
 </ejs-accumulationchart>

 this.legend = { shapePadding: 10 } |

| Background of legend | **Property:** *legend.background*

 <ej-chart [legend]='legend'>
</ej-chart>

 this.legend = { background: 'transparent' } } | **Property:** *legend.background*

 <ejs-accumulationchart [legend]='legend'>
</ ej-accumulationchart>

 this.legend = { background: 'transparent' } |

| Opacity of legend | **Property:** *legend.opacity*

 <ej-chart [legend]='legend'>
</ej-chart>

 this.legend = { opacity: 0.7 } } | **Property:** *legend.opacity*

 <ejs-accumulationchart [legend]='legend'>
</ ej-accumulationchart>

 this.legend = { opacity: 0.6 } |

| Toggle visibility of series legend | **Property:** *legend.toggleSeriesVisibility*

 <ej-chart [legend]='legend'>
</ej-chart>

 this.legend = { toggleSeriesVisibility: true } | **Property:**

legend.toggleVisibility

 < ej-accumulationchart [legend]='legend'>
</ ej-accumulationchart>

 this.legend = { toggleVisibility: true } |

| Title of legend | **Property:** *legend.title*

 <ej-chart [legend]='legend'>
</ej-chart>

 this.legend = { title: 'legend title' } | **Property:** Not Applicable |

| Text over flow of legend | **Property:** *legend.textOverflow*

 <ej-chart [legend]='legend'>
</ej-chart>

 this.legend = { textOverflow: 'Trim' } | **Property:** *legend.textOverflow*

 < ej-accumulationchart [legend]='legend'>
</ ej-accumulationchart>

 this.legend = { textStyle:{ textOverflow: 'Trim' } } |

| Scroll bar for legend | **Property:** *legend.enableScrollBar*

 <ej-chart [legend]='legend'>
</ej-chart>

 this.legend = { enableScrollBar: true } | **Property:** Not Applicable |

| Row count for legend | **Property:** *legend.rowCount*

 <ej-chart [legend]='legend'>
</ej-chart>

 this.legend = { rowCount: 2 } | **Property:** Not Applicable |

| Column count for legend | **Property:** *legend.columnCount*

 <ej-chart [legend]='legend'>
</ej-chart>

 this.legend = { columnCount: 2 } | **Property:** Not Applicable |

| Fill color for legend | **Property:** *legend.fill*

 <ej-chart [legend]='legend'>
</ej-chart>

 this.legend = { fill: 2 } | **Property:** Not Applicable |

Methods

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

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

| Animation | **Property:** *chart.animate*

 <ej-chart (animate)='animation'>
</ej-chart>

 this.animate(args) { } | **Property:** Not Applicable |

| Redraw | **Property:** *chart.redraw*

 <ej-chart (redraw)='redraw'>
 </ej-chart>

 this.redraw(args) { } | **Property:** *chart.refresh*

 <ej-accumulationchart (refresh)='refresh'>
 </ej-accumulationchart>

 this.refresh(args) { } |

| print | **Property:** *chart.print*

 <ej-chart (print)='print'>
</ej-chart>

 this.print(args) { } | **Property:** *chart.print*

 <ej-accumulationchart (print)='print'>
</ej-accumulationchart>

 this.print(args) { } |

| export | **Property:** *chart.export*

 <ej-chart (print)='print'>
</ej-chart>

 this.print(args) { } | **Property:** *chart.export*

 <ej-accumulationchart (export)='export'>
</ej-accumulationchart>

 this.export(args) { } |

| add series | **Property:** Not Applicable | **Property:** *chart.export*

 <ej-accumulationchart (addSeries)='addSeries'>
 </ej-accumulationchart>

 this.addSeries(args) { } |

| remove series | **Property:** Not Applicable | **Property:** *chart.export*

 <ej-accumulationchart (removeSeries)='removeSeries'>
</ej-accumulationchart>

 this.removeSeries(args) { } |

Events

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

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

| Fires on annotation click | **Property:** *chart.annotationClick*

 <ej-chart (annotationClick)=‘annotationClick(\$event)’>
</ej-chart>

 this.annotationClick(args) { } | **Property:** Not Applicable |

| Fires on after animation | **Property:** *chart.animationComplete*

 <ej-chart (animationComplete)=‘animationComplete(\$event)’>
</ej-chart>

 this.animationComplete(args) { } | **Property:** *chart.animationComplete*

 <ejs-accumulationchart (refresh)=‘refresh’>
</ejs-accumulationchart>

 this.refresh(args) { } |

| Fires on after chart resize | **Property:** *chart.afterResize*

 <ej-chart (afterResize)=‘afterResize(\$event)’>
</ej-chart>

 this.afterResize(args) { } | **Property:** Not Applicable |

| Fires before resize | **Property:** *chart.beforeResize*

 <ej-chart (beforeResize)=‘beforeResize(\$event)’>
</ej-chart>

 this.beforeResize(args) { } | **Property:** *chart.resized*

 <ejs-accumulationchart (resized)=‘resized(\$event)’>
</ejs-accumulationchart>

 this.resized(args) { } |

| Fires chart click | **Property:** *chart.chartClick*

 <ej-chart (chartClick)=‘chartClick(\$event)’>
</ej-chart>

 this.chartClick(args) { } | **Property:** *chart.chartMouseClicked*

 <ejs-accumulationchart (chartMouseClicked)=‘chartMouseClicked(\$event)’>
</ejs-accumulationchart>

 this.chartMouseClicked(args) { } |

| Fires chart mouse leave | **Property:** *chart.chartMouseLeave*

 <ej-chart (chartMouseLeave)=‘chartMouseLeave(\$event)’>
</ej-chart>

 this.chartMouseLeave(args) { } | **Property:** *chart.chartMouseClicked*

 <ejs-accumulationchart (chartMouseLeave)=‘chartMouseLeave(\$event)’>
</ejs-accumulationchart>

 this.chartMouseLeave(args) { } |

| Fires chart mouse move | **Property:** *chart.chartMouseMove*

 <ej-chart (chartMouseMove)=‘chartMouseMove(\$event)’>
</ej-chart>

 this.chartMouseMove(args) { } | **Property:** *chart.chartMouseMove*

 <ejs-accumulationchart (chartMouseMove)=‘chartMouseMove(\$event)’>
</ejs-accumulationchart>

 this.chartMouseLeave(args) { } |

| Fires on double click | **Property:** *chart.chartDoubleClick*

 <ej-chart (chartDoubleClick)=‘chartDoubleClick(\$event)’>
</ej-chart>

 this.chartDoubleClick(args) { } | **Property:** Not Applicable |

| Fires chart mouse up | **Property:** Not Applicable | **Property:** *chart.chartmouseUp*

 <ejs-accumulationchart (chartmouseUp)=‘chartmouseUp(\$event)’>
</ejs-accumulationchart>

 this.chartmouseUp(args) { } |

| Fires on chart mouse Down | **Property:** Not Applicable | **Property:** *chart.chartMouseDown*

 <ejs-accumulationchart (chartMouseDown)='chartMouseDown(\$event)'>
</ejs-accumulationchart>

 this.chartMouseDown(args) { } |

| Fires during calculation of area bounds | **Property:** *chart.chartAreaBoundsCalculate*

 <ej-chart (chartAreaBoundsCalculate)='chartAreaBoundsCalculate(\$event)'>
</ej-chart>

 this.chartAreaBoundsCalculate(args) { } | **Property:** Not Applicable |

| Fires after chart destroyed | **Property:** *chart.destroy*

 <ej-chart (destroy)='destroy(\$event)'>
</ej-chart>

 this.destroy(args) { } | **Property:** Not Applicable |

| Fires on chart created | **Property:** *chart.create*

 <ej-chart (create)='create(\$event)'>
</ej-chart>

 this.create(args) { } | **Property:** *chart.load*

 <ejs-accumulationchart (load)='load(\$event)'>
</ejs-accumulationchart>

 this.load(args) { } |

| Fires before rendering the data labels | **Property:** *chart.displayTextRendering*

 <ej-chart (displayTextRendering)='displayTextRendering(\$event)'>
</ej-chart>

 this.displayTextRendering(args) { } | **Property:** *chart.textRender*

 <ejs-accumulationchart (textRender)='textRender(\$event)'>
</ejs-accumulationchart>

 this.textRender(args) { } |

| Fires on clicking the legend item | **Property:** *chart.legendItemClick*

 <ej-chart (legendItemClick)='legendItemClick(\$event)'>
</ej-chart>

 this.legendItemClick(args) { } | **Property:** Not Applicable |

| Fires when moving mouse over legend item | **Property:** *chart.legendItemMouseMove*

 <ej-chart (legendItemMouseMove)='legendItemMouseMove(\$event)'>
</ej-chart>

 this.legendItemMouseMove(args) { } | **Property:** Not Applicable |

| Fires on legend item render | **Property:** *chart.legendItemRendering*

 <ej-chart (legendItemRendering)='legendItemRendering(\$event)'>
</ej-chart>

 this.legendItemRendering(args) { } | **Property:** Not Applicable |

| Fires on clicking a point in chart | **Property:** *chart.pointRegionClick*

 <ej-chart (pointRegionClick)='pointRegionClick(\$event)'>
</ej-chart>

 this.pointRegionClick(args) { } | **Property:** *chart.pointClick*

 <ejs-accumulationchart (pointClick)='pointClick(\$event)'>
</ejs-accumulationchart>

 this.pointClick(args) { } |

| Fires on pre render | **Property:** *chart.preRender*

 <ej-chart (preRender)='preRender(\$event)'>
</ej-chart>

 this.preRender(args) { } | **Property:** Not Applicable |

| Fires when point render | **Property:** Not Applicable. | **Property:** *chart.pointRender*

 <ejs-accumulationchart (pointRender)='pointRender(\$event)'>
</ejs-accumulationchart>

 this.pointRender(args) { } |

| Fires when mouse is moved over a point | **Property:** *chart.pointRegionMouseMove*

 <ej-chart (pointRegionMouseMove)='pointRegionMouseMove(\$event)'>
</ej-chart>

 this.pointRegionMouseMove(args) { } | **Property:** *chart.pointMouseMove*

 <ej-chart

```
(pointMouseMove)='pointMouseMove($event)'><br></ej-chart> <br> <br>
this.pointMouseMove(args) { } |
```

```
| Fires on series render| Property: chart.seriesRendering <br> <br> <ej-chart
(seriesRendering)='seriesRendering($event)'> <br> </ej-chart> <br> <br>
this.seriesRendering(args) { } | Property: chart.seriesRender <br> <br> <ejs-accumulationchart
(seriesRender)='seriesRender($event)'><br></ejs-accumulationchart> <br> <br>
this.seriesRender(args) { } |
```

```
| Fires on title render| Property: chart.titleRendering <br> <br> <ej-chart
(titleRendering)='titleRendering($event)'><br></ej-chart> <br> <br> this.titleRendering(args) { } |
Property: Not Applicable |
```

```
| Fires on sub title render| Property: chart.subTitleRendering <br> <br> <ej-chart
(subTitleRendering)='subTitleRendering($event)'><br></ej-chart> <br> <br>
this.subTitleRendering(args) { } | Property: Not Applicable |
```

```
| Fires before rendering the tooltip| Property: chart.tooltipInitilize <br> <br> <ej-chart
(tooltipInitilize)='tooltipInitilize($event)'> <br></ej-chart> <br> <br> this.tooltipInitilize(args) { } |
Property: chart.tooltipRender <br> <br> <ej-chart
(tooltipRender)='tooltipRender($event)'><br></ej-chart> <br> <br> this.tooltipRender(args) { } |
```

How To

Percentage tool tip in Angular Chart component

By using the [tooltipRender](#) event, you can show the percentage value for each point of pie series in tooltip.

To show the percentage value in pie tooltip, follow the given steps:

Step 1:

By using the [tooltipRender](#) event, you can get the `args.point.y` and `args.series.sumOfPoints` values. You can use these values to calculate the percentage value for each point of pie series. To display the percentage value in tooltip, use the `args.content` property.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ChartModule, ChartAllModule, AccumulationChartAllModule } from
'@syncfusion/ej2-angular-charts'
import { AccumulationChartModule } from '@syncfusion/ej2-angular-charts'
import { PieSeriesService, AccumulationTooltipService,
AccumulationDataLabelService } from '@syncfusion/ej2-angular-charts'
import {
    LineSeriesService, DateTimeService, DataLabelService,
    StackingColumnSeriesService, CategoryService,
    StepAreaSeriesService, SplineSeriesService, ScrollBarService,
    ChartAnnotationService, LegendService, TooltipService, StripLineService,
    SelectionService, ScatterSeriesService, ZoomService, ColumnSeriesService,
    AreaSeriesService, RangeAreaSeriesService
} from '@syncfusion/ej2-angular-charts'
import { Component, OnInit } from '@angular/core';
import { IAccTooltipRenderEventArgs } from '@syncfusion/ej2-angular-charts';
```

```

@Component({
  imports: [
    ChartModule, ChartAllModule, AccumulationChartAllModule,
    AccumulationChartModule
  ],
  providers: [LineSeriesService, DateTimeService, ColumnSeriesService,
    DataLabelService, ZoomService, StackingColumnSeriesService, CategoryService,
    StepAreaSeriesService, SplineSeriesService, ChartAnnotationService,
    LegendService, TooltipService, StripLineService,
    PieSeriesService, AccumulationTooltipService, ScrollBarService,
    AccumulationDataLabelService, SelectionService, ScatterSeriesService,
    AreaSeriesService, RangeAreaSeriesService ],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-accumulationchart id="chart-container"
[tooltip]='tooltip' [title]='title' (tooltipRender)='tooltipRender($event)'>
  <e-accumulation-series-collection>
    <e-accumulation-series [dataSource]='piedata' xName='x' yName='y'
[dataLabel]='datalabel' radius="70%"></e-accumulation-series>
  </e-accumulation-series-collection>
</ejs-accumulationchart>`
})
export class AppComponent implements OnInit {
  public piedata?: Object[];
  public datalabel?: Object;
  public tooltip?: Object;
  public title?: String;
  public tooltipRender(args: IAccTooltipRenderEventArgs): void {
    let value = args.point.y / args.series.sumOfPoints * 100;
    args["text"] = args.point.x + ' : ' + Math.ceil(value) + ' %';
  };
  ngOnInit(): void {
    this.datalabel = { visible: true };
    this.tooltip = { enable: true };
    this.title = 'Mobile Browser Statistics';
    this.piedata = [
      { 'x': 'Chrome', y: 37 }, { 'x': 'UC Browser', y: 17 },
      { 'x': 'iPhone', y: 19 }, { 'x': 'Others', y: 4 }, { 'x':
'Opera', y: 11 }
    ];
  }
}

```

MAIN.TS

```

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

```

Click data in Angular Chart component

By using the [pointClick](#) event, you can get the chart data of clicked area.

To show the clicked area data from pie, follow the given steps:

Step 1:

By using the [pointClick](#) event, you can get the `args.point.x` and `args.point.y` values.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ChartModule, ChartAllModule, AccumulationChartAllModule } from '@syncfusion/ej2-angular-charts'
import { AccumulationChartModule } from '@syncfusion/ej2-angular-charts'
import { PieSeriesService, AccumulationTooltipService, AccumulationDataLabelService } from '@syncfusion/ej2-angular-charts'
import {
    LineSeriesService, DateTimeService, DataLabelService,
    StackingColumnSeriesService, CategoryService,
    StepAreaSeriesService, SplineSeriesService, ScrollBarService,
    ChartAnnotationService, LegendService, TooltipService, StripLineService,
    SelectionService, ScatterSeriesService, ZoomService, ColumnSeriesService,
    AreaSeriesService, RangeAreaSeriesService
} from '@syncfusion/ej2-angular-charts'
import { Component, OnInit } from '@angular/core';
import { IAccTooltipRenderEventArgs, IPointEventArgs } from '@syncfusion/ej2-angular-charts';
@Component({
    imports: [
        ChartModule, ChartAllModule, AccumulationChartAllModule,
        AccumulationChartModule
    ],
    providers: [LineSeriesService, DateTimeService, ColumnSeriesService,
        DataLabelService, ZoomService, StackingColumnSeriesService, CategoryService,
        StepAreaSeriesService, SplineSeriesService, ChartAnnotationService,
        LegendService, TooltipService, StripLineService,
        PieSeriesService, AccumulationTooltipService, ScrollBarService,
        AccumulationDataLabelService, SelectionService, ScatterSeriesService,
        AreaSeriesService, RangeAreaSeriesService ],
    standalone: true,
    selector: 'app-container',
    template: `<ejs-accumulationchart id="chart-container"
[tooltip]='tooltip' [title]='title' (pointClick)='pointClick($event)'>
    <e-accumulation-series-collection>
        <e-accumulation-series [dataSource]='piedata' xName='x' yName='y'
[dataLabel]='datalabel' radius="70%"></e-accumulation-series>
    </e-accumulation-series-collection>
</ejs-accumulationchart>
<label id="lb1"></label> `
})
export class AppComponent implements OnInit {
    public piedata?: Object[];
    public datalabel?: Object;
    public tooltip?: Object;
    public title?: String;
    public pointClick(args: IPointEventArgs): void {
        (document.getElementById("lb1") as HTMLElement).innerText = "X : "+
args.point.x + "\nY : "+ args.point.y;
    };
    ngOnInit(): void {
```

```

    this.datalabel = { visible: true };
    this.tooltip = { enable: true };
    this.title = 'Mobile Browser Statistics';
    this.piedata = [
      { 'x': 'Chrome', y: 37, text: '37%' }, { 'x': 'UC
Browser', y: 17, text: '17%' },
      { 'x': 'iPhone', y: 19, text: '19%' },
      { 'x': 'Others', y: 4, text: '4%' }, { 'x': 'Opera', y:
11, text: '11%' },
      { 'x': 'Android', y: 12, text: '12%' }
    ];
  }
}

```

MAIN.TS

```

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

```

AppBar

Getting started with Angular AppBar component

This section explains how to create a simple AppBar, and demonstrate the basic usage of the AppBar module in an Angular environment.

Dependencies

The list of dependencies required to use the AppBar module in your application is given below:

```

`javascript
|-- @syncfusion/ej2-angular-navigations
|-- @syncfusion/ej2-angular-base
|-- @syncfusion/ej2-navigations
|-- @syncfusion/ej2-base
,

```

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 AppBar 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-navigations](#) package to the application.

```
`bash
```

```
npm install @syncfusion/ej2-angular-navigations --save
```

```
,
```

Angular compatibility compiled package(ngcc)

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

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

```
`bash
```

```
npm install @syncfusion/ej2-angular-navigations@ngcc --save
```

```
,
```

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

```
`bash
```

```
@syncfusion/ej2-angular-navigations:"20.2.38-ngcc"
```

```
,
```

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

Adding AppBar module

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

```
`javascript
```

```
import { NgModule } from "@angular/core";
import { BrowserModule } from "@angular/platform-browser";
// Imported Syncfusion AppBar module from navigations package.
import { AppBarModule } from "@syncfusion/ej2-angular-navigations";
import { AppComponent } from "../app.component";
@NgModule({
  imports: [BrowserModule, AppBarModule], // Registering EJ2 AppBar Module.
  declarations: [AppComponent],
  bootstrap: [AppComponent],
})
export class AppModule {}
```

Adding Syncfusion AppBar component

Modify the template in `app.component.ts` file with `ejs-appbar` to render the AppBar component.

```
`javascript
import { Component } from "@angular/core";
@Component({
  selector: "app-root",
  template: `<!-- To Render AppBar. -->
<div class="control-container">
<ejs-appbar colorMode="Primary">
<button #regularBtn ejs-button cssClass="e-inherit menu" iconCss="e-icons e-menu"></button>
<span class="regular">Angular AppBar</span>
<div class="e-appbar-spacer"></div>
<button ejs-button cssClass="e-inherit login">FREE TRIAL</button>
</ejs-appbar>
</div>`,
})
export class AppComponent {}
```

Adding CSS reference

Add AppBar component's styles as given below in `style.css`.

```
`css
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
```



```
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
```

```
,
```

Running the application

Run the application in the browser using the following command:

```
,
```

```
ng serve
```

```
,
```

The following example shows a basic **AppBar** component.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { AppBarModule } from '@syncfusion/ej2-angular-navigations'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component } from '@angular/core';
@Component({
  imports: [ AppBarModule, ButtonModule],
  standalone: true,
  selector: "app-root",
  template: `<!-- To Render AppBar. -->
    <div class="control-container">
      <ejs-appbar colorMode="Primary">
        <button #regularBtn ejs-button cssClass="e-inherit" iconCss="e-icons
e-menu"></button>
        <span class="regular" style="margin:0 5px">Angular AppBar</span>
        <div class="e-appbar-spacer"></div>
        <button ejs-button cssClass="e-inherit">FREE TRIAL</button>
      </ejs-appbar>
    </div>`,
})
export class AppComponent {}
```

MAIN.TS

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

Size and color in Angular Appbar component

Size

The size of the AppBar can be set using the [mode](#) property. The available types of the AppBar are as follows:

- Regular AppBar
- Prominent AppBar
- Dense AppBar

Regular AppBar

This mode is the default one in which the AppBar is displayed with the default height.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { AppBarModule } from '@syncfusion/ej2-angular-navigations'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component } from "@angular/core";
@Component({
  imports: [ AppBarModule, ButtonModule],
  standalone: true,
  selector: "app-root",
  template: `<!-- To Render AppBar. -->
    <div class="control-container">
      <ejs-appbar colorMode="Primary">
        <button #defaultButtonMenu ejs-button cssClass="e-inherit"
iconCss="e-icons e-menu"></button>
        <span class="regular">Regular AppBar</span>
        <div class="e-appbar-spacer"></div>
        <button #defaultButtonLogin ejs-button cssClass="e-inherit">FREE
TRIAL</button>
      </ejs-appbar>
    </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));
```

Prominent AppBar

This height mode can be set to the AppBar by setting **Prominent** to the property [mode](#). The prominent AppBar is displayed with a longer height and can be used for larger titles, images, or texts. It is also longer than the regular AppBar. In the following example, we have customized the prominent text using align-self and white-space CSS properties. You can change the prominent AppBar height if larger titles, images, or texts are used.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { AppBarModule } from '@syncfusion/ej2-angular-navigations'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component } from "@angular/core";
@Component({
  imports: [ AppBarModule, ButtonModule],
  standalone: true,
  selector: "app-root",
  template: `<!-- To Render AppBar. -->
    <div class="control-container">
```

```

    <ejs-appbar colorMode="Primary" mode="Prominent" cssClass="prominent-
    appbar">
      <button #defaultButtonMenu ejs-button cssClass="e-inherit"
    iconCss="e-icons e-menu"></button>
      <span class="prominent">AppBar Component with Prominent mode</span>
      <div class="e-appbar-spacer"></div>
      <button #defaultButtonLogin ejs-button cssClass="e-inherit">FREE
    TRIAL</button>
    </ejs-appbar>
  </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));

```

Dense AppBar

This height mode can be set to the AppBar by setting Dense to the property `mode`. Dense AppBar is displayed with shorter height which is denser to accommodate all the AppBar content.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { AppBarModule } from '@syncfusion/ej2-angular-navigations'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component } from '@angular/core';
@Component({
  imports: [AppBarModule, ButtonModule],
  standalone: true,
  selector: "app-root",
  template: `<!-- To Render AppBar. -->
    <div class="control-container">
      <ejs-appbar colorMode="Primary" mode="Dense">
        <button #defaultButtonMenu ejs-button cssClass="e-inherit"
    iconCss="e-icons e-menu"></button>
        <span class="dense">Dense AppBar</span>
        <div class="e-appbar-spacer"></div>
        <button #defaultButtonLogin ejs-button cssClass="e-inherit">FREE
    TRIAL</button>
      </ejs-appbar>
    </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));

```

Color

The background and font colors can be set using the [colorMode](#) property. The available types of background color for the AppBar are as follows:

- Light AppBar
- Dark AppBar
- Primary AppBar
- Inherit AppBar

Light AppBar

This color mode is the default one in which the AppBar can be displayed with a light background and its corresponding font color.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { AppBarModule } from '@syncfusion/ej2-angular-navigations'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component } from '@angular/core';
@Component({
  imports: [ AppBarModule, ButtonModule],
  standalone: true,
  selector: "app-root",
  template: `<!-- To Render AppBar. -->
    <div class="control-container">
      <ejs-appbar>
        <a href="https://www.syncfusion.com/angular-components"
target="_blank" rel="noopener" role="link" aria-label="Syncfusion angular
controls">
          <div class="syncfusion-logo"></div>
        </a>
        <div class="e-appbar-spacer"></div>
        <button #defaultButtonLogin ej2-button isPrimary=true>FREE
TRIAL</button>
      </ejs-appbar>
    </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));
```

Dark AppBar

This color mode can be set to the AppBar by setting **Dark** to the property [colorMode](#). A dark AppBar can be displayed with a dark background and its corresponding font color.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { AppBarModule } from '@syncfusion/ej2-angular-navigations'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component } from '@angular/core';
@Component({
  imports: [ AppBarModule, ButtonModule],
  standalone: true,
  selector: "app-root",
  template: `<!-- To Render AppBar. -->
    <div class="control-container">
      <ejs-appbar colorMode="Dark">
        <button #defaultButtonMenu ej2-button cssClass="e-inherit"
iconCss="e-icons e-menu"></button>
        <div class="e-appbar-spacer"></div>
        <button #defaultButtonLogin ej2-button cssClass="e-inherit">FREE
TRIAL</button>
      </ejs-appbar>
    </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));
```

Primary AppBar

This color mode can be set to the AppBar by setting **Primary** to the property [colorMode](#). The primary AppBar can be displayed with primary colors.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { AppBarModule } from '@syncfusion/ej2-angular-navigations'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component } from '@angular/core';
@Component({
  imports: [ AppBarModule, ButtonModule],
  standalone: true,
  selector: "app-root",
  template: `<!-- To Render AppBar. -->
    <div class="control-container">
      <ejs-appbar colorMode="Primary">
        <button #defaultButtonMenu ej2-button cssClass="e-inherit"
iconCss="e-icons e-menu"></button>
        <div class="e-appbar-spacer"></div>
        <button #defaultButtonLogin ej2-button cssClass="e-inherit">FREE
TRIAL</button>
      </ejs-appbar>
    </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));
```

Inherit AppBar

This color mode can be set to the AppBar by setting `Inherit` to the property `colorMode`. The AppBar inherits the background and font color from its parent element.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { AppBarModule } from '@syncfusion/ej2-angular-navigations'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component } from '@angular/core';
@Component({
  imports: [AppBarModule, ButtonModule],
  standalone: true,
  selector: 'app-root',
  template: `<!-- To Render AppBar. -->
    <div class="control-container">
      <ejs-appbar colorMode="Inherit">
        <a href="https://www.syncfusion.com/angular-components"
target="_blank" rel="noopener" role="link" aria-label="Syncfusion angular
controls">
          <div class="syncfusion-logo"></div>
        </a>
        <div class="e-appbar-spacer"></div>
        <button #defaultButtonLogin ej2-button isPrimary=true>FREE
TRIAL</button>
      </ejs-appbar>
    </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));
```

Accessibility in Angular AppBar component

The AppBar 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 AppBar component is outlined below.

| Accessibility Criteria | Compatibility |

```

| -- | -- |
| WCAG 2.2 Support |  |
| Section 508 Support |  |
| Screen Reader Support |  |
| Right-To-Left Support |  |
| Color Contrast |  |
| Mobile Device Support |  |
| Keyboard Navigation Support |  |
| Accessibility Checker Validation |  |
| Axe-core Accessibility Validation |  |
<style>
.post .post-content img {
display: inline-block;
margin: 0.5em 0;
}
</style>
<div> - All
features of the component meet the requirement.</div>
<div> - Some features of the component do not meet the requirement.</div>
<div> - The component does not meet the requirement.</div>

```

Keyboard interaction

The AppBar component provides the focus element navigation based on its's tab key order.

Ensuring accessibility

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

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

See also

- [Accessibility in Syncfusion Angular components](#)

Position in Angular AppBar component

The position of the AppBar can be set using the [position](#) and [isSticky](#) property. The AppBar provides the following options for setting its position:

- Top AppBar
- Bottom AppBar
- Sticky AppBar

Top AppBar

The top AppBar is the default one in which it positions the AppBar at the top of the content.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { AppBarModule } from '@syncfusion/ej2-angular-navigations'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component } from '@angular/core';
@Component({
  imports: [ AppBarModule, ButtonModule],
  standalone: true,
  selector: "app-root",
  template: `<!-- To Render AppBar. -->
    <div class="control-container">
      <ejs-appbar colorMode="Primary">
        <button #defaultButtonMenu ejs-button cssClass="e-inherit"
iconCss="e-icons e-menu"></button>
        <div class="e-appbar-spacer"></div>
        <button #defaultButtonLogin ejs-button cssClass="e-inherit">FREE
TRIAL</button>
      </ejs-appbar>
      <div class="appbar-content" style="font-size: 12px">
        <p>
```

The AppBar also known as action bar or nav bar displays information and actions related to the current application screen. It is used to show branding, screen titles, navigation, and actions. The control supports height mode, color mode, positioning, and more.

The AppBar control provides flexible ways to configure the look and feel of the bar to match your requirement.

Developers can control the appearance and behaviors of the AppBar using a rich set of APIs.

The AppBar component supports built-in themes such as Material, Bootstrap, Fabric (Office 365), Tailwind CSS, and high contrast. Users can customize these built-in themes or create new themes to achieve their desired


```

look and feel by either simply overriding SASS variables or using our Theme
Studio application.
    </p>
    </div>
  </div>`,
  })
  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));

```

Bottom AppBar

This position can be set to the AppBar by setting **Bottom** to the property [position](#). The bottom AppBar positions the AppBar at the bottom of the content.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { AppBarModule } from '@syncfusion/ej2-angular-navigations'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component } from "@angular/core";
@Component({
  imports: [ AppBarModule, ButtonModule],
  standalone: true,
  selector: "app-root",
  template: `<!-- To Render AppBar. -->
    <div class="control-container">
      <ejs-appbar colorMode="Primary" position="Bottom">
        <button #defaultButtonMenu ejs-button cssClass="e-inherit"
iconCss="e-icons e-menu"></button>
        <div class="e-appbar-spacer"></div>
        <button #defaultButtonLogin ejs-button cssClass="e-inherit">FREE
TRIAL</button>
      </ejs-appbar>
      <div class="appbar-content" style="font-size: 12px">
        <p>

```

The AppBar also known **as** action bar or nav bar displays information and actions related to the current application screen. It **is** used to show branding, screen titles, navigation, and actions. The control supports height mode, color mode, positioning, and more.

The AppBar control provides flexible ways to configure the look and feel of the bar to match your requirement.

Developers can control the appearance and behaviors of the AppBar **using a** rich **set** of APIs.

The AppBar component supports built-in themes such as Material, Bootstrap, Fabric (Office 365), Tailwind CSS, and high contrast. Users can customize these built-in themes or create new themes to achieve their desired look and feel by either simply overriding SASS variables or using our Theme Studio application.

```

    </p>
  </div>
</div>`,
  })
  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));

```

Sticky AppBar

This position can be set to the AppBar by setting `true` to the property `isSticky`. AppBar will be sticky while scrolling the AppBar content.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { AppBarModule } from '@syncfusion/ej2-angular-navigations'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component } from '@angular/core';
@Component({
  imports: [AppBarModule, ButtonModule],
  standalone: true,
  selector: 'app-root',
  template: `<!-- To Render AppBar. -->
    <div class="control-container">
      <ejs-appbar colorMode="Primary" isSticky=true>
        <button #defaultButtonMenu ejs-button cssClass="e-inherit"
iconCss="e-icons e-menu"></button>
        <div class="e-appbar-spacer"></div>
        <button #defaultButtonLogin ejs-button cssClass="e-inherit">FREE
TRIAL</button>
      </ejs-appbar>
      <div class="appbar-content" style="font-size: 12px">
        <p>

```

The AppBar also known as action bar or nav bar displays information and actions related to the current application screen. It is used to show branding, screen titles, navigation, and actions. The control supports height mode, color mode, positioning, and more.

```

    </p>
    <p>
      The AppBar control provides flexible ways to configure the look
      and feel of the bar to match your requirement.
    </p>
    <p>

```

Developers can control the appearance and behaviors of the AppBar using a rich set of APIs.

The AppBar component supports built-in themes such as Material, Bootstrap, Fabric (Office 365), Tailwind CSS, and high contrast. Users can customize these built-in themes or create new themes to achieve their desired look and feel by either simply overriding SASS variables or using our Theme Studio application.

```

    </p>
  </div>
</div>`,
}))
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));

```

Design in Angular AppBar component

Spacer

Spacer is used to provide spacing between the AppBar contents which gives additional space to the content layout.

The following example depicts the code to provide spacing between the home and pan buttons in the AppBar:

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { AppBarModule } from '@syncfusion/ej2-angular-navigations'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component } from '@angular/core';
@Component({
  imports: [AppBarModule, ButtonModule],
  standalone: true,
  selector: 'app-root',
  template: `<!-- To Render AppBar. -->
    <div class="control-container">
      <ejs-appbar colorMode="Primary">
        <button #defaultButtonHome ejs-button cssClass="e-inherit"
iconCss="e-icons e-home"></button>
        <div class="e-appbar-spacer"></div>
        <button #defaultButtonCut ejs-button cssClass="e-inherit" iconCss="e-
icons e-cut"></button>
        <div class="e-appbar-spacer"></div>
        <button #defaultButtonPan ejs-button cssClass="e-inherit" iconCss="e-
icons e-pan"></button>
      </ejs-appbar>
    </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

Separator shows a vertical line to visually group or separate the AppBar contents.

The following example depicts the code to provide a vertical line between a group of buttons in the AppBar.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { AppBarModule } from '@syncfusion/ej2-angular-navigations'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component } from '@angular/core';
@Component({
  imports: [ AppBarModule, ButtonModule],
  standalone: true,
  selector: "app-root",
  template: `<!-- To Render AppBar. -->
    <div class="control-container">
      <ejs-appbar colorMode="Primary">
        <button #defaultButtonCut ejs-button cssClass="e-inherit" iconCss="e-
icons e-cut"></button>
        <button #defaultButtonCopy ejs-button cssClass="e-inherit"
iconCss="e-icons e-copy"></button>
        <button #defaultButtonPaste ejs-button cssClass="e-inherit"
iconCss="e-icons e-paste"></button>
        <div class="e-appbar-separator"></div>
        <button #defaultButtonBold ejs-button cssClass="e-inherit"
iconCss="e-icons e-bold"></button>
        <button #defaultButtonUnderline ejs-button cssClass="e-inherit"
iconCss="e-icons e-underline"></button>
        <button #defaultButtonItalic ejs-button cssClass="e-inherit"
iconCss="e-icons e-italic"></button>
        <div class="e-appbar-separator"></div>
        <button #defaultButtonAlignLeft ejs-button cssClass="e-inherit"
iconCss="e-icons e-align-left"></button>
        <button #defaultButtonAlignRight ejs-button cssClass="e-inherit"
iconCss="e-icons e-align-right"></button>
        <button #defaultButtonAlignCenter ejs-button cssClass="e-inherit"
iconCss="e-icons e-align-center"></button>
        <button #defaultButtonJustify ejs-button cssClass="e-inherit"
iconCss="e-icons e-justify"></button>
      </ejs-appbar>
    </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));
```

Media Query

Media Query is used to adjusting the AppBar for different screen sizes. Resize the screen to observe the responsive layout of the AppBar.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { AppBarModule } from '@syncfusion/ej2-angular-navigations'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component } from '@angular/core';
@Component({
  imports: [ AppBarModule, ButtonModule],
  standalone: true,
  selector: "app-root",
  template: `<!-- To Render AppBar. -->
    <div class="control-container">
      <ejs-appbar colorMode="Primary">
        <button #defaultButtonMenu ejs-button cssClass="e-inherit"
iconCss="e-icons e-menu"></button>
        <button #defaultButtonHome ejs-button cssClass="e-
inherit">Home</button>
        <button #defaultButtonAbout ejs-button cssClass="e-
inherit">About</button>
        <button #defaultButtonProducts ejs-button cssClass="e-
inherit">Products</button>
        <button #defaultButtonContacts ejs-button cssClass="e-
inherit">Contacts</button>
        <div class="e-appbar-spacer"></div>
        <div class="e-appbar-separator"></div>
        <button #defaultButtonLogin ejs-button cssClass="e-
inherit">Login</button>
      </ejs-appbar>
    </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));
```

Designing AppBar with Menu

AppBar is rendered with a Menu component in its AppBar header area. Menu component's styles are inherited from the AppBar component using the `e-inherit` CSS class.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { AppBarModule, MenuModule } from '@syncfusion/ej2-angular-navigations'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component } from '@angular/core';
import { MenuItemModel } from '@syncfusion/ej2-angular-navigations';
@Component({
  imports: [ AppBarModule, ButtonModule, MenuModule],
  standalone: true,
  selector: "app-root",
  template: `<!-- To Render AppBar. -->
    <div class="control-container">
      <ejs-appbar colorMode="Primary">
        <button #defaultButtonMenu ejs-button cssClass="e-inherit"
iconCss="e-icons e-menu"></button>
        <ejs-menu #defaultMenuCompany cssClass="e-inherit"
[items]='companyMenuItems'></ejs-menu>
        <ejs-menu #defaultMenuProducts cssClass="e-inherit"
[items]='productMenuItems'></ejs-menu>
        <ejs-menu #defaultMenuAbout cssClass="e-inherit"
[items]='aboutMenuItems'></ejs-menu>
        <ejs-menu #defaultMenuCarrers cssClass="e-inherit"
[items]='carrerMenuItems'></ejs-menu>
        <div class="e-appbar-spacer"></div>
        <button #defaultButtonLogin ejs-button cssClass="e-
inherit">Login</button>
      </ejs-appbar>
    </div>`,
})
export class AppComponent {
  public companyMenuItems: MenuItemModel[] = [
    {
      text : 'Company',
      items: [
        { text: 'About Us' },
        { text: 'Customers' },
        { text: 'Blog' },
        { text: 'Careers' }
      ]
    }
  ];
  public productMenuItems: MenuItemModel[] = [
    {
      text : 'Products',
      items: [
        { text: 'Developer' },
        { text: 'Analytics' },
        { text: 'Reporting' },
        { text: 'Help Desk' }
      ]
    }
  ]
}
```

```

    }
  ];
  public aboutMenuItems: MenuItemModel[] = [
    {
      text : 'About Us'
    }
  ];
  public carrerMenuItems: MenuItemModel[] = [
    {
      text : 'Carrers'
    }
  ];
}

```

MAIN.TS

```

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

```

Designing AppBar with Buttons

The AppBar is rendered with a Button and DropDownButton component in its AppBar header area.

Button and DropDownButton components' styles are inherited from the AppBar component using the `e-inherit` CSS class.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { AppBarModule } from '@syncfusion/ej2-angular-navigations'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { DropDownButtonModule } from '@syncfusion/ej2-angular-splitbuttons'
import { Component } from '@angular/core';
import { ItemModel } from '@syncfusion/ej2-angular-splitbuttons';
@Component({
  imports: [AppBarModule, ButtonModule, DropDownButtonModule],
  standalone: true,
  selector: "app-root",
  template: `<!-- To Render AppBar. -->
    <div class="control-container">
      <ejs-appbar colorMode="Primary">
        <button #defaultButtonMenu ejs-button cssClass="e-inherit"
iconCss="e-icons e-menu"></button>
        <button #defaultDropDownButtonProduct ejs-dropdownbutton cssClass="e-
inherit" [items]='productDropDownButtonItem' content="Products"></button>
        <div class="e-appbar-spacer"></div>
        <button #defaultButtonLogin ejs-button cssClass="e-
inherit">Login</button>
      </ejs-appbar>
    </div>`,
})
export class AppComponent {
  public productDropDownButtonItem: ItemModel[] = [
    { text: 'Developer' },
  ],

```

```

    { text: 'Analytics' },
    { text: 'Reporting' },
    { text: 'E-Signature' },
    { text: 'Help Desk' }
  ];
}

```

MAIN.TS

```

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

```

Designing AppBar with SideBar

The AppBar is rendered with the SideBar component below the AppBar. Click on the menu icon to expand/collapse the Sidebar. In the following sample, the `toggle` method has been used to show or hide the Sidebar on the AppBar button click.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { AppBarModule, SidebarModule, TreeViewModule } from '@syncfusion/ej2-angular-navigations'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { TextBoxModule } from '@syncfusion/ej2-angular-inputs'
import { Component, ViewChild } from '@angular/core';
import { SidebarComponent } from '@syncfusion/ej2-angular-navigations';
@Component({
  imports: [ AppBarModule, ButtonModule, SidebarModule, TreeViewModule,
    TextBoxModule ],
  standalone: true,
  selector: 'app-root',
  template: `<!-- To Render AppBar. -->
    <div class="control-section" id="reswrapper">
      <div>
        <ejs-appbar>
          <button #defaultButtonMenu ej2-button cssClass="e-inherit"
            iconCss="e-icons e-menu" (click)="toggle()"></button>
          <div class="e-folder">
            <div class="e-folder-name">Navigation Pane</div>
          </div>
        </ejs-appbar>
      </div>
      <ejs-sidebar id="sideTree" class="sidebar-treeview"
        #sidebarTreeviewInstance [width]="width" [target]="target"
        [mediaQuery]="mediaQuery" [isOpen]="true">
        <div class='main-menu'>
          <div class="table-content">
            <ejs-textbox id="resSearch"
              placeholder="Search..."></ejs-textbox>
            <p class="main-menu-header">TABLE OF CONTENTS</p>
          </div>
        </div>
      </div>
    </div>
  `;
})

```



```

        <ejs-treeview id='mainTree' cssClass="main-treeview"
[fields]="fields" expandOn='Click'>
            </ejs-treeview>
        </div>
    </div>
</ejs-sidebar>
<div class="main-sidebar-content" id="main-text">
    <div class="sidebar-content">
        <div class="sidebar-heading"> Responsive Sidebar with
Treeview</div>
        <p class="paragraph-content">
            This is a graphical aid for visualising and categorising
the site, in the style of an expandable and
            collapsable treeview component.
            It auto-expands to display the node(s), if any,
corresponding to the currently viewed title,
            highlighting that node(s)
            and its ancestors. Load-on-demand when expanding nodes is
available where supported (most graphical
            browsers),
            falling back to a full-page reload. MediaWiki-supported
            caching, aside from squid, has been considered
            so that
            unnecessary re-downloads of content are avoided where
possible. The complete expanded/collapsed state of
            the treeview persists across page views in most
situations.
        </p>
    </div>
</div>
</div>`,
    })
export class AppComponent {
    @ViewChild('sidebarTreeviewInstance')
    public sidebarTreeviewInstance?: SidebarComponent;
    public data: { [key: string]: Object }[] = [
        {
            nodeId: '01', nodeText: 'Installation',
        },
        {
            nodeId: '02', nodeText: 'Deployment',
        },
        {
            nodeId: '03', nodeText: 'Quick Start',
        },
        {
            nodeId: '04', nodeText: 'Components',
            nodeChild: [
                { nodeId: '04-01', nodeText: 'Calendar' },
                { nodeId: '04-02', nodeText: 'DatePicker' },
                { nodeId: '04-03', nodeText: 'DateTimePicker' },
                { nodeId: '04-04', nodeText: 'DateRangePicker' },
                { nodeId: '04-05', nodeText: 'TimePicker' },
                { nodeId: '04-06', nodeText: 'SideBar' }
            ]
        }
    ]
}
];

```

```

public width: string = '220px';
public target: string = '.main-sidebar-content';
public mediaQuery: string = '(min-width: 600px)';
public fields: object = { dataSource: this.data, id: 'nodeId', text:
'nodeText', child: 'nodeChild' };
toggle() {
  (this.sidebarTreeviewInstance as SidebarComponent).toggle();
};
}

```

MAIN.TS

```

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

```

Style and appearance in Angular AppBar component

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

|CSS Class | Purpose of Class |

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

|.e-appbar|To customize the appbar.|

|.e-appbar.e-prominent|To customize the prominent appbar.|

|.e-appbar.e-dense|To customize the dense appbar.|

|.e-appbar.e-light|To customize the light appbar.|

|.e-appbar.e-dark|To customize the dark appbar.|

|.e-appbar.e-primary|To customize the dark appbar.|

|.e-appbar.e-inherit|To customize the inherit appbar.|

Note: You can change the prominent AppBar height if larger titles, images, or texts are used.

CssClass

CssClass is used for AppBar customization based on the custom class. In the example below, the AppBar background and color are customized using the [cssClass](#) property.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { AppBarModule } from '@syncfusion/ej2-angular-navigations'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component } from '@angular/core';
@Component({
  imports: [AppBarModule, ButtonModule],
  standalone: true,
  selector: 'app-root',
  template: `<!-- To Render AppBar. -->

```

```

    <div class="control-container">
      <ejs-appbar colorMode="Primary" cssClass="custom-appbar">
        <button #defaultButtonMenu ejs-button cssClass="e-inherit"
iconCss="e-icons e-home"></button>
      </ejs-appbar>
    </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));

```

HtmlAttributes

It can be used for additional inline attributes by specifying as inline attributes or by specifying htmlAttributes directive. In the code example below, the aria-label of the AppBar is customized by specifying as attributes.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { AppBarModule } from '@syncfusion/ej2-angular-navigations'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component } from '@angular/core';
@Component({
  imports: [ AppBarModule, ButtonModule],
  standalone: true,
  selector: "app-root",
  template: `<!-- To Render AppBar. -->
    <div class="control-container">
      <ejs-appbar colorMode="Primary" aria-label="appbar">
        <button #defaultButtonMenu ejs-button cssClass="e-inherit"
iconCss="e-icons e-home"></button>
      </ejs-appbar>
    </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));

```

AutoComplete

Getting started with Angular Auto complete component

This section explains how to create a simple **AutoComplete** component and configure its available functionalities in Angular.

Dependencies

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

```
`javascript
|-- @syncfusion/ej2-angular-dropdowns
|-- @syncfusion/ej2-base
|-- @syncfusion/ej2-data
|-- @syncfusion/ej2-lists
|-- @syncfusion/ej2-inputs
|-- @syncfusion/ej2-navigations
|-- @syncfusion/ej2-popups
|-- @syncfusion/ej2-buttons
|-- @syncfusion/ej2-angular-base
`,`
```

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-autocomplete
`,`
```

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-autocomplete --style=scss
`,`
```

Navigate to the created project folder.

```
`bash
```

```
cd syncfusion-angular-autocomplete
```

```
,
```

Installing Syncfusion AutoComplete 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 AutoComplete module

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

```
`javascript
```

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

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

```
// import the AutoCompleteModule for the AutoComplete component
import { AutoCompleteModule } from '@syncfusion/ej2-angular-dropdowns';
import { AppComponent } from './app.component';
@NgModule({
  //declaration of ej2-angular-dropdowns module into NgModule
  imports: [ BrowserModule, AutoCompleteModule ],
  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 AutoComplete component

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

Add the Angular Autocomplete by using `<ejs-autocomplete>` 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 AutoComplete component
  template: <ejs-autocomplete id='atcelement'></ejs-autocomplete>
})
```

```
export class AppComponent { }
```

```
,
```

Binding data source

After initializing, populate the data in AutoComplete using [dataSource](#) property. Here, an array of string values is passed to the AutoComplete component.

```
`typescript
```

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

```
@Component({
```

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

```
  // specifies the template string for the AutoComplete component
```

```
  template: <ejs-autocomplete id='atcelement' [dataSource]='sportsData'></ejs-autocomplete>
```

```
})
```

```
export class AppComponent {
```

```
  constructor() {
```

```
  }
```

```
  // defined the array of data
```

```
  public sportsData: string[] = ['Badminton', 'Basketball', 'Cricket', 'Football', 'Golf', 'Gymnastics', 'Hockey', 'Rugby', 'Snooker', 'Tennis'];
```

```
}
```

```
,
```

Running the application

After completing the configuration required to render a basic AutoComplete, 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 { AutoCompleteModule } from '@syncfusion/ej2-angular-dropdowns'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component } from '@angular/core';
@Component({
  imports: [
    FormsModule, ReactiveFormsModule, AutoCompleteModule, ButtonModule
  ],
  standalone: true,
```

```

    selector: 'app-root',
    // specifies the template string for the AutoComplete component with
    dataSource
    template: `<ejs-autocomplete id='atcelement'
[dataSource]='sportsData'></ejs-autocomplete>`
  })
  export class AppComponent {
    constructor() {
    }
    // defined the array of data
    public sportsData: string[] = ['Badminton', 'Basketball', 'Cricket',
'Football', 'Golf', 'Gymnastics'];
  }

```

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](#) properties respectively.

In the following sample, popup list's width and height are configured.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule, ReactiveFormsModule } from '@angular/forms'
import { AutoCompleteModule } from '@syncfusion/ej2-angular-dropdowns'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component } from '@angular/core';
@Component({
  imports: [
    FormsModule, ReactiveFormsModule, AutoCompleteModule, ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the AutoComplete component with
  dataSource
  template: `<ejs-autocomplete id='atcelement' [dataSource]='sportsData'
[popupHeight]='height' [popupWidth]='width' [placeholder]='text'></ejs-
autocomplete>`
})
export class AppComponent {
  constructor() {
  }
  // defined the array of data

```



```

    public sportsData: string[] = ['Badminton', 'Cricket', 'Football',
    'Golf', 'Hockey', 'Rugby', 'Snooker', 'Tennis'];
    // set width of the popup list
    public width: string = '250px';
    // set height of the popup list
    public height: string = '200px';
    // set placeholder to autocomplete element
    public text: string = "Find 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));

```

Two-way binding

In AutoComplete, the `value` property supports two-way binding functionality.

The following example demonstrates how to work with the two-way binding functionality in AutoComplete.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule, ReactiveFormsModule } from '@angular/forms'
import { AutoCompleteModule } from '@syncfusion/ej2-angular-dropdowns'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component } from '@angular/core';
@Component({
  imports: [
    FormsModule, ReactiveFormsModule, AutoCompleteModule, ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the AutoComplete component and
  // input element for checking the two-way binding support using value
  // property
  template: `<ejs-autocomplete id='atcelement' [dataSource]='sportsData'
  [(value)]='value'></ejs-autocomplete>
  <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 sportsData: string[] = ['Badminton', 'Basketball', 'Cricket',
  'Football', 'Golf', 'Gymnastics'];
  // 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 Auto complete component

The AutoComplete loads the data either from local data sources or remote data services using the [dataSource](#) property. It supports the data type of array or [DataManager](#).

The AutoComplete also supports different kind of data services such as OData, OData V4, Web API and data formats such as XML, JSON, JSONP with the help of DataManager Adaptors.

| Fields | Type | Description |

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

| value | [number or string](#) | Specifies the hidden data value mapped to each list item that should contain a unique value. |

| groupBy | [string](#) | Specifies the category under which the list item has to be grouped. |

| iconCss | [string](#) | Specifies the icon class of each list item. |

While binding complex data to AutoComplete, fields should be mapped correctly. Otherwise, the selected item remains undefined.

Bind to local data

Local data can be represented in two ways as described below.

Array of string

The AutoComplete has support to load array of primitive data such as strings and numbers.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule, ReactiveFormsModule } from '@angular/forms'
import { AutoCompleteModule } from '@syncfusion/ej2-angular-dropdowns'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component } from '@angular/core';
@Component({
  imports: [
    FormsModule, ReactiveFormsModule, AutoCompleteModule, ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the AutoComplete component
```

```

    template: `<ejs-autocomplete id='atcelement' [dataSource]='sportsData'
    [placeholder]='placeholder'></ejs-autocomplete>`
  })
  export class AppComponent {
    constructor() {
    }
    // defined the array of data
    public sportsData: string[] = ['Badminton', 'Basketball', 'Cricket',
    'Football', 'Golf', 'Gymnastics', 'Hockey', 'Tennis'];
    // set placeholder text to AutoComplete input element
    public placeholder: string = 'Find 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));

```

Array of object

The AutoComplete can generate its list items through an array of complex data. For this, the appropriate columns should be mapped to the [fields](#) property.

In the following example, **Game** column from complex data have been mapped to the **value** field.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule, ReactiveFormsModule } from '@angular/forms'
import { AutoCompleteModule } from '@syncfusion/ej2-angular-dropdowns'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component } from '@angular/core';
@Component({
  imports: [
    FormsModule, ReactiveFormsModule, AutoCompleteModule, ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the AutoComplete component
  template: `<ejs-autocomplete id='atcelement' [dataSource]='sportsData'
  [fields]='fields' [placeholder]='text'></ejs-autocomplete>`
})
export class AppComponent {
  constructor() {
  }
  // defined the array of data
  public sportsData: { [key: string]: Object }[] = [
    { Id: 'Game1', Game: 'Badminton' },
    { Id: 'Game2', Game: 'Basketball' },
    { Id: 'Game3', Game: 'Cricket' },
    { Id: 'Game4', Game: 'Football' },
    { Id: 'Game5', Game: 'Golf' },
    { Id: 'Game6', Game: 'Hockey' },
    { Id: 'Game7', Game: 'Rugby' },
  ]
}

```

```

    { Id: 'Game8', Game: 'Snooker' }
  ];
  // maps the appropriate column to fields property
  public fields: Object = { value: 'Game' };
  //set the placeholder to AutoComplete input
  public text: string = "Find 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));

```

Array of complex object

The AutoComplete can generate its list items through an array of complex data. For this, the appropriate columns should be mapped to the [fields](#) property.

In the following example, **Country.Name** column from complex data have been mapped to the **value** field.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule, ReactiveFormsModule } from '@angular/forms'
import { AutoCompleteModule } from '@syncfusion/ej2-angular-dropdowns'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component } from '@angular/core';
@Component({
  imports: [
    FormsModule, ReactiveFormsModule, AutoCompleteModule, ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the AutoComplete component
  template: `<ejs-autocomplete id='atcelement' [dataSource]='countriesData'
[fields]='fields' [placeholder]='text'></ejs-autocomplete>`
})
export class AppComponent {
  constructor() {
  }
  // defined the array of data
  public countriesData: { [key: string]: Object }[] = [
    { Country: { Name: 'Australia' }, Code: { Id: 'AU' } },
    { Country: { Name: 'Bermuda' }, Code: { Id: 'BM' } },
    { Country: { Name: 'Canada' }, Code: { Id: 'CA' } },
    { Country: { Name: 'Cameroon' }, Code: { Id: 'CM' } },
    { Country: { Name: 'Denmark' }, Code: { Id: 'DK' } },
    { Country: { Name: 'France' }, Code: { Id: 'FR' } },
    { Country: { Name: 'Finland' }, Code: { Id: 'FI' } },
    { Country: { Name: 'Germany' }, Code: { Id: 'DE' } },
    { Country: { Name: 'Greenland' }, Code: { Id: 'GL' } },
    { Country: { Name: 'Hong Kong' }, Code: { Id: 'HK' } },
  ]
}

```

```

    { Country:{Name: 'India'}, Code:{ Id: 'IN'} },
    { Country:{ Name: 'Italy'}, Code: { Id: 'IT'} },
    { Country:{ Name: 'Japan'}, Code: { Id: 'JP'} },
    { Country:{Name: 'Mexico'}, Code: { Id: 'MX'} },
    { Country:{Name: 'Norway'}, Code: { Id: 'NO'} },
    { Country:{Name: 'Poland'}, Code: { Id: 'PL'} },
    { Country:{Name: 'Switzerland'}, Code: { Id: 'CH'} },
    { Country:{Name: 'United Kingdom'},Code: { Id: 'GB'} },
    { Country:{Name: 'United States'}, Code: { Id: 'US'} }
  ];
  // maps the appropriate column to fields property
  public fields: Object = { value: 'Country.Name' };
  //set the placeholder to AutoComplete input
  public text: string = "Find 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));

```

Bind to remote data

The AutoComplete supports retrieval of data from remote data services with the help of **DataManager** component. The [Query](#) property is used to fetch data from the database and bind it to the AutoComplete.

The following sample displays the first 6 contacts from the **Customers** table of the **Northwind** data service.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule, ReactiveFormsModule } from '@angular/forms'
import { AutoCompleteModule } from '@syncfusion/ej2-angular-dropdowns'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component } from '@angular/core';
//import DataManager related classes
import { Query, DataManager, ODataV4Adaptor } from '@syncfusion/ej2-data';
@Component({
  imports: [
    FormsModule, ReactiveFormsModule, AutoCompleteModule, ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the AutoComplete component
  template: `<ejs-autocomplete id='atcelement' [dataSource]='data'
    [fields]='fields' [placeholder]='text' [sortOrder]='sorting'
    [query]='query'></ejs-autocomplete>`
})
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/Customers',
    adaptor: new ODataV4Adaptor,
    crossDomain: true
});
// maps the appropriate column to fields property
public fields: Object = { value: 'ContactName' };
//bind the Query instance to query property
public query: Query = new Query().select(['ContactName']).take(6);
//set the placeholder to AutoComplete input
public text: string = "Find 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.

AutoComplete data can be consumed from an **Observable** object by piping it through an **async** pipe. The **async** pipe is used to subscribe the observable object and resolve with the latest value emitted by it.

[app.component.ts]

`ts

```

import { Component } from '@angular/core';
import { Observable } from 'rxjs';
import { map } from 'rxjs/operators';
import { HttpClient } from '@angular/common/http';
@Component({
    selector: 'app-root',
    // specifies the template string for the AutoComplete component with dataSource
    template: <ejs-autocomplete id='customers2' #remote2 [dataSource]='data | async'
    [fields]='remoteFields' [placeholder]='remoteWaterMark' ></ejs-autocomplete >
})
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 AutoComplete input element
public remoteWaterMark: string = 'Select a customer';
}
、

```

[app.module.ts]

`ts

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { HttpClientModule } from '@angular/common/http';
import { AutoCompleteModule} from '@syncfusion/ej2-angular-dropdowns';
import { AppComponent } from './app.component';
@NgModule({
imports: [
BrowserModule,
AutoCompleteModule,
HttpClientModule
],
declarations: [ AppComponent ],
bootstrap:  [ AppComponent ]
})
export class AppModule { }
、

```

[main.ts]

```
`ts
```

```
import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';
import { enableProdMode } from '@angular/core';
import { AppModule } from './app.module';
enableProdMode();
platformBrowserDynamic().bootstrapModule(AppModule);
`
```

[View Samples in Github](#)

See Also

- [How to load data using template](#)
- [How to group the data using header](#)
- [How to filter the bound data](#)

Value binding in AutoComplete Component

Value binding in the AutoComplete control allows you to associate data values with each list item. This facilitates managing and retrieving selected values efficiently. The AutoComplete component provides flexibility in binding both primitive data types and complex objects.

Primitive Data Types

The AutoComplete 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 AutoComplete.

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 { AutoCompleteModule } from '@syncfusion/ej2-angular-dropdowns'
import { Component } from '@angular/core';
import { AutoCompleteComponent, VirtualScroll } from '@syncfusion/ej2-angular-dropdowns';
AutoCompleteComponent.Inject(VirtualScroll);
@Component({
  imports: [
    FormsModule, AutoCompleteModule
  ],
```



```

standalone: true,
  selector: 'app-root',
  // specifies the virtual-scroll url path
  templateUrl: 'virtual-scroll.html'
})
export class AppComponent {
  // defined the array of data
  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 = { value: 'text' };
  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 AutoComplete control, object binding allows you to bind to a dataset of objects. When [allowObjectBinding](#) is enabled, the value of the control will be an object of the same type as the selected item in the [value](#) property. This feature seamlessly binds arrays of objects, whether sourced locally, retrieved from remote endpoints, or customized to suit specific application needs.

The following sample shows the example for preselect values for object data type

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule } from '@angular/forms'
import { AutoCompleteModule } from '@syncfusion/ej2-angular-dropdowns'
import { Component } from '@angular/core';
import { AutoCompleteComponent, VirtualScroll } from '@syncfusion/ej2-angular-dropdowns';
AutoCompleteComponent.Inject(VirtualScroll);
@Component({
  imports: [
    FormsModule, AutoCompleteModule
  ],
  standalone: true,
  selector: 'app-root',
  // specifies the virtual-scroll url path
  templateUrl: 'virtual-scroll.html'
})
export class AppComponent {
  // defined the array of data

```

```

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 = { value: 'text' };
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 Auto complete component

The AutoComplete has been provided with several options to customize each list items, group title, header and footer elements.

Item template

The content of each list item within the AutoComplete can be customized with the help of [itemTemplate](#) property.

In the following sample, each list item is split into two columns to display relevant data's.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule } from '@angular/forms'
import { AutoCompleteModule } from '@syncfusion/ej2-angular-dropdowns'
import { Component } from '@angular/core';
import { Query, DataManager, ODataV4Adaptor } from '@syncfusion/ej2-data'
@Component({
    imports: [
        FormsModule, AutoCompleteModule
    ],
    standalone: true,
    selector: 'app-root',
    // specifies the template url path
    templateUrl: 'itemTemplate.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 = { value: 'FirstName' };
//bind the Query instance to query property
public query: Query = new Query().from('Employees').select(['FirstName',
'City', 'EmployeeID']).take(6);
//set the placeholder to AutoComplete input
public text: string = "Find 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));

```

ITEMTEMPLATE.HTML

```

<div id="wrapper" style='margin-top: 20px'>
  <div id='content' style="margin: 0px auto; width:300px;">
    <!-- specifies the template string for the AutoComplete component-->
    <ejs-autocomplete id='atcelelement' [dataSource]='data'
[sortOrder]='sorting' [fields]='fields' [query]='query' [placeholder]='text'>
      <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-autocomplete>
  </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 { AutoCompleteModule } from '@syncfusion/ej2-angular-dropdowns'
import { Component } from '@angular/core';
//import data manager related classes

```

```

import { Query, Predicate, DataManager, ODataV4Adaptor } from
'@syncfusion/ej2-data';
@Component({
imports: [
    FormsModule, AutoCompleteModule
],
standalone: true,
selector: 'app-root',
// specifies the template url path
templateUrl: 'groupTemplate.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 = { value: 'FirstName', 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 AutoComplete input
    public text: string = "Find 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));

```

GROUPTEMPLATE.HTML

```

<div id="wrapper" style='margin-top: 20px'>
    <div id='content' style="margin: 0px auto; width:250px;">
        <!-- specifies the template string for the AutoComplete component-->
        <ejs-autocomplete id='atcelement' [dataSource]='data'
[fields]='fields' [sortOrder]='sorting' [placeholder]='text' [query]='query'>
            <ng-template #groupTemplate="" let-data="">
                <!--set the value to groupTemplate property-->
                <strong>{{data.City}}</strong>
            </ng-template>
        </ejs-autocomplete>
    </div>
</div>

```

Header template

The header element is shown statically at the top of the suggestion list items within the AutoComplete, and any custom element can be placed as a header element using 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 { AutoCompleteModule } from '@syncfusion/ej2-angular-dropdowns'
import { Component } from '@angular/core';
import { Query, DataManager, ODataV4Adaptor } from '@syncfusion/ej2-data';
@Component({
  imports: [
    FormsModule, AutoCompleteModule
  ],
  standalone: true,
  selector: 'app-root',
  // specifies the template url path
  templateUrl: 'headerTemplate.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 = { value: 'FirstName' };
  //bind the Query instance to query property
  public query: Query = new Query().from('Employees').select(['FirstName',
    'City', 'EmployeeID']).take(6);
  //set the placeholder to AutoComplete input
  public text: string = "Find 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));
```

HEADERTEMPLATE.HTML

```
<div id="wrapper" style='margin-top: 20px'>
```

```

<div id='content' style='margin: 0px auto; width:250px;'>
  <!-- specifies the template string for the AutoComplete component-->
  <ejs-autocomplete id='atcelement' [dataSource]='data'
[fields]='fields' [placeholder]='text' [sortOrder]='sorting' [query]='query'>
    <ng-template #itemTemplate="" let-data="">
      <!--set the value to itemTemplate property-->
      <span class='item'><span class='name'>
{{data.FirstName}}</span><span class ='city'>{{data.City}}</span></span>
    </ng-template>
    <ng-template #headerTemplate="" let-data="">
      <!--set the value to headerTemplate property-->
      <span class='head'><span class='name'>Name</span><span
class='city'>City</span></span>
    </ng-template>
  </ejs-autocomplete>
</div>
</div>

```

Footer template

The AutoComplete has options to show a footer element at the bottom of the list items in the suggestion list. Here, you can place any custom element as a footer element using [footerTemplate](#) property.

In the following sample, footer element displays the total number of list items present in the AutoComplete.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule } from '@angular/forms'
import { AutoCompleteModule } from '@syncfusion/ej2-angular-dropdowns'
import { Component, ViewChild } from '@angular/core';
import { AutoCompleteComponent, DropEventArgs } from '@syncfusion/ej2-
angular-dropdowns';
@Component({
  imports: [
    FormsModule,AutoCompleteModule
  ],
  standalone: true,
  selector: 'app-root',
  // specifies the template url path
  templateUrl: `footerTemplate.html`
})
export class AppComponent {
  @ViewChild('sample')
  public AutoCompleteObj : AutoCompleteComponent | any;
  constructor() {
  }
  // defined the array of data
  public data: Object[] = ['Badminton', 'Basketball', 'Cricket',
'Football', 'Golf', 'Gymnastics', 'Hockey'];
  // set placeholder text to AutoComplete input element
  public text: string = 'Find a game';
  public onOpen(e : DropEventArgs) : void{
    let count=this.AutoCompleteObj.getItems().length;

```

```
//set the value to footerTemplate property
let ele = document.getElementsByClassName('foot')[0];
ele.innerHTML = "Total list item: " + count;
}
}
```

MAIN.TS

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

FOOTERTEMPLATE.HTML

```
<div id="wrapper" style='margin-top: 20px'>
  <div id='content' style="margin: 0px auto; width:250px;">
    <!-- specifies the template string for the AutoComplete component-->
    <ejs-autocomplete id='atcelement' #sample [dataSource]='data'
[placeholder]='text' (open)='onOpen($event)'>
      <!--set the footerTemplate property-->
      <ng-template #footerTemplate="" let-data="">
        <span class='foot'> </span>
      </ng-template>
    </ejs-autocomplete>
  </div>
</div>
```

No records template

The AutoComplete is provided with support to custom design the popup list content when no data is found and no matches found on search with the help of [noRecordsTemplate](#) property.

In the following sample, popup list content displays the notification of no data available.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule } from '@angular/forms'
import { AutoCompleteModule } from '@syncfusion/ej2-angular-dropdowns'
import { Component } from '@angular/core';
@Component({
  imports: [
    FormsModule,AutoCompleteModule
  ],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the AutoComplete component
  template: `<ejs-autocomplete id='atcelement' [dataSource]='data'
placeholder='Find a item'>
    <ng-template #noRecordsTemplate>
      <span class='norecord'> NO DATA AVAILABLE</span>
    </ng-template>
  </ejs-autocomplete>`
})
```

```
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 AutoComplete displays the notification.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule } from '@angular/forms'
import { AutoCompleteModule } 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, AutoCompleteModule
  ],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the AutoComplete component
  template: `<ejs-autocomplete id='atcelement' [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-autocomplete>`,
})
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 = { value: 'FirstName' };
}
```

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](#)

Virtualization in AutoComplete Component

AutoComplete virtualization is a technique used to efficiently render extensive lists of items while minimizing the impact on performance. This method is particularly advantageous when dealing with large datasets because it ensures that only a fixed number of DOM (Document Object Model) elements are created. When scrolling through the list, existing DOM elements are reused to display relevant data instead of generating new elements for each item. This recycling process is managed internally.

During virtual scrolling, the data retrieved from the data source depends on the popup height and the calculation of the list item height. Enabling the [enableVirtualization](#) option in a AutoComplete activates this virtualization technique.

When fetching data from the data source, the [actionBegin](#) event is triggered before data retrieval begins. Then, the [actionComplete](#) event is triggered once the data is successfully fetched.

When the [enableVirtualization](#) property is enabled, the [skip](#) and [take](#) properties provided by the user within the Query class at the initial state or during the [actionBegin](#) or [actionComplete](#) events will not be considered, since it is internally managed and calculated based on certain dimensions with respect to the popup height.

Binding local data

The AutoComplete can generate its list items through an array of complex data. For this, the appropriate columns should be mapped to the [fields](#) property. When using virtual scrolling, the list updates based on the scroll offset value, triggering a request to fetch more data from the server. As the data is being fetched, the [actionBegin](#) event occurs before the data retrieval starts. Once the data retrieval is successful, the [actionComplete](#) event is triggered, indicating that the data fetch process is complete.

In the following example, [text](#) column from complex data have been mapped to the [value](#) field.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule } from '@angular/forms'
import { AutoCompleteModule } from '@syncfusion/ej2-angular-dropdowns'
import { Component } from '@angular/core';
```

```
import { AutoCompleteComponent, VirtualScroll } from '@syncfusion/ej2-angular-dropdowns';
AutoCompleteComponent.Inject(VirtualScroll);
@Component({
  imports: [
    FormsModule, AutoCompleteModule
  ],
  standalone: true,
  selector: 'app-root',
  // specifies the virtual-scroll url path
  templateUrl: 'virtual-scroll.html'
})
export class AppComponent {
  // defined the array of data
  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 = { value: 'text' };
  // 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 AutoComplete component-->
    <ejs-autocomplete id='atcelement' [dataSource]='records'
[fields]='fields' [placeholder]='waterMark' [enableVirtualization]='true'
popupHeight='200px'>
      </ejs-autocomplete>
    </div>
  </div>
```

Binding remote data

The AutoComplete supports retrieval of data from remote data services with the help of **DataManager** component. When using remote data, it initially fetches all the data from the server, triggering the **actionBegin** and **actionComplete** events, and then stores the data locally. During virtual scrolling,

additional data is retrieved from the locally stored data, triggering the `actionBegin` and `actionComplete` events at that time as well.

The following sample displays the `OrderId` from the `Orders` Data Service.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule } from '@angular/forms'
import { AutoCompleteModule } from '@syncfusion/ej2-angular-dropdowns'
import { Component } from '@angular/core';
import { AutoCompleteComponent, VirtualScroll } from '@syncfusion/ej2-angular-dropdowns';
import { Query, DataManager, WebApiAdaptor } from '@syncfusion/ej2-data';
AutoCompleteComponent.Inject(VirtualScroll);
@Component({
  imports: [
    FormsModule, AutoCompleteModule
  ],
  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 } = { 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 AutoComplete component-->
    <ejs-autocomplete id='atcelement' [dataSource]='customerData'
[fields]='customerField' [placeholder]='waterMark'
[enableVirtualization]='true' popupHeight='200px'>
      </ejs-autocomplete>
    </div>
  </div>
```

Grouping

The AutoComplete component supports grouping with Virtualization. It allows you to organize elements into groups based on different categories. Each item in the list can be classified using the [groupBy](#) field in the data table. After grouping, virtualization works similarly to local data binding, providing a seamless user experience. When the data source is bound to remote data, an initial request is made to retrieve all data for the purpose of grouping. Subsequently, the grouped data works in the same way as local data binding virtualization, enhancing performance and responsiveness.

The following sample shows the example for Grouping with Virtualization

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule } from '@angular/forms'
import { AutoCompleteModule } from '@syncfusion/ej2-angular-dropdowns'
import { Component } from '@angular/core';
import { AutoCompleteComponent, VirtualScroll } from '@syncfusion/ej2-angular-dropdowns';
AutoCompleteComponent.Inject(VirtualScroll);
@Component({
  imports: [
    FormsModule, AutoCompleteModule
  ],
  standalone: true,
  selector: 'app-root',
  // specifies the virtual-scroll url path
  templateUrl: 'virtual-scroll.html'
})
export class AppComponent {
  // defined the array of data
  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 AutoComplete component-->
    <ejs-autocomplete id='atcelement' [dataSource]='records'
[fields]='fields' [placeholder]='waterMark' [enableVirtualization]='true'
popupHeight='200px'>
    </ejs-autocomplete>
  </div>
</div>

```

Grouping in Angular Auto complete component

The AutoComplete supports wrapping nested elements into a group based on different categories. The category of each list item can be mapped through the [groupBy](#) field in the data table. The group header is displayed as both inline and fixed headers. The fixed group header content is updated dynamically on scrolling the suggestion list with its category value.

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 { AutoCompleteModule } from '@syncfusion/ej2-angular-dropdowns'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component } from '@angular/core';
@Component({
  imports: [
    FormsModule, ReactiveFormsModule, AutoCompleteModule, ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the AutoComplete component
  template: `<ejs-autocomplete id='atcelement' [dataSource]='vegetableData'
[fields]='fields' [placeholder]='text'></ejs-autocomplete>`
})

```

```

export class AppComponent {
  constructor() {
  }
  // defined the array of data
  public vegetableData: { [key: string]: Object }[] = [
    { Vegetable: 'Cabbage', Category: 'Leafy and Salad', Id: 'item1' },
    { Vegetable: 'Spinach', Category: 'Leafy and Salad', Id: 'item2' },
    { Vegetable: 'Wheat grass', Category: 'Leafy and Salad', Id: 'item3' },
  ],
  { Vegetable: 'Yarrow', Category: 'Leafy and Salad', Id: 'item4' },
  { Vegetable: 'Pumpkins', Category: 'Leafy and Salad', Id: 'item5' },
  { Vegetable: 'Chickpea', Category: 'Beans', Id: 'item6' },
  { Vegetable: 'Green bean', Category: 'Beans', Id: 'item7' },
  { Vegetable: 'Horse gram', Category: 'Beans', Id: 'item8' },
  { Vegetable: 'Garlic', Category: 'Bulb and Stem', Id: 'item9' },
  { Vegetable: 'Nopal', Category: 'Bulb and Stem', Id: 'item10' },
  { Vegetable: 'Onion', Category: 'Bulb and Stem', Id: 'item11' } ];
  // maps the appropriate column to fields property
  public fields: Object = { groupBy: 'Category', value: 'Vegetable' };
  // set the placeholder to the AutoComplete input
  public text: string = "Find a vegetable";
}

```

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

[Group Template support to AutoComplete.](#)

Filtering in Angular Auto complete component

The AutoComplete has built-in support to filter data items. The filter operation starts as soon as you start typing characters in the component.

Change the filter type

Determines on which filter type, the component needs to be considered on search action. The available [filterType](#) and its supported data types are

Filter Type	Description	Supported Types
---	---	---
StartsWith	Checks whether a value begins with the specified value.	String
EndsWith	Checks whether a value ends with specified value.	String
Contains	Checks whether a value contains with specified value.	String

The following examples shows the data filtering is done with **StartsWith** type.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule, ReactiveFormsModule } from '@angular/forms'
import { AutoCompleteModule } from '@syncfusion/ej2-angular-dropdowns'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component } from '@angular/core';
//import DataManager related classes
import { Query, DataManager, ODataV4Adaptor } from '@syncfusion/ej2-data';
@Component({
  imports: [
    FormsModule, ReactiveFormsModule, AutoCompleteModule, ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the AutoComplete component
  template: `<ejs-autocomplete id='atcelement' [dataSource]='sportsData'
[fields]='fields' [placeholder]='text' [query]='query'
[filterType]='filterType' [sortOrder]='sorting'></ejs-autocomplete>`
})
export class AppComponent {
  constructor() {
  }
  //bind the DataManager instance to dataSource property
  public sportsData: DataManager = new DataManager({
    url:
'https://services.odata.org/V4/Northwind/Northwind.svc/Customers',
    adaptor: new ODataV4Adaptor,
    crossDomain: true
  });
  // maps the appropriate column to fields property
  public fields: Object = { value: 'ContactName' };
  //bind the Query instance to query property
  public query: Query = new Query().select(['ContactName',
'CustomerID']).take(6);
  //set the placeholder to AutoComplete input
  public text: string = "Find a customer";
  //set the filterType to searching operation
  public filterType: string='StartsWith';
  //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));

```

Filter item count

You can specify the filter suggestion item count through [suggestionCount](#) property of AutoComplete.

The following examples, to restrict the suggestion list item counts as 5.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule, ReactiveFormsModule } from '@angular/forms'
import { AutoCompleteModule } from '@syncfusion/ej2-angular-dropdowns'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component } from '@angular/core';
//import DataManager related classes
import { Query, DataManager, ODataV4Adaptor } from '@syncfusion/ej2-data';
@Component({
  imports: [
    FormsModule, ReactiveFormsModule, AutoCompleteModule, ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the AutoComplete component
  template: `<ejs-autocomplete id='atcelement' [dataSource]='sportsData'
[fields]='fields' [placeholder]='text' [query]='query'
[filterType]='filterType' [sortOrder]='sorting'
[suggestionCount]='suggestionCount'></ejs-autocomplete>`
})
export class AppComponent {
  constructor() {
  }
  //bind the DataManager instance to dataSource property
  public sportsData: DataManager = new DataManager({
    url:
    'https://services.odata.org/V4/Northwind/Northwind.svc/Customers',
    adaptor: new ODataV4Adaptor,
    crossDomain: true
  });
  // maps the appropriate column to fields property
  public fields: Object = { value: 'ContactName' };
  //bind the Query instance to query property
  public query: Query = new Query().select(['ContactName', 'CustomerID']);
  //set the placeholder to AutoComplete input
  public text: string = "Find a customer";
  //set the filterType to searching operation
  public filterType: string='StartsWith';
  //sort the result items
  public sorting: string='Ascending';
  //set the suggestionCount to show the maximum suggestion list item
  public suggestionCount: Number= 5;
}

```

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

You can set the limit for the character count to filter the data on the AutoComplete. This can be done by set the [minLength](#) property to AutoComplete.

In the following example, the remote request doesn't fetch the search data, until the search key contains three characters.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule, ReactiveFormsModule } from '@angular/forms'
import { AutoCompleteModule } from '@syncfusion/ej2-angular-dropdowns'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component } from '@angular/core';
//import DataManager related classes
import { Query, DataManager, ODataV4Adaptor } from '@syncfusion/ej2-data';
@Component({
  imports: [
    FormsModule, ReactiveFormsModule, AutoCompleteModule, ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the AutoComplete component
  template: `<ejs-autocomplete id='atcelement' [dataSource]='sportsData'
[fields]='fields' [placeholder]='text' [query]='query'
[filterType]='filterType' [sortOrder]='sorting'
[minLength]='minLength'></ejs-autocomplete>`
})
export class AppComponent {
  constructor() {
  }
  //bind the DataManager instance to dataSource property
  public sportsData: DataManager = new DataManager({
    url:
'https://services.odata.org/V4/Northwind/Northwind.svc/Customers',
    adaptor: new ODataV4Adaptor,
    crossDomain: true
  });
  // maps the appropriate column to fields property
  public fields: Object = { value: 'ContactName' };
  //bind the Query instance to query property
  public query: Query = new Query().select(['ContactName', 'CustomerID']);
  //set the placeholder to AutoComplete input
  public text: string = "Find a customer";
  //set the filterType to searching operation
  public filterType: string='StartsWith';
  //sort the result items
  public sorting: string = 'Ascending';
  //set the minLength to restrict the remote request until search key
contains 3 characters.
  public minLength: Number = 3;
}
```

MAIN.TS

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

Case sensitive filtering

Data items can be filtered either with or without case sensitivity using the DataManager. This can be done by setting the [ignoreCase](#) property of AutoComplete.

The following sample depicts how to filter the data with case-sensitive.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule, ReactiveFormsModule } from '@angular/forms'
import { AutoCompleteModule } from '@syncfusion/ej2-angular-dropdowns'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component } from '@angular/core';
@Component({
  imports: [
    FormsModule, ReactiveFormsModule, AutoCompleteModule, ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the AutoComplete component
  template: `<ejs-autocomplete id='atcelement' [dataSource]='data'
[placeholder]='text' [filterType]='filterType' [sortOrder]='sorting'
[ignoreCase]='ignoreCase'></ejs-autocomplete>`
})
export class AppComponent {
  constructor() {
  }
  //bind the DataManager instance to dataSource property
  public data: string[] = ['Badminton', 'Basketball', 'Cricket',
'Football', 'Golf', 'Gymnastics', 'Hockey', 'Tennis'];
  //set the placeholder to AutoComplete input
  public text: string = "Find a game Eg: FootBall";
  //set the filterType to searching operation
  public filterType: string='StartsWith';
  //sort the result items
  public sorting: string = 'Ascending';
  //set the minLength to restrict the remote request until search key
contains 3 characters.
  public ignoreCase: Boolean = false;
}
```

MAIN.TS

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

Diacritics Filtering

An AutoComplete supports diacritics filtering which will ignore the [diacritics](#) and makes it easier to filter the results in international characters lists when the [ignoreAccent](#) is enabled.

In the following sample, data with diacritics are bound as dataSource for AutoComplete.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule, ReactiveFormsModule } from '@angular/forms'
import { AutoCompleteModule } from '@syncfusion/ej2-angular-dropdowns'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component } from '@angular/core';
@Component({
  imports: [
    FormsModule, ReactiveFormsModule, AutoCompleteModule, ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the DropDownList component with
  // change event
  template: `<ejs-autocomplete id='diacritics' [dataSource]='data'
[ignoreAccent]='true' placeholder='e.g. aero'>
    </ejs-autocomplete>`
})
export class AppComponent {
  constructor() {
  }
  // create local data
  public data: string[] = [
    'Aeróbics',
    'Aeróbics en Agua',
    'Aerografía',
    'Aerodelaje',
    '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 achieve autofill while filtering](#)
- [How to group the data using header](#)
- [How to highlight the search data](#)

Localization in Angular Auto complete component

The Localization library allows you to localize static text content of the

[noRecordsTemplate](#) and [actionFailureTemplate](#) property according to the culture currently assigned to the AutoComplete.

| Locale key | en-US (default) |

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

| noRecordsTemplate | No Records Found |

| actionFailureTemplate | The Request Failed |

Loading translations

To load translation object to your application, use load function of the **L10n** class.

In the following sample, French culture is set to the AutoComplete and no data is loaded. Hence, the [noRecordsTemplate](#) property displays its text in French culture initially and if the sample is run offline, the [actionFailureTemplate](#) property displays its text appropriately.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule, ReactiveFormsModule } from '@angular/forms'
import { AutoCompleteModule } 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, AutoCompleteModule, ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the AutoComplete component
  template: `<ejs-autocomplete id='atcelement' [dataSource]='data'
[query]='query' [fields]='fields' [placeholder]='text'
[locale]='locale'></ejs-autocomplete>`
})
export class AppComponent implements OnInit {
  constructor() {
    //set the placeholder text in french to AutoComplete input
    public text: string = 'Trouver un client';
    // 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 the appropriate columns to fields property
    public fields: Object = { value: 'ContactName' };
    //bind the Query instance to query property
```

```

    public query: Query = new Query().select(['ContactName',
'CustomerID']).take(0);
    // set placeholder to AutoComplete input element
    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 autocomplete](#)

Style in Angular Auto complete component

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

Customizing the appearance of wrapper element

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

```

`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;
}
、
```

Customizing the outline theme's focus color

Use the following CSS to customize the focusing color of outline theme.

```
`css
.e-outline.e-input-group.e-input-focus: hover: not(.e-success): not(.e-warning): not(.e-error): not(.e-
disabled): not(.e-float-icon-left),.e-outline.e-input-group.e-input-focus.e-control-wrapper: hover: not(.e-
success): not(.e-warning): not(.e-error): not(.e-disabled): not(.e-float-icon-left),.e-outline.e-input-group.e-
input-focus: not(.e-success): not(.e-warning): not(.e-error): not(.e-disabled),.e-outline.e-input-group.e-
control-wrapper.e-input-focus: not(.e-success): not(.e-warning): not(.e-error): not(.e-disabled) {
border-color: #b1bd15;
box-shadow: inset 1px 1px #b1bd15, inset -1px 0 #b1bd15, inset 0 -1px #b1bd15;
}
、
```

Customizing the 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;
}
、
```

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-input-group.e-control-wrapper.e-float-input.e-input-focus .e-float-text.e-label-top, .e-float-
input.e-control-wrapper:not(.e-error).e-input-focus input ~ label.e-float-text {
color: #2319b8;
}
```

,

Customizing the color of the placeholder text

Use the following CSS to customize the text color of placeholder.

```
`css
.e-ddl.e-input-group input.e-input::placeholder {
color: red;
}
```

,

Customizing the text selection color

Use the following CSS to customize the selection color of text and background.

```
`css
.e-ddl.e-input-group input.e-input::selection {
color: red;
background: yellow;
}
```

,

Customizing the background color of focus, hover, and active item's

Use the following CSS to customize the background color of focus, hover and active item's.

```
`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 { AutoCompleteModule } from '@syncfusion/ej2-angular-dropdowns'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component } from '@angular/core';
@Component({
  imports: [
    FormsModule, ReactiveFormsModule, AutoCompleteModule, ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the AutoComplete component with
  // dataSource
  template: `<ejs-autocomplete id='atcelement'
[dataSource]='sportsData'></ejs-autocomplete>`
})
export class AppComponent {
  constructor() {
  }
  // defined the array of data
  public sportsData: string[] = ['Badminton', 'Basketball', 'Cricket',
'Football', 'Golf', 'Gymnastics'];
}
```

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 Auto complete component

The AutoComplete component has been designed, keeping in mind the **WAI-ARIA** specifications, and applies the **WAI-ARIA** roles, states, and properties along with **keyboard support**. This component is characterized by complete keyboard interaction support and ARIA accessibility support that makes it easy for people who

use assistive technologies (AT) or those who completely rely on keyboard navigation.

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

The accessibility compliance for the AutoComplete component is outlined below.

| Accessibility Criteria | Compatibility |


| -- | -- |

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

<style>

.post .post-content img {

display: inline-block;

margin: 0.5em 0;

}

</style>

<div> - All features of the component meet the requirement.</div>

```
<div> - Some features of the component do not meet the requirement.</div>
```

```
<div> - The component does not meet the requirement.</div>
```

WAI-ARIA attributes

The AutoComplete component uses the **combobox** role and each list item has an **option** role. The following **ARIA Attributes** denote the AutoComplete state.

| Property | Functionalities |

| --- | --- |

| aria-haspopup | Indicates whether the AutoComplete input element has a suggestion list or not. |

| aria-expanded | Indicates whether the suggestion list has expanded or not. |

| aria-selected | Indicates the selected option from the list. |

| aria-readonly | Indicates the readonly state of the AutoComplete element. |

| aria-disabled | Indicates whether the AutoComplete component is in a disabled state or not. |

| aria-activedescendent | This attribute holds the ID of the active list item to focus its descendant child element. |

| aria-owns | This attribute contains the ID of the suggestion list to indicate popup as a child element. |

| aria-autocomplete | This attribute contains the 'both' to a list of options shows and the currently selected suggestion also shows inline. |

Keyboard interaction

You can use the following key shortcuts to access the AutoComplete without interruptions.

| Keyboard shortcuts | Actions |

| --- | --- |

| Arrow Down | In popup hidden state, opens the suggestion list. In popup open state, selects the first item when no item selected else selects the item next to the currently selected item. |

| Arrow Up | In popup hidden state, opens the suggestion list. In popup open state, selects the last item when no item selected else selects the item previous to the currently selected one. |

| Page Down | Scrolls down to the next page and selects the first item when popup list opens. |

| Page Up | Scrolls up to previous page and select the first item when popup list open. |

| Enter | Selects the focused item and set to AutoComplete component. |

| Tab | Focuses on the next tab indexed element when the popup is closed. Otherwise, closes the popup list and remains the focus in component suppose if it is in an open state. |

| Shift + tab | Focuses the previous tab indexed element when the popup is closed. Otherwise, closes the popup list and remains the focus in component suppose if it is in an open state. |

| Alt + Down | Opens the popup list. |

| Alt + Up | In popup hidden state, opens the popup list. In popup open state, closes the popup list. |

| Esc(Escape) | Closes the popup list when it is in an open state then remove the selection. |

| Home | Cursor moves to before of first character in input. |

| End | Cursor moves to next of last character in input. |

In the below sample, focus the AutoComplete component using alt+t keys.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule, ReactiveFormsModule } from '@angular/forms'
import { AutoCompleteModule } from '@syncfusion/ej2-angular-dropdowns'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component, HostListener, ViewChild } from '@angular/core';
import { AutoCompleteComponent } from '@syncfusion/ej2-angular-dropdowns';
@Component({
  imports: [
    FormsModule, ReactiveFormsModule, AutoCompleteModule, ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the AutoComplete component
  template: `<ejs-autocomplete id='atcelement' #samples
[dataSource]='sportsData' [fields]='fields' [placeholder]='text'></ejs-
autocomplete>`
})
export class AppComponent {
  @ViewChild('samples')
  public sports?: AutoCompleteComponent;
  constructor() {
  }
  public sportsData: { [key: string]: Object }[] = [
    { Id: 'Game1', Game: 'Badminton' },
    { Id: 'Game2', Game: 'Basketball' },
    { Id: 'Game3', Game: 'Cricket' },
    { Id: 'Game4', Game: 'Football' },
    { Id: 'Game5', Game: 'Golf' },
    { Id: 'Game6', Game: 'Hockey' },
    { Id: 'Game7', Game: 'Rugby' },
    { Id: 'Game8', Game: 'Snooker' },
    { Id: 'Game9', Game: 'Tennis' }
  ];
  // maps the appropriate column to fields property
  public fields: Object = { value: 'Game' };
  //set the placeholder to AutoComplete input
  public text: string = "Find a game";
  @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 AutoComplete component's accessibility levels are ensured through an [accessibility-checker](#) and [axe-core](#) software tools during automated testing.

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

See also

- [Accessibility in Syncfusion Angular components](#)

Form support in Angular Auto complete component

The AutoComplete 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 Autocomplete element which will be used to identify the form element. To register an Autocomplete element to ngForm, give the `ngModel` to it so the FormsModule will automatically detect the AutoComplete as a form element After that, the AutoComplete value will be selected based on the `ngModel` value.

so the FormsModule will automatically detect the AutoComplete as a form element. After that, the AutoComplete 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 { AutoCompleteModule } 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, AutoCompleteModule, 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';
    sname: any;
    skillname: any;
    smail: any;
    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 AutoComplete 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 { AutoCompleteModule } 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, AutoCompleteModule, 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 | any;
  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

Autofill in Angular Auto complete component

The AutoComplete supports the autofill behavior with the help of [autofill](#) property. Whenever you change the input value, the AutoComplete will autocomplete your data by matching the typed character. Suppose, if no matches found then, AutoComplete doesn't suggest any item.

In the below sample, showcase that how to work `autofill` with AutoComplete.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule, ReactiveFormsModule } from '@angular/forms'
import { AutoCompleteModule } from '@syncfusion/ej2-angular-dropdowns'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component } from '@angular/core';
@Component({
  imports: [
    FormsModule, ReactiveFormsModule, AutoCompleteModule, ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the AutoComplete component
  template: `<ejs-autocomplete id='atcelement' [dataSource]='searchData'
[fields]='fields' [placeholder]='text' [autofill]='autofill'></ejs-
autocomplete>`
})
export class AppComponent {
  constructor() {
  }
  // defined the array of data
  public searchData: { [key: string]: Object }[] = [
    { Name: 'Australia', Code: 'AU' },
    { Name: 'Bermuda', Code: 'BM' },
    { Name: 'Canada', Code: 'CA' },
    { Name: 'Cameroon', Code: 'CM' },
    { Name: 'Denmark', Code: 'DK' },
    { Name: 'France', Code: 'FR' },
    { Name: 'Finland', Code: 'FI' },
    { Name: 'Germany', Code: 'DE' },
    { Name: 'Greenland', Code: 'GL' },
    { Name: 'Hong Kong', Code: 'HK' },
    { Name: 'India', Code: 'IN' },
    { Name: 'Italy', Code: 'IT' },
    { Name: 'Japan', Code: 'JP' },
    { Name: 'Mexico', Code: 'MX' },
    { Name: 'Norway', Code: 'NO' },
    { Name: 'Poland', Code: 'PL' },
    { Name: 'Switzerland', Code: 'CH' },
    { Name: 'United Kingdom', Code: 'GB' },
    { Name: 'United States', Code: 'US' }];
  // maps the appropriate column to fields property
  public fields: Object = { value: "Name" };
  // set the placeholder to the AutoComplete input
  public text: string = "Find a country";
  //enable the highlight property to highlight the matched character in
  suggestion list
  public autofill: 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));

```

Icon support in Angular Auto complete component

You can render **icons** to the list items by mapping the `iconCss` field. This `iconCss` field create a span in the list item with mapped class name to allow styling as per your need.

In the following sample, the icon classes are mapped with `iconCss` field.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule } from '@angular/forms'
import { AutoCompleteModule } from '@syncfusion/ej2-angular-dropdowns'
import { Component } from '@angular/core';
@Component({
  imports: [
    FormsModule, AutoCompleteModule
  ],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the AutoComplete component
  template: `<ejs-autocomplete id='atcelement'
[dataSource]='sortFormatData' [fields]='fields' [placeholder]='text'></ejs-autocomplete>`
})
export class AppComponent {
  constructor() {
  }
  // defined the array of data
  public sortFormatData: { [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' }
  ];
  // maps the appropriate column to fields property
  public fields: Object = { value: 'Type', iconCss: 'Class' };
  // set the placeholder to the AutoComplete input
  public text: string = "Find 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));
```

Custom search in Angular Auto complete component

The AutoComplete has built-in support to highlight the searched characters on suggested list items when enabled the `highlight` property.

In the below sample, to customize the matched character in suggestion list by `e-highlight` class.

APP.COMPONENT.TS


```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule } from '@angular/forms'
import { AutoCompleteModule } from '@syncfusion/ej2-angular-dropdowns'
import { Component } from '@angular/core';
@Component({
  imports: [
    FormsModule, AutoCompleteModule
  ],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the AutoComplete component
  template: `<ejs-autocomplete id='atcelement' [dataSource]='searchData'
[fields]='fields' [placeholder]='text' [highlight]='highlight'></ejs-
autocomplete>`
})
export class AppComponent {
  constructor() {
  }
  // defined the array of data
  public searchData: { [key: string]: Object }[] = [
    { Name: 'Australia', Code: 'AU' },
    { Name: 'Bermuda', Code: 'BM' },
    { Name: 'Canada', Code: 'CA' },
    { Name: 'Cameroon', Code: 'CM' },
    { Name: 'Denmark', Code: 'DK' },
    { Name: 'France', Code: 'FR' },
    { Name: 'Finland', Code: 'FI' },
    { Name: 'Germany', Code: 'DE' },
    { Name: 'Greenland', Code: 'GL' },
    { Name: 'Hong Kong', Code: 'HK' },
    { Name: 'India', Code: 'IN' },
    { Name: 'Italy', Code: 'IT' },
    { Name: 'Japan', Code: 'JP' },
    { Name: 'Mexico', Code: 'MX' },
    { Name: 'Norway', Code: 'NO' },
    { Name: 'Poland', Code: 'PL' },
    { Name: 'Switzerland', Code: 'CH' },
    { Name: 'United Kingdom', Code: 'GB' },
    { Name: 'United States', Code: 'US' }];
  // maps the appropriate column to fields property
  public fields: Object = { value: "Name" };
  // set the placeholder to the AutoComplete input
  public text: string = "Find a country";
  //enable the highlight property to highlight the matched character in
  suggestion list
  public highlight: 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));

```

Filter in Angular Auto complete component

The AutoComplete data can be filtered based on both text and value fields using predicate of dataManager through filtering event. The filtered data can be again updated through updateData method.

In the following example, filtering is done based on text and value fields.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule } from '@angular/forms'
import { AutoCompleteModule } from '@syncfusion/ej2-angular-dropdowns'
import { Component } from '@angular/core';
import { Query, DataManager, Predicate } from '@syncfusion/ej2-data';
@Component({
  imports: [
    FormsModule, AutoCompleteModule
  ],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the AutoComplete component
  template: `<ejs-autocomplete id='ddlelement' #samples
[dataSource]='searchData' [fields]='fields' [placeholder]='text'
[itemTemplate]="itemTemplate"
[query]='query' (filtering)="onFiltering($event)"></ejs-autocomplete> `
})
export class AppComponent {
  query: any;
  constructor() {
  }
  // defined the array of data
  public searchData: { [key: string]: Object }[] = [
    { Name: 'Australia', Code: 'AU' },
    { Name: 'Bermuda', Code: 'BM' },
    { Name: 'Canada', Code: 'CA' },
    { Name: 'Cameroon', Code: 'CM' },
    { Name: 'Denmark', Code: 'DK' },
    { Name: 'France', Code: 'FR' },
    { Name: 'Finland', Code: 'FI' },
    { Name: 'Germany', Code: 'DE' },
    { Name: 'Greenland', Code: 'GL' },
    { Name: 'Hong Kong', Code: 'HK' },
    { Name: 'India', Code: 'IN' },
    { Name: 'Italy', Code: 'IT' },
    { Name: 'Japan', Code: 'JP' },
    { Name: 'Mexico', Code: 'MX' },
    { Name: 'Norway', Code: 'NO' },
    { Name: 'Poland', Code: 'PL' },
    { Name: 'Switzerland', Code: 'CH' },
    { Name: 'United Kingdom', Code: 'GB' },
    { Name: 'United States', Code: 'US' }];
  // maps the appropriate column to fields property
  public fields: Object = { value: "Code" , text:"Name"};
  // set the placeholder to the AutoComplete input
  public text: string = "Find a country";
```

```

    public itemTemplate:string= "<span><span class='name'>${Name}</span>-<br><span class='code'>${Code}</span></span>";
    public onFiltering (e : any)
    {
        e.preventDefault=true;
        var predicate = new Predicate('Name', 'contains', e.text);
        predicate = predicate.or('Code', 'contains', e.text);
        var query = new Query();
        //frame the query based on search string with filter type.
        query = (e.text != "") ? query.where(predicate) : query;
        //pass the filter data source, filter query to updateData method.
        e.updateData(this.searchData, query);
    }
}

```

MAIN.TS

```

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

```

Suggestion in Angular Auto complete component

The AutoComplete supports to displaying suggestion list upon focusing an empty auto complete component, using the focus event in the control. We have used the filtering and change events to get the typed and selected words and stored them in the browser's local storage. Then using the focus event, we have displayed the stored list as suggestions.

In the below sample, showcase that how to show **suggestion list** with AutoComplete.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule, ReactiveFormsModule } from '@angular/forms'
import { AutoCompleteModule } from '@syncfusion/ej2-angular-dropdowns'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component, ViewChild } from '@angular/core';
import { Query, DataManager, WebApiAdaptor } from '@syncfusion/ej2-data';
import { AutoCompleteComponent, FilteringEventArgs } from '@syncfusion/ej2-angular-dropdowns';
@Component({
  imports: [
    FormsModule, ReactiveFormsModule, AutoCompleteModule, ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-autocomplete id='country' #local [dataSource]='countries' [fields]='localFields' (change)='onChange()' (filtering)='onFiltering($event)' (focus)='onFocus()'></ejs-autocomplete>`,
})
export class AppComponent {
  @ViewChild('local')
  public localObj : AutoCompleteComponent | any;
  public suggestList: Array<string> = [];
}

```

```

public countries?: { [key: string]: Object; }[] = [
  { Name: 'Australia', Code: 'AU' },
  { Name: 'Bermuda', Code: 'BM' },
  { Name: 'Canada', Code: 'CA' },
  { Name: 'Cameroon', Code: 'CM' },
  { Name: 'Denmark', Code: 'DK' },
  { Name: 'France', Code: 'FR' },
  { Name: 'Finland', Code: 'FI' }
];
// maps the local data column to fields property
public localFields: Object = { value: 'Name' };
onChange() {
  localStorage.setItem("value", this.localObj.value as string);
  if (localStorage.getItem('value') !== 'null') {
    this.suggestList.push((localStorage as any).getItem('value'));
    var proxy = this;
    this.suggestList = this.suggestList.filter(function (item, pos, self) {
      return proxy.suggestList.indexOf(item) == pos;
    });
  }
}
onFocus() {
  if (this.suggestList.length > 0) {
    (this.localObj.dataSource as any) = this.suggestList;
    this.localObj.dataBind();
    let keyEventArgs: any = { preventDefault: (): void => { }, action:
'down', keyCode: 40, type: null };
    (this.localObj as any).onFilterUp(keyEventArgs);
    (this.localObj as any).popupObj.element.classList.add('e-suggestion');
  }
}
onFiltering(e: FilteringEventArgs){
  let query: any = new Query();
  query = (e.text !== '') ? query.where('Name', 'startswith', e.text, true)
: query;
  e.updateData((this as any).countries, query);
  (this.localObj as any).popupObj.element.classList.remove('e-suggestion');
}
}

```

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 Auto complete component

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

MultiSelect concept is not present in EJ2-AutoComplete. If you want to use multiselection support in autocomplete, we suggest you to use MultiSelect component.

DataBinding

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

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

| **Default** | **Property:** *dataSource* `
<input type="text" id="databindinglocal" ej-autocomplete [dataSource]="states" />` | **Property:** *datasource* `
<ejs-autocomplete [datasource]="data"></ejs-autocomplete>` |

| **Fields for mapping** | **Property:** *fields* `
<input type="text" id="databindinglocal" ej-autocomplete [fields]="fields" />` | **Property:** *fields* `
<ejs-autocomplete id="country" [fields]="fields"></ejs-autocomplete>` |

| **Query** | **Property:** *query* `
<input type="text" id="databindinglocal" ej-autocomplete [query]="query" />` | **Property:** *query* `
<ejs-autocomplete id="autocomplete" [query]="ej.Query().from('Customers').take(10)"></ejs-autocomplete>` |

| **Begin event** | **Event:** *actionBegin* `
<input type="text" id="databindinglocal" ej-autocomplete (actionBegin)="actionBegin($event)" />` | **Event:** *actionBegin* `
 <ejs-autocomplete id="autocomplete" (actionBegin)="onBegin($event)"></ejs-autocomplete>` |

| **Complete event** | **Event:** *actionComplete* `
<input type="text" id="databindinglocal" ej-autocomplete (actionComplete)="actionComplete($event)" />` | **Event:** *actionComplete* `
 <ejs-autocomplete id="autocomplete" (actionComplete)="onComplete($event)"></ejs-autocomplete>` |

| **Failure event** | **Event:** *actionFailure* `
<input type="text" id="databindinglocal" ej-autocomplete (actionFailure)="actionFailure($event)" />` | **Event:** *actionFailure* `
 <ejs-autocomplete id="autocomplete" (actionFailure)="onFailure($event)"></ejs-autocomplete>` |

| **Success event** | **Event:** *actionSuccess* `
<input type="text" id="databindinglocal" ej-autocomplete (actionSuccess)="actionSuccess($event)" />` | **Not Applicable** |

Filtering

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

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

| **Case sensitivity** | **Property:** *caseSensitiveSearch* `
<input type="text" id="databindinglocal" ej-autocomplete [caseSensitiveSearch]="true" />` | **Property:** *ignoreCase* `
<ejs-autocomplete id="autocomplete" [ignoreCase]="ignoreCase"></ejs-autocomplete>` |

| **Accent effective search** | **Not applicable** | **Property :** *ignoreAccent* `
<ejs-autocomplete id="autocomplete" [ignoreAccent]="ignoreAccent"></ejs-autocomplete>` |

| **Filtering Type** | **Property:** *filterType* `
<input type="text" id="databindinglocal" ej-autocomplete [filterType]="filterType" />` | **Property:** *filterType* `
<ejs-autocomplete id="autocomplete" [filterType]="filterType"></ejs-autocomplete>` |

| **Autofill** | **Property:** *enableAutoFill* `
<input type="text" id="databindinglocal" ej-autocomplete [enableAutoFill]="enableAutoFill" />` | **Property:** *autoFill* `
<ejs-autocomplete id="autocomplete" [autoFill]="autoFill"></ejs-autocomplete>` |

| **Highlight the search word** | **Property:** *highlightSearch* `<input type="text" id="databindinglocal" ej-autocomplete [highlightSearch]="highlightSearch" />` | **Property:** *highlight* `
ej-autocomplete id="autocomplete" [highlight]="highlight"></ej-autocomplete>` |

| **No of items to be shown** | **Property:** *itemsCount* `
<input type="text" id="databindinglocal" ej-autocomplete [itemsCount]="itemsCount" />` | **Property:** *suggestionCount* `
<ej-autocomplete id="autocomplete" [suggestionCount]="suggestionCount"></ej-autocomplete>` |

| **Minimum characters to enter** | **Property:** *minCharacter* `
<input type="text" id="databindinglocal" ej-autocomplete [minCharacter]="minCharacter" />` | **Property:** *minLength* `
ej-autocomplete id="autocomplete" [minLength]="minLength"></ej-autocomplete>` |

| **Search** | **Method:** *search* `
<input type="text" id="databindinglocal" ej-autocomplete />

$("#autocomplete").ejAutoComplete("search");` | **Not applicable** |

Placeholder

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

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

| **Watermark text** | **Property:** *watermarkText* `
<input type="text" id="databindinglocal" ej-autocomplete [watermarkText]="select" />` | **Property:** *placeholder* `
ej-autocomplete id="autocomplete" [placeholder]="select"></ej-autocomplete>` |

| **Floating of watermark text** | **Not applicable** | **Property:** *floatLabelType* `
ej-autocomplete id="autocomplete" [floatLabelType]="floatLabelType"></ej-autocomplete>` |

Popup

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

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

| **No records showing** | **Property:** *showEmptyResultText* `
<input type="text" id="databindinglocal" ej-autocomplete [showEmptyResultText]="true" />` | **Not applicable** |

| **Popupbutton** | **Property:** *showPopupButton* `
<input type="text" id="databindinglocal" ej-autocomplete [showPopupButton]="true" />` | **Property:** *showPopupButton* `
<ej-autocomplete id="autocomplete" [showPopupButton]="true"></ej-autocomplete>` |

| **Clear button** | **Property:** *showResetIcon* `
<input type="text" id="databindinglocal" ej-autocomplete [showResetIcon]="true" />` | **Property:** *showClearButton* `
<ej-autocomplete id="autocomplete" [showClearButton]="true"></ej-autocomplete>` |

| **Animation** | **Property:** *animateType* `
<input type="text" id="databindinglocal" ej-autocomplete [animateType]="animateType" />` | **Not Applicable** |

| **Focusing the list item** | **Property:** *autoFocus* `
<input type="text" id="databindinglocal" ej-autocomplete [autoFocus]="true" />` | **Not applicable** |

| **Delaying the popup open time** | **Property:** *delaySuggestionTimeout*

 <input type="text" id="databindinglocal" ej-
 autocomplete [delaySuggestionTimeout]="delaySuggestionTimeout" /> | **Not applicable** |

| **Popup text when there is no popup items** | **Property:** *emptyResultText*

 <input type="text" id="databindinglocal" ej-autocomplete [emptyResultText]="no records" />
 | <https://ej2.syncfusion.com/angular/demos/#/material/auto-complete/template> |

| **Enable/disable the duplicate option** | **Property:** *enableDistinct*

 <input type="text" id="databindinglocal" ej-autocomplete [enableDistinct]="true" /> | **Not applicable** |

| **Popup height** | **Property:** *popupHeight*
 <input type="text" id="databindinglocal" ej-
 autocomplete [popupHeight]="popupHeight" /> | **Property:** *popupHeight*
 <ejs-autocomplete
 id="autocomplete" [popupHeight]="300px"></ejs-autocomplete> |

| **Popup Width** | **Property:** *popupWidth*
 <input type="text" id="databindinglocal" ej-
 autocomplete [popupWidth]="popupWidth" /> | **Property:** *popupWidth*
 <ej-autocomplete
 id="autocomplete" [popupWidth]="300px"></ej-autocomplete> |

| **Open popup** | **Method:** *open*
 <input type="text" id="databindinglocal" ej-
 autocomplete />
\$(("#autocomplete").ejAutoComplete("open")); | **Method:** *showPopup*

 <ej-autocomplete id="autocomplete"></ej-autocomplete>

@ViewChild('sample')
 public obj: AutoCompleteComponent;
obj.open(); |

| **Close event** | **Event:** *close*
 <input type="text" id="databindinglocal" ej-
 autocomplete (close)="close(\$event)" /> | **Event:** *close*
 <ej-autocomplete
 id="autocomplete" (close)="onClose(\$event)"></ej-autocomplete> |

| **Open event** | **Event:** *open*
 <input type="text" id="databindinglocal" ej-
 autocomplete (open)="open(\$event)" /> | **Event:** *open*
 <ej-autocomplete
 id="autocomplete" (open)="onOpen(\$event)"></ej-autocomplete> |

CSS

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

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

| **Default** | **Property:** *cssClass*
 <input type="text" id="databindinglocal" ej-
 autocomplete [cssClass]="cssClass" /> | **Property:** *cssClass*
 <ej-autocomplete
 id="autocomplete" [cssClass]="customClass"></ej-autocomplete> |

| **Height** | **Property:** *height*
 <input type="text" id="databindinglocal" ej-
 autocomplete [height]="height" /> | Achievable through the [cssClass](#) property. |

| **showRoundedCorner** | **Property:** *showRoundedCorner*

 <input type="text" id="databindinglocal" ej-
 autocomplete [showRoundedCorner]="showRoundedCorner" /> | Achievable through the
[cssClass](#) property. |

| **Width** | **Property:** *width*
 <input type="text" id="databindinglocal" ej-autocomplete [width]="width" /> | **Property:** *width*
 <ej-autocomplete id="autocomplete" [width]="300px"></ej-autocomplete> |

| **Visibility** | **Property:** *visible*
 <input type="text" id="databindinglocal" ej-autocomplete [visible]="true" /> | Achievable through the [cssClass](#) property. |

Grouping

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

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

| **Default** | **Property:** *fields*
 <input type="text" id="databindinglocal" ej-autocomplete [fields]="fields" /> | **Property:** *fields*
 <ejs-autocomplete id="country" [fields]="fields"></ejs-autocomplete> |

Localization

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

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

| **Default** | **Property:** *locale*
 <input type="text" id="databindinglocal" ej-autocomplete [locale]="fr-FE" /> | **Property:** *locale*
 <ejs-autocomplete id="country" [locale]="locale"></ejs-autocomplete> |

Template

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

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

| **Default** | **Property:** *template* <input type="text" id="databindinglocal" ej-autocomplete [template]="template" /> | **Property:** *itemTemplate*
 <ejs-autocomplete id="employees" [itemTemplate]="itemTemplate"></ejs-autocomplete> |

| **Group Template** | **Not Applicable** | **Property:** *groupTemplate*
 <ejs-autocomplete id="employees" [groupTemplate]="groupTemplate"></ejs-autocomplete> |

| **ValueTemplate** | **Not applicable** | **Property:** *valueTemplate*
 <ejs-autocomplete id="employees" valueTemplate="data"></ejs-autocomplete> |

| **Header Template** | **Not applicable** | **Property:** *headerTemplate*
 <ejs-autocomplete id="employees" [headerTemplate]="headerTemplate"></ejs-autocomplete> |

| **FooterTemplate** | **Not applicable** | **Property:** *footerTemplate*
 <ejs-autocomplete id="employees" [footerTemplate]="footerTemplate"></ejs-autocomplete> |

| **No records Template** | **Not applicable** | **Property:** *noRecordsTemplate*
 <ejs-autocomplete id="employees" [noRecordsTemplate]="noRecordsTemplate"></ejs-autocomplete> |

| **Action failure Template** | **Not applicable** | **Property:** *actionFailureTemplate*
 <ejs-autocomplete id="employees" [actionFailureTemplate]="actionFailureTemplate"></ejs-autocomplete> |

Sorting

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

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

| **Default** | **Property:** *allowSorting*
<input type="text" id="databindinglocal" ej-autocomplete [allowSorting]="true" /> | Achievable through the [sortOrder](#) property. |

| **Order of sorting** | **Property:** *sortOrder*
<input type="text" id="databindinglocal" ej-autocomplete [sortOrder]="sortOrder" /> | **Property:** *sortOrder*
 <ejs-autocomplete id="country" [sortOrder]="sortOrder"></ejs-autocomplete> |

Accessibility

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

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

| **RTL support** | **Property:** *enableRtl*
<input type="text" id="databindinglocal" ej-autocomplete [enableRtl]="true" /> | **Property:** *enableRtl*
 <ejs-autocomplete id="country" [enableRtl]="true"></ejs-autocomplete> |

Selection

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

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

| **Selecting particular value** | **Property:** *selectValueByKey*
<input type="text" id="databindinglocal" ej-autocomplete [selectValueByKey]="selectValueByKey" /> | Achievable through the [value](#) property. | **Property:** *value*
<input type="text" id="databindinglocal" ej-autocomplete [value]="value" /> | **Property:** *value*
 <ejs-autocomplete id="country" [value]="data"></ejs-autocomplete> |

| **Selecting particular text** | **Property:** *text*
 <input type="text" id="databindinglocal" ej-autocomplete [text]="text" /> | **Not applicable** |

| **Selecting particular value** | **Method:** *selectValueByKey*
<input type="text" id="databindinglocal" ej-autocomplete />
@ViewChild('sample') public obj: AutoCompleteComponent;
obj.selectValueByKey("key") | Achievable through the [value](#) property. |

| **Selecting particular text** | **Method:** *selectValueByText*
<input type="text" id="databindinglocal" ej-autocomplete />
@ViewChild('sample') public obj: AutoCompleteComponent;
 obj.selectValueByText("key") | **Not applicable** |

| **Select event** | **Event:** *select*
<input type="text" id="databindinglocal" ej-autocomplete (select)="select(\$event)" /> | **Event:** *select*
 <ejs-autocomplete id="country" (select)="select(\$event)"></ejs-autocomplete> |

Miscellaneous

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

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

| **Enable/disable** | **Property:** *enabled*
 <input type="text" id="databindinglocal" ej-autocomplete [enabled]="true" /> | **Property:** *enabled*
 <ejs-autocomplete id="country" [enabled]="true"></ejs-autocomplete> |

| **Enable persistence** | **Property:** *enablePersistence*
 <input type="text" id="databindinglocal" ej-autocomplete [enablePersistence]="true" /> | **Property:** *enablePersistence*
 <ejs-autocomplete id="country" [enablePersistence]="true"></ejs-autocomplete> |

| **Loading icon** | **Property:** *showLoadingIcon*
 <input type="text" id="databindinglocal" ej-autocomplete [showLoadingIcon]="true" /> | **By default, it is showing** |

| **Read only** | **Property:** *readOnly*
 <input type="text" id="databindinglocal" ej-autocomplete [readOnly]="true" /> | **Property:** *readOnly* <ej-autocomplete id="autocomplete" [readOnly]="true"></ej-autocomplete> |

| **Disable** | **Method:** *disable*
 <input type="text" id="databindinglocal" ej-autocomplete />
 @ViewChild('sample') public obj: AutoCompleteComponent;
 obj.disable(); | Achievable through the [enabled](#) property. |

Common

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

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

| **Addition of new option watermark text** | **Property:** *addNewText*
 <input type="text" id="databindinglocal" ej-autocomplete [addNewText]="addNewText" /> | **Not applicable** |

| **Addition of new item** | **Property:** *allowAddNew*
 <input type="text" id="databindinglocal" ej-autocomplete [allowAddNew]="allowAddNew" /> | **Property:** *allowCustom*
 <ej-autocomplete id="autocomplete" [allowCustom]="true"></ej-autocomplete> |

| **Reset the autocomplete** | **Property:** *showResetIcon*
 <input type="text" id="databindinglocal" ej-autocomplete [showResetIcon]="showResetIcon" /> | **Property:** *showClearIcon*
 <ejs-autocomplete id="country" [showClearIcon]="true"></ejs-autocomplete> |

| **Destroy** | **Method:** *destroy*
 <input type="text" id="autocomplete" ej-autocomplete />
 \$(("#autocomplete").ejAutoComplete("destroy")); | **Method:** *destroy*
 <ejs-autocomplete id="country"></ejs-autocomplete>

 @ViewChild('sample') public autoObj: AutoCompleteComponent;

 autoObj.destroy(); |

| **Reset the autocomplete** | **Method:** *clearText*
 <input type="text" id="databindinglocal" ej-autocomplete />
 @ViewChild('sample') public obj: AutoCompleteComponent;
 obj.clearText(); | <ej-autocomplete id="autocomplete" [value]=""></ej-autocomplete> |

| **Multicolumn** | **Property:** *multiColumnSettings*
 <ej-autocomplete id="autocomplete" datasource="ViewBag.datasource"><e-multicolumnsettings enable="true" show-header="true" string-format="{0} ({1})" search-column-indices="@val.SearchColumnIndices"><e-multi-

```
columns><e-multi-column field="UniqueKey" header-text="Unique Key"></e-multi-column><e-
multi-column field="Text" header-text="Text"></e-multi-column></e-multi-columns></e-
multicolumnsettings></ej-autocomplete> | Not applicable |
```

| **Hide the Autocomplete** | Method: *hide*
 <input type="text" id="databindinglocal" ej-autocomplete />
@ViewChild('sample') public obj:

AutoCompleteComponent;
obj.hide(); | Achievable through the [cssClass](#) property.

| **Getting particular text** | Method: *getActiveText*
 <input type="text" id="databindinglocal" ej-autocomplete />
@ViewChild('sample') public obj:

AutoCompleteComponent;
obj.getActiveText(); | **Not applicable** |

| **Getting particular value** | Method: *getValue*
 <input type="text" id="databindinglocal" ej-autocomplete />
@ViewChild('sample') public obj:

AutoCompleteComponent;
obj.getValue(); | Achievable through the [value](#) property. |

| **Change event** | Event: *change*
<input type="text" id="databindinglocal" ej-autocomplete (change)="change(\$event)" /> | Event: *change*
<ejs-autocomplete id="country" (change)="onChange(\$event)"></ejs-autocomplete> |

| **Create event** | Event: *create*
<input type="text" id="databindinglocal" ej-autocomplete (create)="create(\$event)" /> | Event: *created*
<ejs-autocomplete id="country" (created)="onCreated(\$event)"></ejs-autocomplete> |

| **Destroy event** | Event: *destroy*
<input type="text" id="databindinglocal" ej-autocomplete (destroy)="destroy(\$event)" /> | Event: *destroyed*
<ejs-autocomplete id="country" (destroyed)="onDestroy(\$event)"></ejs-autocomplete> |

| **Focus out event** | Event: *focusOut*
<input type="text" id="databindinglocal" ej-autocomplete (focusOut)="focusOut(\$event)" /> | Event: *blur*
<ejs-autocomplete id="country" (blur)="onChange(\$event)"></ejs-autocomplete> |

| **Focus in event** | Event : *focusIn*
<input type="text" id="databindinglocal" ej-autocomplete (focusIn)="focusIn(\$event)" /> | Event: *focus*
<ejs-autocomplete id="country" (focus)="onFocus(\$event)"></ejs-autocomplete> |

Avatar

Getting started with Angular Avatar component

The following section explains the steps required to create a simple **Avatar** component using styles and its basic usage.

Setting up angular project

Angular provides the easiest way to set angular CLI projects using Angular CLI tool.

Install the CLI application globally to your machine by using following command.

```
`sh
```

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

```
`
```

Create a new angular application

```
`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.

Dependencies

The `Avatar` component is pure CSS component which doesn't need specific dependencies to render.

```
`javascript
|-- @syncfusion/ej2-layouts
`
```

Installing Syncfusion Layouts 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-layouts](#) package to the application.

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

```
`bash
npm install @syncfusion/ej2-angular-layouts@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-layouts:"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 Avatar component, import the Avatar 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-layouts/styles/material.css';
@import "../node_modules/@syncfusion/ej2-angular-layouts/styles/material.css";
`
```

Note: To refer the combined component styles, use Syncfusion [CRG](#) (Custom Resource Generator) in your application.

Add Avatar into application

Modify the `template` in `app.component.ts` file to render Avatar component.

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

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

@Component({
  selector: 'my-app',
  template: <div id='element'><span class="e-avatar">GR</span></div>
})
export class AppComponent {}
`
```

Run the application

Run the application in the browser using the following command.

```
`html
npm start
`
```

The following example shows a basic Avatar component.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Component } from '@angular/core';
@Component({
  imports: [

    ],
  standalone: true,
  selector: 'my-app',
  template: `
    <div id='element'>
      <span class="e-avatar e-avatar-xlarge"></span>
      <span class="e-avatar e-avatar-large"></span>
      <span class="e-avatar"></span>
      <span class="e-avatar e-avatar-small"></span>
      <span class="e-avatar e-avatar-xsmall"></span>
    </div>
  `
})
export class AppComponent { }
```

MAIN.TS

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

You can refer to our [Angular Avatar](#) feature tour page for its groundbreaking feature representations. You can also explore our [Angular Avatar example](#) to know how to present and manipulate data.

Types in Angular Avatar component

This section explains different types of avatar.

Avatar size

The Essential JS 2 Avatar has the following predefined sizes that can be used with the `.e-avatar` class to change the appearance of the avatar.

Class Name	Description
:-----	:-----
e-avatar-xlarge	Displays xlarge size avatar.
e-avatar-large	Displays apply large size avatar.
e-avatar	Displays apply default size avatar.
e-avatar-small	Displays apply small size avatar.
e-avatar-xsmall	Displays apply xsmall size avatar.

APP.COMPONENT.TS

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

```
@Component({
  imports: [

  ],
  standalone: true,
  selector: 'my-app',
  template: `
    <div id='element'>
      <span class="e-avatar e-avatar-xlarge"></span>
      <span class="e-avatar e-avatar-large"></span>
      <span class="e-avatar"></span>
      <span class="e-avatar e-avatar-small"></span>
      <span class="e-avatar e-avatar-xsmall"></span>
    </div>
  `
})
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));
```

Avatar types

The types of Essential JS 2 avatar are:

- Default
- Circle

Default

The default style of the avatar is rectangular shape with rounded corners, which can be applied from adding the modifier class `.e-avatar` to the target element.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Component } from '@angular/core';
@Component({
  imports: [

  ],
  standalone: true,
  selector: 'my-app',
  template: `
    <div id='element'>
      <span class="e-avatar e-avatar-xlarge">RT</span>
      <span class="e-avatar e-avatar-large">RT</span>
      <span class="e-avatar">RT</span>
      <span class="e-avatar e-avatar-small">RT</span>
      <span class="e-avatar e-avatar-xsmall">RT</span>
    </div>
  `
})
```

```

  })
  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));

```

Circle

The circle avatar style can be applied by adding the modifier class `e-avatar-circle` to the target element.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { Component } from '@angular/core';
@Component({
  imports: [

  ],
  standalone: true,
  selector: 'my-app',
  template: `
    <div id='element'>
      <span class="e-avatar e-avatar-xlarge e-avatar-circle">SJ</span>
      <span class="e-avatar e-avatar-large e-avatar-circle">SJ</span>
      <span class="e-avatar e-avatar-circle">SJ</span>
      <span class="e-avatar e-avatar-small e-avatar-circle">SJ</span>
      <span class="e-avatar e-avatar-xsmall e-avatar-circle">SJ</span>
    </div>
  `
})
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));

```

How To

Avatar customization in Angular Avatar component

Colour customization

The avatar comes with default background colour (grey). This can be easily customized to desired colour by adding custom class or directly selecting the avatar class from the CSS.

APP.COMPONENT.TS


```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Component } from '@angular/core';
@Component({
  imports: [

    ],
  standalone: true,
  selector: 'my-app',
  template: `
    <div id='element'>
      <span class="e-avatar e-avatar-xlarge e-avatar-circle
green">AJ</span>
      <span class="e-avatar e-avatar-xlarge e-avatar-circle
violet">JK</span>
      <span class="e-avatar e-avatar-xlarge e-avatar-circle
rose">EL</span>
      <span class="e-avatar e-avatar-xlarge e-avatar-circle
blue">SR</span>
      <span class="e-avatar e-avatar-xlarge e-avatar-circle
red">PD</span>
    </div>
  `
})
export class AppComponent { }
```

MAIN.TS

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

Customize avatar sizes

Even though the avatar comes with five predefined sizes, sometimes it's not enough. So, the avatar is designed in such a way that the width and height will be relative to font-size. By changing the font-size of the avatar element, you can change the width and height automatically.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Component } from '@angular/core';
@Component({
  imports: [

    ],
  standalone: true,
  selector: 'my-app',
  template: `
    <div id='element'>
      <span class="e-avatar">26px</span>
      <span class="e-avatar">24px</span>
      <span class="e-avatar">22px</span>
    </div>
  `
})
export class AppComponent { }
```

```

        <span class="e-avatar">20px</span>
        <span class="e-avatar">18px</span>
    </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));

```

Use various media in avatar

Avatars can be used with a wide variety of media formats like SVG, font-icons, images, letters, words, etc. Some of them are given below.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Component } from '@angular/core';
@Component({
  imports: [
    ],
  standalone: true,
  selector: 'my-app',
  template: `
    <div id='element'>
      <div class="sample_container avatar-types">
        <div class="avatar-block">
          <!-- Card Component -->
          <div class="e-card e-avatar-showcase">
            <div class="e-card-content">
              <!-- XLarge Circle Avatar Component -->
              <div class="e-avatar e-avatar-xlarge e-avatar-circle image"></div>
            </div>
            <div class="e-card-content">
              <div>Image</div>
            </div>
          </div>
        </div>
        <div class="avatar-block">
          <!-- Card Component -->
          <div class="e-card e-avatar-showcase">
            <div class="e-card-content">
              <!-- XLarge Circle Avatar Component -->
              <div class="e-avatar e-avatar-xlarge e-avatar-circle">
                <div class="svg_icons chrome"></div>
              </div>
            </div>
          </div>
        </div>
      </div>
    </div>
  `
})

```

```

        <div class="e-card-content">
            <div>SVG</div>
        </div>
    </div>
</div>
<div class="avatar-block">
    <!-- Card Component -->
    <div class="e-card e-avatar-showcase">
        <div class="e-card-content">
            <!-- XLarge Circle Avatar Component -->
            <div class="e-avatar e-avatar-xlarge e-avatar-
circle">GR</div>
        </div>
        <div class="e-card-content">
            <div>Initial</div>
        </div>
    </div>
</div>
<div class="avatar-block">
    <!-- Card Component -->
    <div class="e-card e-avatar-showcase">
        <div class="e-card-content">
            <!-- XLarge Circle Avatar Component -->
            <div class="e-avatar e-avatar-xlarge e-avatar-
circle">

                <div class="e-people icons"></div>
            </div>
        </div>
        <div class="e-card-content">
            <div>FontIcon</div>
        </div>
    </div>
</div>
<div class="avatar-block">
    <!-- Card Component -->
    <div class="e-card e-avatar-showcase">
        <div class="e-card-content">
            <!-- XLarge Circle Avatar Component -->
            <div class="e-avatar e-avatar-xlarge e-avatar-
circle">User</div>
        </div>
        <div class="e-card-content">
            <div>Word</div>
        </div>
    </div>
</div>
<div class="avatar-block">
    <!-- Card Component -->
    <div class="e-card e-avatar-showcase">
        <div class="e-card-content">
            <!-- XLarge Circle Avatar Component -->
            <div class="e-avatar e-avatar-xlarge e-avatar-
circle custom">

                <div class="e-people icons"></div>
            </div>
        </div>
        <div class="e-card-content">

```

```

        <div>Custom</div>
      </div>
    </div>
  </div>
</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));
```

Dynamic avatar rendering from datasource

We can render avatar component dynamically from a data-source. In this sample we have rendered the avatar component

using a data-source which contains the image source in different sizes dynamically. This is applicable also for

data-source from the server or remote data or AJAX. We have also rendered the avatar using CSS property

background-image and using image tag.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { Component } from '@angular/core';
@Component({
  imports: [

  ],
  standalone: true,
  selector: 'my-app',
  template: `
    <div class='control-pane'>
      <div class="sample_container avatar-badge">
        <div class="avatar-block">
          <div class="e-card e-avatar-showcase">
            <div class="e-card-content">
              <div *ngFor="let item of dataSource"
                [style.backgroundImage]='url('+ item.src +)'"
                class="e-avatar e-avatar-circle {{item.size}}">
              </div>
            </div>
            <div class="e-card-content">
              <div>Using <code>background-image</code>
            </div>
          </div>
        </div>
      </div>
    </div>
  `
})
export class AppComponent { }
```

```

        </div>
        <div class="circleAvatar avatar-block">
            <div class="e-card e-avatar-showcase">
                <div class="e-card-content">
                    <div *ngFor="let item of dataSource" class="e-avatar
e-avatar-circle {{item.size}}">
                        <img src= {{item.src}}/>
                    </div>
                </div>
                <div class="e-card-content">
                    <div>Using <code>img</code> tag</div>
                </div>
            </div>
        </div>
    </div>
</div>
`
    })
    export class AppComponent {
        public dataSource: { [key: string]: string }[] | any = [
            { src: 'https://ej2.syncfusion.com/demos/src/grid/images/2.png',
size: 'e-avatar-xsmall' },
            { src: 'https://ej2.syncfusion.com/demos/src/grid/images/2.png',
size: 'e-avatar-small' },
            { src: 'https://ej2.syncfusion.com/demos/src/grid/images/2.png',
size: 'e-avatar' },
            { src: 'https://ej2.syncfusion.com/demos/src/grid/images/2.png',
size: 'e-avatar-large' },
            { src: 'https://ej2.syncfusion.com/demos/src/grid/images/2.png',
size: 'e-avatar-xlarge' }
        ];
    }
}

```

MAIN.TS

```

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

```

Integrate avatar into listview in Angular Avatar component

Avatar is integrated into the listview to create contacts applications. The `xsmall` size avatar is used to match the size of the list item. Letters and images are also used as avatar content.

APP.COMPONENT.TS

```

{% raw %}
import { Component } from '@angular/core';
@Component({
  selector: 'my-app',
  template: `
<div id="letterAvatarList">
  <ejs-listview id='letterAvatarList' [dataSource]='dataSource'
[headerTitle]='headerTitle' [showHeader]='true' [sortOrder]='sortOrder'>
  <ng-template #template let-ds="dataSource">

```

```

<div class="listWrapper">
  <span class="e-avatar e-avatar-small e-avatar-circle {{dataSource.color}}"
    *ngIf="dataSource.avatar !== ''">{{dataSource.avatar}}</span>
  <span class="{{dataSource.pic}} e-avatar e-avatar-small e-avatar-circle"
    *ngIf="dataSource.pic !== '' "> </span>
  <span class="text">{{dataSource.text}}</span>
</div>
</ng-template>
</ejs-listview>
</div>
`
})
export class AppComponent {
  // Listview datasource with avatar and image source fields
  public dataSource?: { [key: string]: Object; }[] = [
    { id: 's_01', text: 'Robert', avatar: '', pic: 'pic04', color: '' },
    { id: 's_02', text: 'Nancy', avatar: 'N', pic: '', color: 'green' },
    { id: 's_05', text: 'John', pic: 'pic01', avatar: '', color: '' },
    { id: 's_03', text: 'Andrew', avatar: 'A', pic: '', color: 'blue' },
    { id: 's_06', text: 'Michael', pic: 'pic02', avatar: '', color: '' },
    { id: 's_07', text: 'Steven', pic: 'pic03', avatar: '', color: '' },
    { id: 's_08', text: 'Margaret', avatar: 'M', pic: '', color: 'red' }
  ];
  public headerTitle: string = 'Contacts';
  public sortOrder: string = 'Ascending';
}
{% enddraw %}

```

MAIN.TS

```

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

```

Integrate avatar into badge in Angular Avatar component

The badge is dependent and supportive component, and it can be used with avatar to create a notification avatar.

The default avatar (.e-avatar) and circle avatar (.e-avatar-circle) have been used with notification badges (.e-badge-notification) in the following sample.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Component } from '@angular/core';
@Component({
  imports: [

  ],
  standalone: true,
  selector: 'my-app',
  template: `
    <div id='element'>

```

```

<div class="sample_container avatar-badge">
  <div class="avatar-block">
    <!-- Card Component -->
    <div class="e-card e-avatar-showcase">
      <div class="e-card-content">
        <div class="avatar-sub-block">
          <!-- xSmall Avatar-->
          <div class="e-avatar e-avatar-xsmall">
            
          </div>
          <!-- Notification Badge -->
          <span class="e-badge e-badge-primary e-badge-overlap
e-badge-notification e-badge-circle">6</span>
        </div>
        <div class="avatar-sub-block">
          <!-- Small Avatar-->
          <div class="e-avatar e-avatar-small">
            
          </div>
          <!-- Notification Badge -->
          <span class="e-badge e-badge-primary e-badge-overlap
e-badge-notification e-badge-circle">12</span>
        </div>
        <div class="avatar-sub-block">
          <!-- Avatar-->
          <div class="e-avatar">
            
          </div>
          <!-- Notification Badge -->
          <span class="e-badge e-badge-primary e-badge-overlap
e-badge-notification">46</span>
        </div>
        <div class="avatar-sub-block">
          <!-- Large Avatar-->
          <div class="e-avatar e-avatar-large">
            
          </div>
          <!-- Notification Badge -->
          <span class="e-badge e-badge-primary e-badge-overlap
e-badge-notification">82</span>
        </div>
        <div class="avatar-sub-block">
          <!-- xLarge Avatar-->
          <div class="e-avatar e-avatar-xlarge">
            
          </div>
          <!-- Notification Badge -->

```

```

        <span class="e-badge e-badge-primary e-badge-overlap
e-badge-notification">99+</span>
    </div>
</div>
</div>
</div>
<div class="circleAvatar avatar-block">
    <!-- Card Component -->
    <div class="e-card e-avatar-showcase">
        <div class="e-card-content">
            <div class="avatar-sub-block">
                <!-- xSmall Circle Avatar-->
                <div class="e-avatar e-avatar-circle e-avatar-
xsmall">
                    
                </div>
                <!-- Notification Badge -->
                <span class="e-badge e-badge-primary e-badge-overlap
e-badge-notification e-badge-circle">6</span>
            </div>
            <div class="avatar-sub-block">
                <!-- Small Circle Avatar-->
                <div class="e-avatar e-avatar-circle e-avatar-small">
                    
                </div>
                <!-- Notification Badge -->
                <span class="e-badge e-badge-primary e-badge-overlap
e-badge-notification e-badge-circle">12</span>
            </div>
            <div class="avatar-sub-block">
                <!-- Circle Avatar-->
                <div class="e-avatar e-avatar-circle">
                    
                </div>
                <!-- Notification Badge -->
                <span class="e-badge e-badge-primary e-badge-overlap
e-badge-notification">46</span>
            </div>
            <div class="avatar-sub-block">
                <!-- Large Circle Avatar-->
                <div class="e-avatar e-avatar-circle e-avatar-large">
                    
                </div>
                <!-- Notification Badge -->
                <span class="e-badge e-badge-primary e-badge-overlap
e-badge-notification">82</span>
            </div>
            <div class="avatar-sub-block">
                <!-- xLarge Circle Avatar-->

```



```
<div class="e-avatar e-avatar-circle e-avatar-  
xlarge">  
  
      
  
    </div>  
    <!-- Notification Badge -->  
    <span class="e-badge e-badge-primary e-badge-overlap  
e-badge-notification">99</span>  
    </div>  
    </div>  
    </div>  
    </div>  
</div>  
,  
  
}))  
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));
```

Badge

Getting started with Angular Badge component

The following section explains the steps required to create a simple **Badge** component using styles and its basic usage.

Setting up angular project

Angular provides the easiest way to set angular CLI projects using Angular CLI tool.

Install the CLI application globally to your machine by using following command.

`sh

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

—

Create a new angular application

`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.

Dependencies

The `Badge` component is pure CSS component which doesn't need specific dependencies to render.

```
`javascript
|-- @syncfusion/ej2-notifications
`
```

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

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

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

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

```
`bash
@syncfusion/ej2-angular-notifications:"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 Badge component, import the Badge 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-notifications/styles/material.css';
@import '../node_modules/@syncfusion/ej2-angular-notifications/styles/material.css';
`
```

Note: To refer the combined component styles, use Syncfusion [CRG](#) (Custom Resource Generator) in your application.

Add Badge into application

Modify the `template` in `app.component.ts` file to render the Badge component.

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

```
`typescript
import { Component } from '@angular/core';
@Component({
  selector: 'my-app',
  template: <div id='element'><h1>Badge Component <span class="e-
  badge">New</span></h1></div>
})
export class AppComponent {}
`
```

Run the application

Run the application in the browser using the following command.

```
`html
npm start
`
```

The following example shows a basic badge component.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Component } from '@angular/core';
@Component({
  imports: [

  ],
  standalone: true,
  selector: 'my-app',
```

```

    template: `<div id='element'><h1>Badge Component <span class="e-
badge">New</span></h1></div>`
  })
  export class AppComponent {}

```

MAIN.TS

```

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

```

You can refer to our [Angular Badge](#) feature tour page for its groundbreaking feature representations. You can also explore our [Angular Badge example](#) to know how to present and manipulate data.

Types in Angular Badge component

This section explains different styles and types of the badges.

Badge styles

The Essential JS 2 Badge has the following predefined styles that can be used with `.e-badge` class to change the appearance of a badge.

Class Name	Description
:-----	:-----
e-badge-primary	Represents a primary notification.
e-badge-secondary	Represents a secondary notification.
e-badge-success	Represents a positive notification.
e-badge-danger	Represents a negative notification.
e-badge-warning	Represents notification with caution.
e-badge-info	Represents an informative notification.
e-badge-light	Represents notification in light variant.
e-badge-dark	Represents notification in dark variant.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { Component } from '@angular/core';
@Component({
  imports: [

  ],
  standalone: true,
  selector: 'my-app',
  template: `
    <div id='element'>
      <div class="sample_container">
        <div class="block" style="display:inline-block">
          <div class="e-card e-badge-showcase">

```

```


.e-badge-primary</code>



.e-badge-secondary</code>



.e-badge-success</code>



.e-badge-danger</code>



.e-badge-warning</code>


```

```
<div class="block" style="display:inline-block">
  <div class="e-card e-badge-showcase">
    <div class="e-card-content">
      <span class="e-badge e-badge-info">Info</span>
    </div>
    <div class="e-card-content">
      <code>.e-badge-info</code>
    </div>
  </div>
</div>
<div class="block" style="display:inline-block">
  <div class="e-card e-badge-showcase">
    <div class="e-card-content">
      <span class="e-badge e-badge-light">Light</span>
    </div>
    <div class="e-card-content">
      <code>.e-badge-light</code>
    </div>
  </div>
</div>
<div class="block" style="display:inline-block">
  <div class="e-card e-badge-showcase">
    <div class="e-card-content">
      <span class="e-badge e-badge-dark">Dark</span>
    </div>
    <div class="e-card-content">
      <code>.e-badge-dark</code>
    </div>
  </div>
</div>
</div>
</div>
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));
```

Badge types

The types of Essential JS 2 badges are as follows:

- Circle
- Pill
- Link
- Notification
- Overlap
- Dot
- Position

Circle

The circle badge style can be applied by adding the modifier class `.e-badge-circle` to the target element.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Component } from '@angular/core';
@Component({
  imports: [
    ],
  standalone: true,
  selector: 'my-app',
  template: `
    <div id='element'>
      <div style="position:relative;display:inline-block;margin:20px
20px 10px 20px;">
        <div class="skype svg_icons"></div>
        <span class="e-badge e-badge-success e-badge-overlap e-badge-
notification e-badge-circle">18</span>
      </div>
      <div style="position:relative;display:inline-block;margin:20px
20px 10px 20px;">
        <div class="twitter svg_icons"></div>
        <span class="e-badge e-badge-info e-badge-overlap e-badge-
notification e-badge-circle">9</span>
      </div>
      <div style="position:relative;display:inline-block;margin:20px
20px 10px 20px;">
        <div class="facebook svg_icons"></div>
        <span class="e-badge e-badge-info e-badge-overlap e-badge-
notification e-badge-circle">2</span>
      </div>
      <div style="position:relative;display:inline-block;margin:20px
20px 10px 20px;">
        <div class="firefox svg_icons"></div>
        <span class="e-badge e-badge-danger e-badge-overlap e-badge-
notification e-badge-circle">35</span>
      </div>
    </div>
  `
})
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));
```

Pill

The pill badge style can be applied by adding the modifier class `.e-badge-pill` to the target element.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Component } from '@angular/core';
@Component({
  imports: [

    ],
  standalone: true,
  selector: 'my-app',
  template: `
    <div id='element'>
      <h1>Badge Component <span class="e-badge e-badge-primary e-badge-pill">New</span></h1>
    </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));
```

Link

When badge modifier classes are applied to the anchor tag, the badge's appearance will change from normal state to hover state on mouseover.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Component } from '@angular/core';
@Component({
  imports: [

    ],
  standalone: true,
  selector: 'my-app',
  template: `
    <div id='element'>
      <div style="display: inline-block; margin-top: 15px;">
        <a href="#" class="e-badge e-badge-primary">Link Badge</a>
      </div>
    </div>
  `
})
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));
```

Notification

The notification badge style can be applied by adding the modifier class `.e-badge-notification` to the target element. Notification badges are used when a content or a context needs special attention.

While using the notification badge, set the parent element to `position: relative`.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Component } from '@angular/core';
@Component({
  imports: [

    ],
  standalone: true,
  selector: 'my-app',
  template: `
    <div id='element'>
      <div style="position:relative;display:inline-block;margin:20px
20px 10px 20px;">
        <div class="skype svg_icons"></div>
        <span class="e-badge e-badge-success e-badge-overlap e-badge-
notification">99</span>
      </div>
      <div style="position:relative;display:inline-block;margin:20px
20px 10px 20px;">
        <div class="twitter svg_icons"></div>
        <span class="e-badge e-badge-info e-badge-overlap e-badge-
notification">27</span>
      </div>
      <div style="position:relative;display:inline-block;margin:20px
20px 10px 20px;">
        <div class="facebook svg_icons"></div>
        <span class="e-badge e-badge-info e-badge-overlap e-badge-
notification">2</span>
      </div>
      <div style="position:relative;display:inline-block;margin:20px
20px 10px 20px;">
        <div class="firefox svg_icons"></div>
        <span class="e-badge e-badge-danger e-badge-overlap e-badge-
notification">35</span>
      </div>
    </div>
  `
})
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));
```

Dot

Dot can be applied by adding the modifier class `.e-badge-dot` to the target element. Dot badges are similar to notification badges, but in a minimalistic way. While using the dot badge, set the parent element to `position: relative`.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Component } from '@angular/core';
@Component({
  imports: [

  ],
  standalone: true,
  selector: 'my-app',
  template: `
    <div id='element'>
      <div style="position:relative;display:inline-block;margin:20px
20px 10px 20px;">
        <div class="skype svg_icons"></div>
        <span class="e-badge e-badge-success e-badge-overlap e-badge-
dot"></span>
      </div>
      <div style="position:relative;display:inline-block;margin:20px
20px 10px 20px;">
        <div class="twitter svg_icons"></div>
        <span class="e-badge e-badge-info e-badge-overlap e-badge-
dot"></span>
      </div>
      <div style="position:relative;display:inline-block;margin:20px
20px 10px 20px;">
        <div class="facebook svg_icons"></div>
        <span class="e-badge e-badge-info e-badge-overlap e-badge-
dot"></span>
      </div>
      <div style="position:relative;display:inline-block;margin:20px
20px 10px 20px;">
        <div class="firefox svg_icons"></div>
        <span class="e-badge e-badge-danger e-badge-overlap e-badge-
dot"></span>
      </div>
    </div>
  `
})
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));
```

Overlap

The overlap badge can be used with **notification** or **dot** badge, which overlaps with the target -element by adding the modifier class **.e-badge-overlap**. While using the overlap badge, set the parent element to **position: relative**.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Component } from '@angular/core';
@Component({
  imports: [
    ],
  standalone: true,
  selector: 'my-app',
  template: `
    <div id='element'>
      <div style="position:relative;display:inline-block;margin:20px 20px 10px 20px;">
        <div class="skype svg_icons"></div>
        <span class="e-badge e-badge-success e-badge-overlap e-badge-notification">99+</span>
      </div>
      <div style="position:relative;display:inline-block;margin:20px 20px 10px 20px;">
        <div class="twitter svg_icons"></div>
        <span class="e-badge e-badge-info e-badge-overlap e-badge-notification">27</span>
      </div>
      <div style="position:relative;display:inline-block;margin:20px 20px 10px 20px;">
        <div class="facebook svg_icons"></div>
        <span class="e-badge e-badge-info e-badge-overlap e-badge-notification">2</span>
      </div>
      <div style="position:relative;display:inline-block;margin:20px 20px 10px 20px;">
        <div class="firefox svg_icons"></div>
        <span class="e-badge e-badge-danger e-badge-overlap e-badge-notification">35</span>
      </div>
    </div>
  `
})
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));
```

Position

The default position of the **notification** or **dot** badge is top. But, the position can be changed to **bottom** using the modifier class **.e-badge-bottom**. For example, the bottom class modifier is used with dot badge to display the status in the avatar as shown in the following sample.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Component } from '@angular/core';
@Component({
  imports: [

  ],
  standalone: true,
  selector: 'my-app',
  template: `
    <div id='element'>
      <div class="badge-block">
        <div class="contact svg_icons"></div>
        <!-- Success Colored Bottom Dot Badge -->
        <span class="e-badge e-badge-success e-badge-overlap e-badge-dot e-badge-bottom"></span>
      </div>
      <div class="badge-block">
        <div class="skype svg_icons"></div>
        <!-- Info Colored Bottom Dot Badge -->
        <span class="e-badge e-badge-info e-badge-overlap e-badge-dot e-badge-bottom"></span>
      </div>
      <div class="badge-block">
        <div class="facebook svg_icons"></div>
        <!-- Info Colored Dot Badge -->
        <span class="e-badge e-badge-info e-badge-overlap e-badge-dot"></span>
      </div>
      <div class="badge-block">
        <div class="pinterest svg_icons"></div>
        <!-- Danger Colored Dot Badge -->
        <span class="e-badge e-badge-danger e-badge-overlap e-badge-dot e-badge-bottom"></span>
      </div>
    </div>
  `
})
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));
```

How To

Badge customization in Angular Badge component

Colour customization

Even though badges come with 8 predefined colors, you can also customize the colour of the badge as desired.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Component } from '@angular/core';
@Component({
  imports: [
    ],
  standalone: true,
  selector: 'my-app',
  template: `
    <div id='element'>
      <h1>Color Customization <span class="e-badge e-badge-primary e-
badge-pill green">New</span></h1>
      <h1>Color Customization <span class="e-badge e-badge-primary e-
badge-pill blue">New</span></h1>
      <h1>Color Customization <span class="e-badge e-badge-primary e-
badge-pill purple">New</span></h1>
      <h1>Color Customization <span class="e-badge e-badge-primary e-
badge-pill gradient">New</span></h1>
    </div>
  `
})
export class AppComponent { }
```

MAIN.TS

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

Customize badge size

Badges are designed to change its size based on the content. To change the size of a badge, adjust the font size of the badge.

APP.COMPONENT.TS

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

```
imports: [
  ],
standalone: true,
selector: 'my-app',
template: `
<div id='element'>
  <h1>Badge Component <span class="e-badge e-badge-primary
size_1">New</span></h1>
  <h1>Badge Component <span class="e-badge e-badge-primary
size_2">New</span></h1>
  <h1>Badge Component <span class="e-badge e-badge-primary
size_3">New</span></h1>
</div>
`
))
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 position

Even though the badges support the conventional **top** and **bottom** positions, the position of the badges can be changed as desired.

This can be done by adding a custom class to the badge element to override the default position applied from the source.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Component } from '@angular/core';
@Component({
  imports: [
    ],
  standalone: true,
  selector: 'my-app',
  template: `
    <div id='element'>
      <div style="width: 200px;margin: 10px auto;">
        <div class="badge-block">
          <div class="whatsapp svg_icons"></div>
          <!-- Warning Colored Notification Badge -->
          <span class="e-badge e-badge-warning e-badge-notification
e-badge-overlap leftTop">99+</span>
        </div>
        <div class="badge-block">
          <div class="facebook svg_icons"></div>
          <!-- Danger Colored Notification Badge -->
        </div>
      </div>
    </div>
  `
})
export class AppComponent { }
```

```

        <span class="e-badge e-badge-danger e-badge-notification
e-badge-overlap leftTop">99+</span>
      </div>
      <div class="badge-block">
        <div class="skype svg_icons"></div>
        <!-- Secondary Colored Notification Badge -->
        <span class="e-badge e-badge-secondary e-badge-
notification e-badge-overlap leftTop">18</span>
      </div>
    </div>
    <div style="width: 200px;margin: 10px auto;">
      <div class="badge-block">
        <div class="whatsapp svg_icons"></div>
        <!-- Warning Colored Notification Badge -->
        <span class="e-badge e-badge-warning e-badge-notification
e-badge-overlap leftBottom">99+</span>
      </div>
      <div class="badge-block">
        <div class="facebook svg_icons"></div>
        <!-- Danger Colored Notification Badge -->
        <span class="e-badge e-badge-danger e-badge-notification
e-badge-overlap leftBottom">99+</span>
      </div>
      <div class="badge-block">
        <div class="skype svg_icons"></div>
        <!-- Secondary Colored Notification Badge -->
        <span class="e-badge e-badge-secondary e-badge-
notification e-badge-overlap leftBottom">18</span>
      </div>
    </div>
  </div>
  ,
})
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));

```

Integrate badge into listview in Angular Badge component

The badges can be integrated with the `listview` component to indicate new notification with colour based on priority.

In the following sample, `default` badges are used and there is no need to customize the badge size. The component will automatically adjust the size based on the container element.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ListViewModule } from '@syncfusion/ej2-angular-lists'
import { Component } from '@angular/core';

```

```

@Component({
  imports: [

    ListViewModule
  ],
  standalone: true,
  selector: 'my-app',
  template: `
    <div class="sample_container badge-list">
      <ejs-listview id='lists' [dataSource]='dataSource' [fields]='fields'
headerTitle='Inbox' [showHeader]='true'>
        <ng-template #template let-dataSource="">
          <div class="listWrapper" style="width: inherit; height:
inherit;">
            <span class="{{dataSource.icons}} list_svg">&#160;</span>
            <span class="list_text">{{dataSource.text}} </span>
            <span class="{{dataSource.badge}}" style="float:
right;margin-top: 16px;font-size: 12px;">{{dataSource.messages}}</span>
          </div>
        </ng-template>
      </ejs-listview>
    </div>
  `
})
export class AppComponent {
  // Datasource for listview, badge field is the class data for Badges
  public dataSource: { [key: string]: Object }[] = [
    { id: 'p_01', text: 'Primary', messages: '3 New', badge: 'e-badge e-
badge-primary', icons: 'primary', type: 'Primary' },
    { id: 'p_02', text: 'Social', messages: '27 New', badge: 'e-badge e-
badge-secondary', icons: 'social', type: 'Primary' },
    { id: 'p_03', text: 'Promotions', messages: '7 New', badge: 'e-badge
e-badge-success', icons: 'promotion', type: 'Primary' },
    { id: 'p_04', text: 'Updates', messages: '13 New', badge: 'e-badge e-
badge-info', icons: 'updates', type: 'Primary' },
    { id: 'p_05', text: 'Starred', messages: '', badge: '', icons:
'starred', type: 'All Labels' },
    { id: 'p_06', text: 'Important', messages: '2 New', badge: 'e-badge
e-badge-danger', icons: 'important', type: 'All Labels' },
    { id: 'p_07', text: 'Sent', messages: '', badge: '', icons: 'sent',
type: 'All Labels' },
    { id: 'p_08', text: 'Outbox', messages: '', badge: '', icons:
'outbox', type: 'All Labels' },
    { id: 'p_09', text: 'Drafts', messages: '7 New', badge: 'e-badge e-
badge-warning', icons: 'draft', type: 'All Labels' },
  ];
  // Map fields
  public fields: object = { groupBy: 'type' };
}

```

MAIN.TS

```

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

```


Dynamic badge content in Angular Badge component

Badges in real-time needs to be updated dynamically based on the requirements. In this sample, using Angular data binding the badges content will be updated dynamically. Click the increment button to change the badge value.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ListViewModule } from '@syncfusion/ej2-angular-lists'
import { Component } from '@angular/core';
@Component({
  imports: [
    ListViewModule
  ],
  standalone: true,
  selector: 'my-app',
  template: `
    <div class="sample_container badge-list">
      <ejs-listview id='lists' [dataSource]='dataSource' [fields]='fields'
headerTitle='Inbox' [showHeader]='true'>
        <ng-template #template let-dataSource="">
          <div class="listWrapper" style="width: inherit; height:
inherit;">
            <span class="{{dataSource.icons}} list_svg">&#160;</span>
            <span class="list_text">{{dataSource.text}} </span>
            <span class="{{dataSource.badge}}">
*ngIf="dataSource.badge !== ' ' " style="float: right; margin-top: 16px; font-
size: 12px;">{{badge[dataSource.text]}} New</span>
          </div>
        </ng-template>
      </ejs-listview>
      <p class='crossline'></p>
      <span class='incr_button'>
        <button class='e-btn e-primary' (click)="onClick()">Increment Badge
Count</button>
      </span>
    </div>
  `
})
export class AppComponent {
  public badge = {
    Primary: 3,
    Social: 27,
    Promotions: 7,
    Updates: 13,
    Important: 2,
    Drafts: 7
  } as any;
  // Datasource for listview, badge field is the class data for Badges
  public dataSource: { [key: string]: Object }[] = [
    { id: 'p_01', text: 'Primary', badge: 'e-badge e-badge-primary',
icons: 'primary', type: 'Primary' },
```

```

    { id: 'p_02', text: 'Social', badge: 'e-badge e-badge-secondary',
icons: 'social', type: 'Primary' },
    { id: 'p_03', text: 'Promotions', badge: 'e-badge e-badge-success',
icons: 'promotion', type: 'Primary' },
    { id: 'p_04', text: 'Updates', badge: 'e-badge e-badge-info', icons:
'updates', type: 'Primary' },
    { id: 'p_05', text: 'Starred', badge: '', icons: 'starred', type:
'All Labels' },
    { id: 'p_06', text: 'Important', badge: 'e-badge e-badge-danger',
icons: 'important', type: 'All Labels' },
    { id: 'p_07', text: 'Sent', badge: '', icons: 'sent', type: 'All
Labels' },
    { id: 'p_08', text: 'Outbox', badge: '', icons: 'outbox', type: 'All
Labels' },
    { id: 'p_09', text: 'Drafts', badge: 'e-badge e-badge-warning',
icons: 'draft', type: 'All Labels' },
  ];
  // Map fields
  public fields: object = { groupBy: 'type' };
  public onClick() {
    let badgeKeys = Object.keys(this.badge);
    for (let badge of badgeKeys) {
      this.badge[badge]++;
    }
  }
}

```

MAIN.TS

```

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

```

Barcode

Getting started with Angular Barcode component

This section explains you the steps required to create a simple barcode and demonstrate the basic usage of the barcode 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 Barcode Generator 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-barcode-generator](#) package to the application.

```
`bash
```

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

```
`bash
```

```
npm install @syncfusion/ej2-angular-barcode-generator@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-barcode-generator:"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 Syncfusion Barcode Generator package

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

To install Barcode Generator component, use the following command.

```
`bash
```

```
npm install @syncfusion/ej2-angular-barcode-generator --save
```

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

Registering Barcode Generator Module

Import Barcode Generator module into Angular application(app.module.ts) from the package **@syncfusion/ej2-angular-barcode-generator** [src/app/app.module.ts].

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

import { BarcodeGeneratorAllModule,QRCodeGeneratorAllModule,DataMatrixGeneratorAllModule }
from '@syncfusion/ej2-angular-barcode-generator';
import { AppComponent } from './app.component';

@NgModule({
  //declaration of ej2-angular-Barcode Generator module into NgModule
  imports: [ BrowserModule, BarcodeGeneratorAllModule, QRCodeGeneratorAllModule
,DataMatrixGeneratorAllModule ],
  declarations: [ AppComponent ],
  bootstrap: [ AppComponent ]
})
export class AppModule { }
```

Adding Barcode Generator control

You can start adding Essential JS 2 barcode-generator component to the application. To get started, add the Angular Barcode component in **app.ts** and **index.html** files using the following code.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import {
BarcodeGeneratorAllModule,QRCodeGeneratorAllModule,DataMatrixGeneratorAllModule }
from '@syncfusion/ej2-angular-barcode-generator'
import { Component } from '@angular/core';
@Component({
  imports: [
    BarcodeGeneratorAllModule, QRCodeGeneratorAllModule
,DataMatrixGeneratorAllModule
  ],
  providers: [ ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the barcode generator component
```

```

    template: `<ejs-barcodegenerator style="display: block;" #barcode
id="barcode" width="200px" height="150px" mode="SVG" type="Codabar"
value="123456789">`
  })
  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));

```

Adding QR Generator control

You can add the QR code in our Angular Barcode Generator component.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import {
BarcodeGeneratorAllModule, QRCodeGeneratorAllModule, DataMatrixGeneratorAllModule
} from '@syncfusion/ej2-angular-barcode-generator'
import { Component } from '@angular/core';
@Component({
  imports: [
    BarcodeGeneratorAllModule, QRCodeGeneratorAllModule
    ,DataMatrixGeneratorAllModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  // specifies the template string for the barcode generator component
  template: `<ejs-qrcodegenerator style="display: block;" #barcode
id="barcode" width="200px" height="150px" mode="SVG" value="Syncfusion"></ejs-
qrcodegenerator>`
})
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));

```

Adding Datamatrix Generator control

You can add the datamatrix code in our Angular Barcode Generator component.

APP.COMPONENT.TS

```

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

```

```
import {
  BarcodeGeneratorAllModule, QRCodeGeneratorAllModule, DataMatrixGeneratorAllModule } from '@syncfusion/ej2-angular-barcode-generator'
import { Component } from '@angular/core';
@Component({
  imports: [
    BarcodeGeneratorAllModule, QRCodeGeneratorAllModule, DataMatrixGeneratorAllModule
  ],
  providers: [],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the barcode generator component
  template: `<ejs-datamatrixgenerator style="display: block;" #barcode id="barcode" width="200px" height="200px" mode="SVG" type="DataMatrix" value="Syncfusion">
    </ejs-datamatrixgenerator>`
})
export class AppComponent {}
```

MAIN.TS

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

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

BarcodeGenerator in Angular Barcode component

Code39

The Code 39 character set includes the digits 0-9, the letters A-Z (upper case only), and the symbols: space, minus (-), plus (+), period (.), dollar sign (\$), slash (/), and percent (%). A special start / stop character is placed at the beginning and ending of each barcode. The barcode can be of any length; even more than 25 characters begin to push the bounds. Code 39 is the only type of barcode that does not require a checksum for common use.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import {
  BarcodeGeneratorAllModule, QRCodeGeneratorAllModule, DataMatrixGeneratorAllModule } from '@syncfusion/ej2-angular-barcode-generator'
import { Component } from '@angular/core';
@Component({
  imports: [
    BarcodeGeneratorAllModule, QRCodeGeneratorAllModule, DataMatrixGeneratorAllModule
  ],
  providers: [],
  standalone: true,
```

```

    selector: "app-container",
    // specifies the template string for the barcode generator component
    template: ` <ejs-barcodegenerator style="display: block;" #barcode
id="barcode" width="200px" height="150px"mode="SVG" type="Code39"
value="SYNCFUSION"></ejs-barcodegenerator> `
  })
}
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));

```

Code39 Extended

Code 39 Extended is an extended version of Code 39 that supports ASCII character set. In Code 39 Extended, you can also code 26 lower letters (a-z) and the special characters in the keyboard.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import {
BarcodeGeneratorAllModule, QRCodeGeneratorAllModule, DataMatrixGeneratorAllModule
} from '@syncfusion/ej2-angular-barcode-generator'
import { Component } from '@angular/core';
@Component({
  imports: [
    BarcodeGeneratorAllModule, QRCodeGeneratorAllModule
    ,DataMatrixGeneratorAllModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  // specifies the template string for the barcode generator component
  template: `<ejs-barcodegenerator style="display: block;" #barcode
id="barcode"
width="200px"height="150px"mode="SVG"type="Code39Extension"value="SYNCFUSION"
><ejs-barcodegenerator> `
  })
}
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));

```

Code 11

Code 11 is used primarily for labeling the telecommunication equipment's. The character set includes the digits 0 to 9, a dash (-), and a start / stop code.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import {
  BarcodeGeneratorAllModule, QRCodeGeneratorAllModule, DataMatrixGeneratorAllModule
} from '@syncfusion/ej2-angular-barcode-generator'
import { Component } from '@angular/core';
@Component({
  imports: [
    BarcodeGeneratorAllModule, QRCodeGeneratorAllModule
    ,DataMatrixGeneratorAllModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  // specifies the template string for the barcode generator component
  template: `<ejs-barcodegenerator style="display: block;" #barcode
id="barcode"
width="200px"height="150px"mode="SVG"type="Code11"value="112"><ejs-
barcodegenerator>`
})
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));
```

Codabar

Codabar is a variable length symbol that encodes the following 20 characters:

0123456789-\$./+ABCD

The characters, A, B, C and D are used as start and stop characters. Codabar is used in libraries, blood banks, the package delivery industry and a variety of other information processing applications.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import {
  BarcodeGeneratorAllModule, QRCodeGeneratorAllModule, DataMatrixGeneratorAllModule
} from '@syncfusion/ej2-angular-barcode-generator'
import { Component } from '@angular/core';
@Component({
  imports: [
    BarcodeGeneratorAllModule, QRCodeGeneratorAllModule
    ,DataMatrixGeneratorAllModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  // specifies the template string for the barcode generator component
```



```

    template: `<ejs-barcodegenerator style="display: block;" #barcode
    id="barcode"
    width="200px"height="150px"mode="SVG"type="Codabar"value="123456789"><ejs-
    barcodegenerator>`
  })
  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));

```

Code 32

Code 32 is mainly used for coding pharmaceuticals, cosmetics and dietetics. It is often used to encode Italian Pharmacode that has the following structure:

- 'A' character (ASCII 65), that is not really encoded.
- 8 digits for Pharmacode (It generally begins with / and prefixed with 0).
- 1 digit for checksum module 10, that is automatically calculated by barcode.

The value to be encoded must be 8 digits Pharmacode (prefix it with '0' if necessary) and the 9th digit (the checksum) is automatically calculated by barcode.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import {
  BarcodeGeneratorAllModule, QRCodeGeneratorAllModule, DataMatrixGeneratorAllModule
} from '@syncfusion/ej2-angular-barcode-generator'
import { Component } from '@angular/core';
@Component({
  imports: [
    BarcodeGeneratorAllModule, QRCodeGeneratorAllModule
    ,DataMatrixGeneratorAllModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  // specifies the template string for the barcode generator component
  template: `<ejs-barcodegenerator style="display: block;" #barcode
  id="barcode" width="200px" height="150px"mode="SVG" type="Code32"
  value="01234567"></ejs-barcodegenerator>`
})
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));
```

Code 93

Code 93 is designed to complement and improve upon Code 39. It can represent the entire ASCII character set by using combinations of 2 characters. Code 93 is a continuous, variable-length symbology and produces denser code. The Standard Mode (default implementation) can encode uppercase letters (A-Z), digits (0-9), and special characters like *, -, \$, %, (Space), ., /, and +.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import {
  BarcodeGeneratorAllModule, QRCodeGeneratorAllModule, DataMatrixGeneratorAllModule
} from '@syncfusion/ej2-angular-barcode-generator'
import { Component } from "@angular/core";
@Component({
  imports: [
    BarcodeGeneratorAllModule, QRCodeGeneratorAllModule
    , DataMatrixGeneratorAllModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  // specifies the template string for the barcode generator component
  template: `<ejs-barcodegenerator style="display: block;" #barcode
id="barcode" width="200px" height="150px" mode="SVG" type="Code93"
value="01234567"></ejs-barcodegenerator>`
})
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));
```

Code 93 Extended

The Code 93 Extended Barcode symbology is continuous, variable length, and self-checking. It is based on Code 93 Barcode. The Extended Version can encode all 128 ASCII characters.

Code 128

Code 128 is a variable length, high density, alphanumeric, linear bar code symbology, capable of encoding the full 128-character ASCII character set and extended character sets. This symbology includes a checksum digit for verification and the barcode can also be verified character-by-character by verifying the parity of each data byte.

Code 128 Code Sets

- Code Set A (or Chars Set A) includes all of the standard upper case U.S. alphanumeric keyboard characters and punctuation characters along with the control characters, (namely, characters with ASCII values from 0 to 95 inclusive), and seven special characters.

- Code Set B (or Chars Set B) includes all of the standard upper case alphanumeric keyboard characters and punctuation characters along with the lower case alphabetic characters (namely, characters with ASCII values from 32 to 127 inclusive), and seven special characters.
- Code Set C (or Chars Set C) includes the set of 100 digit pairs from 00 to 99 inclusive along with three special characters. This allows numeric data to be encoded as two data digits per symbol character, at effectively twice the density of standard data.

Code 128 Special characters

The last seven characters of Code Sets A and B (character values 96 - 102) and the last three characters of Code Set C (character values 100 - 102) are special non-data characters with no ASCII character equivalents that have a particular significance to the Barcode reading device.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import {
  BarcodeGeneratorAllModule, QRCodeGeneratorAllModule, DataMatrixGeneratorAllModule
} from '@syncfusion/ej2-angular-barcode-generator'
import { Component } from "@angular/core";
@Component({
  imports: [
    BarcodeGeneratorAllModule, QRCodeGeneratorAllModule
    ,DataMatrixGeneratorAllModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  // specifies the template string for the barcode generator component
  template: `<ejs-barcodegenerator style="display: block;" #barcode
id="barcode" width="200px" height="150px" mode="SVG" type="Code128"
value="SYNCFUSION"></ejs-barcodegenerator>`
})
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));
```

Customizing the Barcode color

A page or printed media with barcode often appears colorful in the background and surrounding region with other contents. In such cases the barcode can also be customized to suit the needs. You can achieve this by using foreground color property .

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import {
  BarcodeGeneratorAllModule, QRCodeGeneratorAllModule, DataMatrixGeneratorAllModule
} from '@syncfusion/ej2-angular-barcode-generator'
```

```
import { Component } from "@angular/core";
@Component({
  imports: [
    BarcodeGeneratorAllModule, QRCodeGeneratorAllModule
    ,DataMatrixGeneratorAllModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  // specifies the template string for the barcode generator component
  template: `<ejs-barcodegenerator style="display: block;" #barcode
id="barcode" width="200px" foreColor="red" height="150px"mode="SVG"
type="Code128" value="SYNCFUSION"></ejs-barcodegenerator>`
})
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));
```

Customizing the Barcode dimension

The dimension of the barcode can be changed using the height and width property of the barcodegenerator.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import {
BarcodeGeneratorAllModule,QRCodeGeneratorAllModule,DataMatrixGeneratorAllModule
} from '@syncfusion/ej2-angular-barcode-generator'
import { Component } from "@angular/core";
@Component({
  imports: [
    BarcodeGeneratorAllModule, QRCodeGeneratorAllModule
    ,DataMatrixGeneratorAllModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  // specifies the template string for the barcode generator component
  template: `<ejs-barcodegenerator style="display: block;" #barcode
id="barcode" width="300px" height="300px"mode="SVG" type="Code128"
value="SYNCFUSION"></ejs-barcodegenerator>`
})
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));
```

Customizing the text

In barcode generators you can customize the barcode text by using display text property .

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import {
  BarcodeGeneratorAllModule, QRCodeGeneratorAllModule, DataMatrixGeneratorAllModule
} from '@syncfusion/ej2-angular-barcode-generator'
import { Component, ViewChild, ViewEncapsulation } from '@angular/core';
import { BarcodeGeneratorComponent, DisplayTextModel } from '@syncfusion/ej2-angular-barcode-generator';
@Component({
  imports: [
    BarcodeGeneratorAllModule, QRCodeGeneratorAllModule,
    DataMatrixGeneratorAllModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  // specifies the template string for the barcode generator component
  template: `<ejs-barcodegenerator style="display: block;" #barcode
id="barcode" width="200px" height="150px" [displayText] = 'displayText'
mode="SVG" type="Code128" value="SYNCFUSION"></ejs-barcodegenerator>`,
})
export class AppComponent {
  // @ViewChild('barcode')
  @ViewChild('displayText')
  public displayText?: DisplayTextModel;
  ngOnInit(): void {
    this.displayText = {
      text: 'text'
    }
  }
}
```

MAIN.TS

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

Qrcodegenerator in Angular Barcode component

QR Code

A QR Code is a two-dimensional barcode that consists of a grid of dark and light dots or blocks that form a square. The data encoded in the barcode can be numeric, alphanumeric, or Shift Japanese Industrial Standards (JIS8) characters. The QR Code uses version from 1 to 40. Version 1 measures 21 modules x 21 modules, Version 2 measures 25 modules x 25 modules, and so on. The number of modules increases in steps of 4 modules per side up to Version 40 that measures 177 modules x 177 modules. Each version

has its own capacity. By default, the barcode control automatically set the version according to the length of the input text. The QR Barcodes are designed for industrial uses and also commonly used in consumer advertising.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import {
  BarcodeGeneratorAllModule, QRCodeGeneratorAllModule, DataMatrixGeneratorAllModule
} from '@syncfusion/ej2-angular-barcode-generator'
import { Component } from "@angular/core";
@Component({
  imports: [
    BarcodeGeneratorAllModule, QRCodeGeneratorAllModule
    ,DataMatrixGeneratorAllModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  // specifies the template string for the barcode generator component
  template: `<ejs-qrcodegenerator style="display: block;" #barcode
id="barcode" width="200px" height="150px" mode="SVG"
value="Syncfusion"></ejs-qrcodegenerator>`
})
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));
```

Customizing the Barcode color

A page or printed media with barcode often appears colorful in the background and surrounding region with other contents. In such cases the barcode can also be customized to suit the needs. You can achieve this by using for forecolor property .

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import {
  BarcodeGeneratorAllModule, QRCodeGeneratorAllModule, DataMatrixGeneratorAllModule
} from '@syncfusion/ej2-angular-barcode-generator'
import { Component } from "@angular/core";
@Component({
  imports: [
    BarcodeGeneratorAllModule, QRCodeGeneratorAllModule
    ,DataMatrixGeneratorAllModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
```

```
// specifies the template string for the barcode generator component
template: `<ejs-qrcodegenerator style="display: block;" #barcode
id="barcode" width="200px" foreColor="red" height="150px" mode="SVG"
value="SynCFusion"></ejs-qrcodegenerator>`
})
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));
```

Customizing the Barcode dimension

The dimension of the barcode can be changed using the height and width properties of the barcodegenerator.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import {
BarcodeGeneratorAllModule, QRCodeGeneratorAllModule, DataMatrixGeneratorAllModule
} from '@syncfusion/ej2-angular-barcode-generator'
import { Component } from '@angular/core';
@Component({
imports: [
BarcodeGeneratorAllModule, QRCodeGeneratorAllModule
, DataMatrixGeneratorAllModule
],
providers: [ ],
standalone: true,
selector: "app-container",
// specifies the template string for the barcode generator component
template: `<ejs-qrcodegenerator style="display: block;" #barcode
id="barcode" width="300px" height="300px" mode="SVG"
value="SynCFusion"></ejs-qrcodegenerator>`
})
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));
```

Customizing the text

In barcode generators You can customize the barcode text by using display text property .

APP.COMPONENT.TS

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

```

import { BrowserModule } from '@angular/platform-browser'
import {
BarcodeGeneratorAllModule,QRCodeGeneratorAllModule,DataMatrixGeneratorAllModu
le } from '@syncfusion/ej2-angular-barcode-generator'
import { Component ,ViewChild} from "@angular/core";
import { BarcodeGeneratorComponent,DisplayTextModel } from '@syncfusion/ej2-
angular-barcode-generator';
@Component({
imports: [
BarcodeGeneratorAllModule, QRCodeGeneratorAllModule
,DataMatrixGeneratorAllModule
],
providers: [ ],
standalone: true,
selector: "app-container",
// specifies the template string for the barcode generator component
template: `<ejs-qrcodegenerator style="display: block;" #barcode
id="barcode" width="200px" height="150px" [displayText] = 'displayText'
mode="SVG" value="Syncfusion"></ejs-qrcodegenerator>`
})
export class AppComponent {
// @ViewChild('barcode')
@ViewChild('displayText')
public displayText?: DisplayTextModel;
ngOnInit(): void {
this.displayText = {
text: 'text'
}
}
}

```

MAIN.TS

```

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

```

Datamatrixgenerator in Angular Barcode component

Data Matrix

DataMatrix Barcode is a two dimensional barcode that consists of a grid of dark and light dots or blocks forming square or rectangular symbol. The data encoded in the barcode can either be numbers or alphanumeric. They are widely used in printed media such as labels and letters. You can read it easily with the help of a barcode reader and mobile phones.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import {
BarcodeGeneratorAllModule,QRCodeGeneratorAllModule,DataMatrixGeneratorAllModu
le } from '@syncfusion/ej2-angular-barcode-generator'
import { Component } from "@angular/core";
@Component({

```



```
imports: [
    BarcodeGeneratorAllModule, QRCodeGeneratorAllModule
  ],
providers: [ ],
standalone: true,
selector: "app-container",
// specifies the template string for the barcode generator component
template: `<ejs-datamatrixgenerator style="display: block;" #barcode
id="barcode" width="200px" height="200px" mode="SVG" type="DataMatrix"
value="Syncfusion"></ejs-datamatrixgenerator>`
})
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));
```

Customizing the Barcode color

A page or printed media with barcode often appears colorful in the background and surrounding region with other contents. In such cases the barcode can also be customized to suit the needs. You can achieve this by using the forecolor property.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import {
  BarcodeGeneratorAllModule, QRCodeGeneratorAllModule, DataMatrixGeneratorAllModule
} from '@syncfusion/ej2-angular-barcode-generator'
import { Component } from '@angular/core';
@Component({
  imports: [
    BarcodeGeneratorAllModule, QRCodeGeneratorAllModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  // specifies the template string for the barcode generator component
  template: `<ejs-datamatrixgenerator style="display: block;" #barcode
id="barcode" width="200px" foreColor="red" height="200px" mode="SVG"
type="DataMatrix" value="Syncfusion"></ejs-datamatrixgenerator>`
})
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));
```

Customizing the Barcode dimension

The dimension of the barcode can be changed using the height and width property of the barcodegenerator.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import {
  BarcodeGeneratorAllModule, QRCodeGeneratorAllModule, DataMatrixGeneratorAllModule
} from '@syncfusion/ej2-angular-barcode-generator'
import { Component } from '@angular/core';
@Component({
  imports: [
    BarcodeGeneratorAllModule, QRCodeGeneratorAllModule,
    DataMatrixGeneratorAllModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
  // specifies the template string for the barcode generator component
  template: `<ejs-datamatrixgenerator style="display: block;" #barcode
id="barcode" width="300px" height="300px" mode="SVG" type="DataMatrix"
value="Syncfusion"></ejs-datamatrixgenerator>`
})
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));
```

Customizing the text

In barcode generators you can customize the barcode text by using the display text property .

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import {
  BarcodeGeneratorAllModule, QRCodeGeneratorAllModule, DataMatrixGeneratorAllModule
} from '@syncfusion/ej2-angular-barcode-generator'
import { Component, ViewChild, ViewEncapsulation } from '@angular/core';
import { DisplayTextModel } from '@syncfusion/ej2-angular-barcode-generator';
@Component({
  imports: [
    BarcodeGeneratorAllModule, QRCodeGeneratorAllModule,
    DataMatrixGeneratorAllModule
  ],
  providers: [ ],
  standalone: true,
  selector: "app-container",
```

```
// specifies the template string for the barcode generator component
template: `<ejs-datamatrixgenerator style="display: block;" #barcode
id="barcode" width="200px" height="200px" [displayText] = 'displayText'
mode="SVG" type="DataMatrix" value="Syncfusion"></ejs-datamatrixgenerator>`
})
export class AppComponent {
  // @ViewChild('barcode')
  @ViewChild('displayText')
  public displayText?: DisplayTextModel;
  ngOnInit(): void {
    this.displayText = {
      text: 'text'
    }
  }
}
```

MAIN.TS

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

Export in Angular Barcode component

Export barcode as an image and base64 string is common for barcode,QRcode and datamatrix. The following code samples explain how to export the barcode as an image and base64 string.

Export

Barcode provides the support to export its content as an image in the specified image type and downloads it in the browser.

The following code example shows how to export the barcode as an image

`typescript

```
import { Component,ViewChild } from '@angular/core';
import { BarcodeGeneratorComponent } from '@syncfusion/ej2-angular-barcode-generator';
import { ClickEventArgs, } from '@syncfusion/ej2-navigations';
@Component({
  selector: "app-container",
  // specifies the template string for the barcode generator component
  template: `
<div class="col-xs-12 col-sm-12 col-lg-6 col-md-6">
<button type="button" (click)="onClick()" id="btn">Export</button>
</div>
<div>
```

```

<ejs-barcodegenerator style="display: block;" #barcode id="barcode" width="200px" height="150px"
mode="SVG" type="Codabar" value="123456789">
</div>`
})
export class AppComponent {
  @ViewChild('barcode')
  public barcode: BarcodeGeneratorComponent;
  public data: string;
  public filename = 'Export';
  public onClick(){
    this.barcode.exportImage(this.filename,'PNG');
  };
}
`

```

The filename specifies the name of the file to be downloaded.

[Export As Base64String](#)

Barcode provides the support to export its content as an image in the specified image type and returns it as base64 string.

The following code example shows how to export the barcode as a base64 string

```

`typescript
import { Component,ViewChild } from '@angular/core';
import { BarcodeGeneratorComponent } from '@syncfusion/ej2-angular-barcode-generator';
import { ClickEventArgs, } from '@syncfusion/ej2-navigations';
@Component({
  selector: "app-container",
  // specifies the template string for the barcode generator component
  template: `
<div class="col-xs-12 col-sm-12 col-lg-6 col-md-6">
<button type="button" (click)="onClick()" id="btn">Export</button>
</div>
<div>
<ejs-barcodegenerator style="display: block;" #barcode id="barcode" width="200px" height="150px"
mode="SVG" type="Codabar" value="123456789">
</div>`

```

```

})
export class AppComponent {
  @ViewChild('barcode')
  public barcode: BarcodeGeneratorComponent;
  public data: string ;
  public filename = 'Export';
  async onClick(){
    this.data = await this.barcode.exportAsBase64Image('JPG');
  };
}
`

```

Note:

Format is to specify the type or format of the exported file. You can export the barcode to the image formats such as PNG, JPG.

Breadcrumb

Getting started with Angular Breadcrumb component

This section explains how to create a simple Breadcrumb, and demonstrate the basic usage of the Breadcrumb module in an Angular environment.

Dependencies

The list of dependencies required to use the Breadcrumb module in your application is given below:

```

`javascript
|-- @syncfusion/ej2-angular-navigations
|-- @syncfusion/ej2-angular-base
|-- @syncfusion/ej2-navigations
|-- @syncfusion/ej2-base
|-- @syncfusion/ej2-data
|-- @syncfusion/ej2-lists
|-- @syncfusion/ej2-inputs
|-- @syncfusion/ej2-splitbuttons
|-- @syncfusion/ej2-popups
|-- @syncfusion/ej2-buttons
`

```

Setup Angular environment

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

`

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

`

Create an Angular application

Start a new Angular application using below Angular CLI command.

`

```
ng new my-app
```

```
cd my-app
```

`

Installing Syncfusion Breadcrumb Package

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

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

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

Ivy library distribution package

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

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

```
`bash
```

```
npm install @syncfusion/ej2-angular-navigations --save
```

`

Angular compatibility compiled package(ngcc)

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

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

```
`bash
```

```
npm install @syncfusion/ej2-angular-navigations@ngcc --save
```

`

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

```
`bash
```

```
@syncfusion/ej2-angular-navigations:"20.2.38-ngcc"
```

```
,
```

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

Adding Breadcrumb module

Import Breadcrumb module into Angular application(app.module.ts) from the package

```
@syncfusion/ej2-angular-navigations.
```

```
`javascript
```

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

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

```
// Imported Syncfusion Breadcrumb module from navigations package.
```

```
import { BreadcrumbModule } from '@syncfusion/ej2-angular-navigations';
```

```
import { AppComponent } from './app.component';
```

```
@NgModule({
```

```
imports: [BrowserModule, BreadcrumbModule], // Registering EJ2 Breadcrumb Module.
```

```
declarations: [AppComponent],
```

```
bootstrap: [AppComponent]
```

```
})
```

```
export class AppModule { }
```

```
,
```

Adding Syncfusion Breadcrumb component

Modify the template in `app.component.ts` file with `ejs-breadcrumb` to render the Breadcrumb component.

```
`javascript
```

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

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

```
enableRipple(true);
```

```
@Component({
```

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

```
template: `<!-- To Render Breadcrumb. -->
```

```
<ejs-breadcrumb [enableNavigation]="false"></ejs-breadcrumb>`
```

```
})
```

```
export class AppComponent { }
```

```
,
```

Adding CSS reference

Add Breadcrumb component's styles as given below in `style.css`.

```
`css
```

```
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
```

```
@import '../node_modules/@syncfusion/ej2-navigations/styles/material.css';
```

```
,
```

Running the application

Run the application in the browser using the following command:

```
,
```

```
ng serve
```

```
,
```

The following example shows a basic `Breadcrumb` component.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { BreadcrumbModule } from '@syncfusion/ej2-angular-navigations'
import { Component } from '@angular/core';
import { enableRipple } from '@syncfusion/ej2-base';
enableRipple(true);
@Component({
  imports: [ BreadcrumbModule],
  standalone: true,
  selector: 'app-root',
  template: `<div class="e-section-control">
    <!-- To Render Breadcrumb. -->
    <ejs-breadcrumb [enableNavigation]="false"></ejs-
breadcrumb></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));
```

Add Items to the Breadcrumb Component

Use `items` property to bind items for Breadcrumb component. The below example demonstrates the basic rendering of Breadcrumb with items support.

APP.COMPONENT.TS

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



```
import { enableRipple } from '@syncfusion/ej2-base';
enableRipple(true);
@Component({
  imports: [ BreadcrumbModule],
  standalone: true,
  selector: 'app-root',
  template: `<div class="e-section-control">
    <!-- To Render Breadcrumb with items. -->
    <ejs-breadcrumb [enableNavigation]="false">
      <e-breadcrumb-items>
        <e-breadcrumb-item iconCss="e-icons e-home"
url="https://ej2.syncfusion.com/home/angular.html"></e-breadcrumb-item>
        <e-breadcrumb-item text="Components"
url="https://ej2.syncfusion.com/angular/demos/#/material/grid/over-view"></e-
breadcrumb-item>
        <e-breadcrumb-item text="Navigations"
url="https://ej2.syncfusion.com/angular/demos/#/material/breadcrumb/default">
</e-breadcrumb-item>
        <e-breadcrumb-item text="Breadcrumb"
url="./breadcrumb/default"></e-breadcrumb-item>
      </e-breadcrumb-items>
    </ejs-breadcrumb>
  </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));
```

Enable or Disable Navigation

This feature enables or disables the item navigation. By default, the navigation will be enabled when setting `Url` property. To prevent breadcrumb item navigation, set `enableNavigation` property as `false` in `Breadcrumb`. The below example shows enabling and disabling the navigation of `Breadcrumb` items.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { BreadcrumbModule } from '@syncfusion/ej2-angular-navigations'
import { Component } from '@angular/core';
import { enableRipple } from '@syncfusion/ej2-base';
enableRipple(true);
@Component({
  imports: [ BreadcrumbModule],
  standalone: true,
  selector: 'app-root',
  template: `<div class="e-section-control">
    <!-- To Render Breadcrumb with items. -->
    <div id="breadcrumb-control">
      <div class="header"><b>EnableNavigation - false</b></div><br />
      <ejs-breadcrumb [enableNavigation]='false'>
```

```

        <e-breadcrumb-items>
            <e-breadcrumb-item iconCss= 'e-icons e-home' url=
'https://ej2.syncfusion.com/angular/demos/'>
            </e-breadcrumb-item>
            <e-breadcrumb-item text= 'Components' url=
'https://ej2.syncfusion.com/angular/demos/datagrid/overview'>
            </e-breadcrumb-item>
            <e-breadcrumb-item text= 'Navigations' url=
'https://ej2.syncfusion.com/angular/demos/menu/default'>
            </e-breadcrumb-item>
            <e-breadcrumb-item text= 'Breadcrumb' url=
'https://ej2.syncfusion.com/angular/demos/breadcrumb/default'>
            </e-breadcrumb-item>
        </e-breadcrumb-items>
    </ejs-breadcrumb>
    <br />
    <br />
    <div class="header"><b>EnableNavigation - true</b></div><br />
    <ejs-breadcrumb [enableNavigation]='true'>
        <e-breadcrumb-items>
            <e-breadcrumb-item iconCss= 'e-icons e-home' url=
'https://ej2.syncfusion.com/angular/demos/'>
            </e-breadcrumb-item>
            <e-breadcrumb-item text= 'Components' url=
'https://ej2.syncfusion.com/angular/demos/datagrid/overview'>
            </e-breadcrumb-item>
            <e-breadcrumb-item text= 'Navigations' url=
'https://ej2.syncfusion.com/angular/demos/menu/default'>
            </e-breadcrumb-item>
            <e-breadcrumb-item text= 'Breadcrumb' url=
'https://ej2.syncfusion.com/angular/demos/breadcrumb/default'>
            </e-breadcrumb-item>
        </e-breadcrumb-items>
    </ejs-breadcrumb>
</div>
</div>`
    })
    export class AppComponent {}

```

MAIN.TS

```

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

```

Data binding in Angular Breadcrumb component

Items as tag directive

The Angular Breadcrumb contains `e-breadcrumb-items` and `e-breadcrumb-item` tags to render items for the component. To bind items, use `e-breadcrumb-items` tag and `e-breadcrumb-item` tag to bind properties for breadcrumb items.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { BreadcrumbModule } from '@syncfusion/ej2-angular-navigations'
import { Component } from '@angular/core';
import { enableRipple } from '@syncfusion/ej2-base';
enableRipple(true);
@Component({
  imports: [ BreadcrumbModule],
  standalone: true,
  selector: 'app-root',
  template: `<div class="e-section-control">
    <!-- To Render Breadcrumb with items. -->
    <ejs-breadcrumb [enableNavigation]="false">
      <e-breadcrumb-items>
        <e-breadcrumb-item text="Home"
url="https://ej2.syncfusion.com/home/angular.html"></e-breadcrumb-item>
        <e-breadcrumb-item text="Components"
url="https://ej2.syncfusion.com/angular/demos/#/material/grid/over-view"></e-
breadcrumb-item>
        <e-breadcrumb-item text="Navigations"
url="https://ej2.syncfusion.com/angular/demos/#/material/breadcrumb/default">
</e-breadcrumb-item>
        <e-breadcrumb-item text="Breadcrumb"
url="./breadcrumb/default"></e-breadcrumb-item>
      </e-breadcrumb-items>
    </ejs-breadcrumb></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));

```

Items based on current Url

The breadcrumb items can be generated from the current URL of the page, if the `url` property is not provided or when the user does not specify the breadcrumb items using the `BreadcrumbItemDirective` tag. The following example shows the breadcrumb items that are generated based on the current URL.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { BreadcrumbModule } from '@syncfusion/ej2-angular-navigations'
import { Component } from '@angular/core';
import { enableRipple } from '@syncfusion/ej2-base';
enableRipple(true);
@Component({
  imports: [ BreadcrumbModule],
  standalone: true,
  selector: 'app-root',
  template: `<div class="e-section-control">
    <!-- To Render Breadcrumb. -->

```

```

        <ejs-breadcrumb [enableNavigation]="false"></ejs-
breadcrumb></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));

```

This sample is hosted in different location, so the Breadcrumb Component is rendered with different location instead of the actual location.

Static URL

The breadcrumb items can be generated by providing the `url` property in the component, if the user does not specify the breadcrumb items using the `BreadcrumbItemDirective` tag. The following example shows the breadcrumb items generated from the provided url in the component.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { BreadcrumbModule } from '@syncfusion/ej2-angular-navigations'
import { Component } from '@angular/core';
import { enableRipple } from '@syncfusion/ej2-base';
enableRipple(true);
@Component({
  imports: [ BreadcrumbModule],
  standalone: true,
  selector: 'app-root',
  template: `<div class="e-section-control">
    <!-- To Render Breadcrumb. -->
    <ejs-breadcrumb [enableNavigation]="false"
url="https://ej2.syncfusion.com/angular/demos/breadcrumb/bind-to-
location"></ejs-breadcrumb></div>`
})
export class AppComponent {}

```

MAIN.TS

```

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

```

Customize text when generated items using Url

The breadcrumb items text can be customized by using the `beforeItemRender` event. In the following example, `bind-to-location` text was customized as `location`.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'

```

```
import { BrowserModule } from '@angular/platform-browser'
import { BreadcrumbModule } from '@syncfusion/ej2-angular-navigations'
import { Component } from '@angular/core';
import { enableRipple } from '@syncfusion/ej2-base';
import { BreadcrumbBeforeItemRenderEventArgs } from '@syncfusion/ej2-navigations';
enableRipple(true);
@Component({
  imports: [ BreadcrumbModule],
  standalone: true,
  selector: 'app-root',
  template: `<div class="e-section-control">
    <!-- To Render Breadcrumb with items. -->
    <ejs-breadcrumb [enableNavigation]="false"
(beforeItemRender)="beforeItemRenderHandler($event)"
url="https://ej2.syncfusion.com/angular/demos/breadcrumb/bind-to-location"></ejs-breadcrumb></div>`
})
export class AppComponent {
  beforeItemRenderHandler(args: BreadcrumbBeforeItemRenderEventArgs): void
  {
    if (args.item.text === 'bind-to-location') {
      args.item.text = 'location';
    }
  }
}
```

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 Breadcrumb component

The Breadcrumb component contains an icon/image to provide a visual representation of an item.

Loading icon in BreadcrumbItem

To load the icon on the breadcrumb item, set the `iconCss` property.

Breadcrumb with Font Icon

To place the font icon on the breadcrumb item, set the `iconCss` property to `e-icons` with the required icon CSS. By default, the icon is positioned to the left side of the item.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { BreadcrumbModule } from '@syncfusion/ej2-angular-navigations'
import { Component } from '@angular/core';
import { enableRipple } from '@syncfusion/ej2-base';
enableRipple(true);
@Component({
  imports: [ BreadcrumbModule],
  standalone: true,
```

```

    selector: 'app-root',
    template: `<div class="e-section-control">
      <!-- To Render Breadcrumb with items. -->
      <ejs-breadcrumb [enableNavigation]="false">
        <e-breadcrumb-items>
          <e-breadcrumb-item iconCss="e-icons e-home"
url="https://ej2.syncfusion.com/home/angular.html"></e-breadcrumb-item>
          <e-breadcrumb-item text="Components"
url="https://ej2.syncfusion.com/angular/demos/#/material/grid/over-view"></e-
breadcrumb-item>
          <e-breadcrumb-item text="Navigations"
url="https://ej2.syncfusion.com/angular/demos/#/material/breadcrumb/default">
</e-breadcrumb-item>
          <e-breadcrumb-item text="Breadcrumb"
url="./breadcrumb/default"></e-breadcrumb-item>
        </e-breadcrumb-items>
      </ejs-breadcrumb>
    </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));

```

Breadcrumb with Image

In the Breadcrumb component, images can be added for the items using the `iconCss` property. In the following example, an image was added to the breadcrumb item by using the iconCss class `e-image-home` and specifying height and width.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { BreadcrumbModule } from '@syncfusion/ej2-angular-navigations'
import { Component } from '@angular/core';
import { enableRipple } from '@syncfusion/ej2-base';
enableRipple(true);
@Component({
  imports: [ BreadcrumbModule],
  standalone: true,
  selector: 'app-root',
  template: `<div class="e-section-control">
    <!-- To Render Breadcrumb with items. -->
    <ejs-breadcrumb [enableNavigation]="false">
      <e-breadcrumb-items>
        <e-breadcrumb-item iconCss="e-image-home"
url="https://ej2.syncfusion.com/home/angular.html"></e-breadcrumb-item>
        <e-breadcrumb-item text="Components"
url="https://ej2.syncfusion.com/angular/demos/#/material/grid/over-view"></e-
breadcrumb-item>
      </e-breadcrumb-items>
    </ejs-breadcrumb>
  </div>`
})

```

```

        <e-breadcrumb-item text="Navigations"
url="https://ej2.syncfusion.com/angular/demos/#/material/breadcrumb/default">
</e-breadcrumb-item>
        <e-breadcrumb-item text="Breadcrumb"
url="./breadcrumb/default"></e-breadcrumb-item>
    </e-breadcrumb-items>
</ejs-breadcrumb>
</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));

```

Breadcrumb with SVG Image

In the Breadcrumb component, SVG image can be added for the items using the `iconCss` property. In the following example, SVG image was added to the breadcrumb item by using the `iconCss` class `e-svg-home` and specifying height and width.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { BreadcrumbModule } from '@syncfusion/ej2-angular-navigations'
import { Component } from '@angular/core';
import { enableRipple } from '@syncfusion/ej2-base';
enableRipple(true);
@Component({
  imports: [ BreadcrumbModule],
  standalone: true,
  selector: 'app-root',
  template: `<div class="e-section-control">
    <!-- To Render Breadcrumb with items. -->
    <ejs-breadcrumb [enableNavigation]="false">
      <e-breadcrumb-items>
        <e-breadcrumb-item iconCss="e-svg-home"
url="https://ej2.syncfusion.com/home/angular.html"></e-breadcrumb-item>
        <e-breadcrumb-item text="Components"
url="https://ej2.syncfusion.com/angular/demos/#/material/grid/over-view"></e-
breadcrumb-item>
        <e-breadcrumb-item text="Navigations"
url="https://ej2.syncfusion.com/angular/demos/#/material/breadcrumb/default">
</e-breadcrumb-item>
        <e-breadcrumb-item text="Breadcrumb"
url="./breadcrumb/default"></e-breadcrumb-item>
      </e-breadcrumb-items>
    </ejs-breadcrumb>
  </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));
```

Icon Position

By default, the icon is positioned to the left side of the item in the Breadcrumb component. If you need to add the icon right to the breadcrumb item, add the `e-icon-right` class to the required item. In the following example, the `e-icon-right` class was added to the breadcrumb items using the `beforeItemRender` event.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { BreadcrumbModule } from '@syncfusion/ej2-angular-navigations'
import { Component } from '@angular/core';
import { BreadcrumbBeforeItemRenderEventArgs } from '@syncfusion/ej2-angular-navigations';
import { enableRipple } from '@syncfusion/ej2-base';
enableRipple(true);
@Component({
  imports: [ BreadcrumbModule],
  standalone: true,
  selector: 'app-root',
  template: `<div class="e-section-control">
    <!-- To Render Breadcrumb. -->
    <div id="breadcrumb-control" class="control-section">
      <div class="header"><b>Icon Position - Left</b></div><br />
      <ejs-breadcrumb [enableNavigation]="false">
        <e-breadcrumb-items>
          <e-breadcrumb-item iconCss="e-bicons e-folder"
text="Program Files"></e-breadcrumb-item>
          <e-breadcrumb-item iconCss="e-bicons e-folder"
text="Services"></e-breadcrumb-item>
          <e-breadcrumb-item iconCss="e-bicons e-file"
text="Config.json"></e-breadcrumb-item>
        </e-breadcrumb-items>
      </ejs-breadcrumb>
    <br /><br />
    <div class="header"><b>Icon Position - Right</b></div><br />
    <ejs-breadcrumb [enableNavigation]="false"
(beforeItemRender)="beforeItemRenderHandler($event)">
      <e-breadcrumb-items>
        <e-breadcrumb-item iconCss="e-bicons e-folder"
text="Program Files"></e-breadcrumb-item>
        <e-breadcrumb-item iconCss="e-bicons e-folder"
text="Services"></e-breadcrumb-item>
        <e-breadcrumb-item iconCss="e-bicons e-file"
text="Config.json"></e-breadcrumb-item>
      </e-breadcrumb-items>
    </ejs-breadcrumb>
  </div>
```



```

        </div>`
    })
    export class AppComponent {
        public beforeItemRenderHandler(args:
        BreadcrumbBeforeItemRenderEventArgs): void {
            if(args.item.text !== '') {
                args.element.classList.add('e-icon-right');
            }
        }
    }
}

```

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

To display only icons for the items, add icons using the `iconCss` property. In the following example, breadcrumb items were demonstrated with only icons by providing the `iconCss` property.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { BreadcrumbModule } from '@syncfusion/ej2-angular-navigations'
import { Component } from '@angular/core';
import { enableRipple } from '@syncfusion/ej2-base';
enableRipple(true);
@Component({
    imports: [ BreadcrumbModule],
    standalone: true,
    selector: 'app-root',
    template: `<div class="e-section-control">
        <!-- To Render Breadcrumb. -->
        <ejs-breadcrumb [enableNavigation]="false">
            <e-breadcrumb-items>
                <e-breadcrumb-item iconCss="e-icons e-home"></e-
breadcrumb-item>
                <e-breadcrumb-item iconCss="e-bicons e-folder"></e-
breadcrumb-item>
                <e-breadcrumb-item iconCss="e-bicons e-file"></e-
breadcrumb-item>
            </e-breadcrumb-items>
        </ejs-breadcrumb>
        </div>`
})
export class AppComponent {}

```

MAIN.TS

```

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

```

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

Show icon only for first item

To show icon only for the first item in the Breadcrumb component, add icons to the first item using the `iconCss` property. In the following example, the icon was provided only for the first item by setting the `iconCss` property.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { BreadcrumbModule } from '@syncfusion/ej2-angular-navigations'
import { Component } from '@angular/core';
import { enableRipple } from '@syncfusion/ej2-base';
enableRipple(true);
@Component({
  imports: [ BreadcrumbModule],
  standalone: true,
  selector: 'app-root',
  template: `<div class="e-section-control">
    <!-- To Render Breadcrumb with items. -->
    <ejs-breadcrumb [enableNavigation]="false">
      <e-breadcrumb-items>
        <e-breadcrumb-item iconCss="e-icons e-home"
url="https://ej2.syncfusion.com/home/angular.html"></e-breadcrumb-item>
        <e-breadcrumb-item text="Components"
url="https://ej2.syncfusion.com/angular/demos/#/material/grid/over-view"></e-
breadcrumb-item>
        <e-breadcrumb-item text="Navigations"
url="https://ej2.syncfusion.com/angular/demos/#/material/breadcrumb/default">
</e-breadcrumb-item>
        <e-breadcrumb-item text="Breadcrumb"
url="./breadcrumb/default"></e-breadcrumb-item>
      </e-breadcrumb-items>
    </ejs-breadcrumb>
  </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));
```

Navigations in Angular Breadcrumb component

The breadcrumb item navigates to the path while clicking the item. To enable navigation, `url` property was bound to the items.

URL

In the Breadcrumb component, the item represents the url. The breadcrumb items can be provided with either relative or absolute URL.

Relative URL

The breadcrumb items with relative URL contain only the path but do not locate the path or server. The following example represents the breadcrumb items with relative url.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { BreadcrumbModule } from '@syncfusion/ej2-angular-navigations'
import { Component } from '@angular/core';
import { enableRipple } from '@syncfusion/ej2-base';
enableRipple(true);
@Component({
  imports: [ BreadcrumbModule],
  standalone: true,
  selector: 'app-root',
  template: `<div class="e-section-control">
    <!-- To Render Breadcrumb. -->
    <ejs-breadcrumb [enableNavigation]="false">
      <e-breadcrumb-items>
        <e-breadcrumb-item text="Home" url=".."></e-breadcrumb-
item>
        <e-breadcrumb-item text="Getting"
url="./breadcrumb/getting-started"></e-breadcrumb-item>
        <e-breadcrumb-item text="Icons"
url="./breadcrumb/icons"></e-breadcrumb-item>
        <e-breadcrumb-item text="Navigations"
url="./breadcrumb/navigations"></e-breadcrumb-item>
        <e-breadcrumb-item text="Overflow"
url="./breadcrumb/overflow"></e-breadcrumb-item>
      </e-breadcrumb-items>
    </ejs-breadcrumb></div>`
  })
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));
```

Absolute URL

The breadcrumb items with Absolute URL contain the path and locate to the resource if the absolute url is bound to the breadcrumb item. The following example represents the breadcrumb items with absolute url.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { BreadcrumbModule } from '@syncfusion/ej2-angular-navigations'
import { Component } from '@angular/core';
import { enableRipple } from '@syncfusion/ej2-base';
enableRipple(true);
@Component({
```

```

imports: [ BreadcrumbModule],
standalone: true,
  selector: 'app-root',
  template: `<div class="e-section-control">
    <!-- To Render Breadcrumb. -->
    <ejs-breadcrumb [enableNavigation]="false">
      <e-breadcrumb-items>
        <e-breadcrumb-item text="Home"
url="https://ej2.syncfusion.com/documentation/introduction/"></e-breadcrumb-
item>
        <e-breadcrumb-item text="Getting"
url="https://ej2.syncfusion.com/documentation/breadcrumb/getting-
started"></e-breadcrumb-item>
        <e-breadcrumb-item text="Icons"
url="https://ej2.syncfusion.com/documentation/breadcrumb/icons"></e-
breadcrumb-item>
        <e-breadcrumb-item text="Navigations"
url="https://ej2.syncfusion.com/documentation/breadcrumb/navigations"></e-
breadcrumb-item>
        <e-breadcrumb-item text="Overflow"
url="https://ej2.syncfusion.com/documentation/breadcrumb/overflow"></e-
breadcrumb-item>
      </e-breadcrumb-items>
    </ejs-breadcrumb>
  </div>`
  ))
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));

```

Enable navigation for last Breadcrumb item

The feature enables the last item of the Breadcrumb component by setting the `enableActiveItemNavigation` property to true. In the following example, the last item of the Breadcrumb was enabled.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { BreadcrumbModule } from '@syncfusion/ej2-angular-navigations'
import { Component } from '@angular/core';
import { enableRipple } from '@syncfusion/ej2-base';
enableRipple(true);
@Component({
  imports: [ BreadcrumbModule],
  standalone: true,
    selector: 'app-root',
    template: `<div class="e-section-control">
      <!-- To Render Breadcrumb. -->

```

```

        <ejs-breadcrumb [enableNavigation]="false"
[enableActiveItemNavigation]="true">
            <e-breadcrumb-items>
                <e-breadcrumb-item text="Home"
url="https://ej2.syncfusion.com/documentation/introduction/"></e-breadcrumb-
item>
                <e-breadcrumb-item text="Getting"
url="https://ej2.syncfusion.com/documentation/breadcrumb/getting-
started"></e-breadcrumb-item>
                <e-breadcrumb-item text="Icons"
url="https://ej2.syncfusion.com/documentation/breadcrumb/icons"></e-
breadcrumb-item>
                <e-breadcrumb-item text="Navigations"
url="https://ej2.syncfusion.com/documentation/breadcrumb/navigations"></e-
breadcrumb-item>
                <e-breadcrumb-item text="Overflow"
url="https://ej2.syncfusion.com/documentation/breadcrumb/overflow"></e-
breadcrumb-item>
            </e-breadcrumb-items>
        </ejs-breadcrumb>
    </div>`
    })
    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));

```

Open URL in new page or tab

To open the breadcrumb item in a new page or tab, set the target property of the required item url to blank in the Breadcrumb component. In the following example, the target property of All Components item url was set to blank by using the `beforeItemRender` event which locates to the path in the new tab.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { BreadcrumbModule } from '@syncfusion/ej2-angular-navigations'
import { Component } from '@angular/core';
import { enableRipple } from '@syncfusion/ej2-base';
import { BreadcrumbBeforeItemRenderEventArgs } from '@syncfusion/ej2-
navigations';
enableRipple(true);
@Component({
  imports: [ BreadcrumbModule],
  standalone: true,
  selector: 'app-root',
  template: `<div class="e-section-control">
    <!-- To Render Breadcrumb. -->
    <ejs-breadcrumb
(beforeItemRender)="beforeItemRenderHandler($event)">

```

```

        <e-breadcrumb-items>
            <e-breadcrumb-item text="Home"
url="https://ej2.syncfusion.com/documentation/introduction/"></e-breadcrumb-
item>
            <e-breadcrumb-item text="Getting"
url="https://ej2.syncfusion.com/documentation/breadcrumb/getting-
started"></e-breadcrumb-item>
            <e-breadcrumb-item text="Icons"
url="https://ej2.syncfusion.com/documentation/breadcrumb/icons"></e-
breadcrumb-item>
            <e-breadcrumb-item text="Navigations"
url="https://ej2.syncfusion.com/documentation/breadcrumb/navigations"></e-
breadcrumb-item>
            <e-breadcrumb-item text="Overflow"
url="https://ej2.syncfusion.com/documentation/breadcrumb/overflow"></e-
breadcrumb-item>
        </e-breadcrumb-items>
    </ejs-breadcrumb>
</div>`
})
export class AppComponent {
    public beforeItemRenderHandler(args:
BreadcrumbBeforeItemRenderEventArgs): void {
        if (args.element.children[0]) {
            (args.element.children[0] as any).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));

```

Templates in Angular Breadcrumb component

The Breadcrumb component provides a way to customize the items using `itemTemplate` and the separators using `separatorTemplate` properties.

Item Template

In the following example, Shopping Cart details are used as breadcrumb Items and the items are customized by the chips component using `itemTemplate`.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { BreadcrumbModule } from '@syncfusion/ej2-angular-navigations'
import { ChipListModule } from '@syncfusion/ej2-angular-buttons'
import { Component } from '@angular/core';
import { enableRipple } from '@syncfusion/ej2-base';
enableRipple(true);
@Component({
    imports: [ BreadcrumbModule, ChipListModule],

```

```

standalone: true,
selector: 'app-root',
template: `<div class="e-section-control">
  <!-- To Render Breadcrumb. -->
  <ejs-breadcrumb cssClass="e-breadcrumb-chips">
    <e-breadcrumb-items>
      <e-breadcrumb-item text="Cart"></e-breadcrumb-item>
      <e-breadcrumb-item text="Billing"></e-breadcrumb-item>
      <e-breadcrumb-item text="Shipping"></e-breadcrumb-item>
      <e-breadcrumb-item text="Payment"></e-breadcrumb-item>
    </e-breadcrumb-items>
    <ng-template #itemTemplate let-dataSource="">
      <ejs-chiplist>
        <e-chips>
          <e-chip text={{dataSource.text}} cssClass="e-
primary"></e-chip>
        </e-chips>
      </ejs-chiplist>
    </ng-template>
  </ejs-breadcrumb>
</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 Template

In the following example, the separators are customized with icons using `separatorTemplate`.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { BreadcrumbModule } from '@syncfusion/ej2-angular-navigations';
import { ChipListModule } from '@syncfusion/ej2-angular-buttons';
import { Component } from '@angular/core';
import { enableRipple } from '@syncfusion/ej2-base';
enableRipple(true);
@Component({
  imports: [BreadcrumbModule, ChipListModule],
  standalone: true,
  selector: 'app-root',
  template: `<div class="e-section-control">
    <!-- To Render Breadcrumb. -->
    <ejs-breadcrumb cssClass="e-breadcrumb-chips">
      <e-breadcrumb-items>
        <e-breadcrumb-item text="Cart"></e-breadcrumb-item>
        <e-breadcrumb-item text="Billing"></e-breadcrumb-
item>

```

```

        <e-breadcrumb-item text="Shipping"></e-breadcrumb-
item>
        <e-breadcrumb-item text="Payment"></e-breadcrumb-
item>
    </e-breadcrumb-items>
    <ng-template #separatorTemplate>
        <span class="e-icons e-arrow"></span>
    </ng-template>
</ejs-breadcrumb>
</div>`
}))
export class AppComponent {}

```

MAIN.TS

```

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

```

Customize Specific Item Template

The specific breadcrumb item can be customizable using itemTemplate with conditional rendering. In the following example, added the span element only for the breadcrumb text in breadcrumb item.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { BreadcrumbModule } from '@syncfusion/ej2-angular-navigations'
import { ChipListModule } from '@syncfusion/ej2-angular-buttons'
import { Component } from '@angular/core';
import { enableRipple } from '@syncfusion/ej2-base';
enableRipple(true);
@Component({
  imports: [ BreadcrumbModule, ChipListModule],
  standalone: true,
  selector: 'app-root',
  template: `<div class="e-section-control">
    <!-- To Render Breadcrumb. -->
    <ejs-breadcrumb [enableNavigation]="false" cssClass="e-specific-item-
template">
      <e-breadcrumb-items>
        <e-breadcrumb-item text="Home"
url="https://ej2.syncfusion.com/home/angular.html"></e-breadcrumb-item>
        <e-breadcrumb-item text="Components"
url="https://ej2.syncfusion.com/angular/demos/#/material/grid/over-view"></e-
breadcrumb-item>
        <e-breadcrumb-item text="Navigations"
url="https://ej2.syncfusion.com/angular/demos/#/material/menu/default"></e-
breadcrumb-item>
        <e-breadcrumb-item text="Breadcrumb"
url="./breadcrumb/default"></e-breadcrumb-item>
      </e-breadcrumb-items>
      <ng-template #itemTemplate let-dataSource="">
        <div>

```



```

        <span *ngIf="dataSource.text === 'Breadcrumb'; else
otherContent" class="e-searchfor-text"><span style="margin-right: 5px">Search
for:</span>
        <a class="e-breadcrumb-text" href="dataSource.url"
onclick="return false">{{dataSource.text}}</a></span>
        <ng-template #otherContent>
        <a class="e-breadcrumb-text" href="dataSource.url"
onclick="return false">{{dataSource.text}}</a>
        </ng-template>
    </div>
</ng-template>
</ejs-breadcrumb>
</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));

```

Overflow in Angular Breadcrumb component

Overflow Mode

In the Breadcrumb component, `maxItems` and `overflowMode` properties were used to limit the number of breadcrumb items to be displayed.

In the following example, the `maxItems` is set as 3 with `overflowMode` as Default. To prevent breadcrumb item navigation, the `enableNavigation` property has been set to false in the Breadcrumb component.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { BreadcrumbModule } from '@syncfusion/ej2-angular-navigations'
import { Component } from '@angular/core';
import { enableRipple } from '@syncfusion/ej2-base';
enableRipple(true);
@Component({
  imports: [ BreadcrumbModule],
  standalone: true,
  selector: 'app-root',
  template: `<div class="e-section-control">
    <!-- To Render Breadcrumb. -->
    <ejs-breadcrumb [enableNavigation]="false" [maxItems]=3 >
      <e-breadcrumb-items>
        <e-breadcrumb-item text="Home" url=".."></e-breadcrumb-
item>
        <e-breadcrumb-item text="Getting"
url="./breadcrumb/getting-started"></e-breadcrumb-item>
        <e-breadcrumb-item text="Data-Binding"
url="./breadcrumb/data-binding"></e-breadcrumb-item>

```

```

        <e-breadcrumb-item text="Icons"
url="./breadcrumb/icons"></e-breadcrumb-item>
        <e-breadcrumb-item text="Navigations"
url="./breadcrumb/navigations"></e-breadcrumb-item>
        <e-breadcrumb-item text="Templates"
url="./breadcrumb/templates"></e-breadcrumb-item>
        <e-breadcrumb-item text="Overflow"
url="./breadcrumb/overflow"></e-breadcrumb-item>
    </e-breadcrumb-items>
    <ng-template #separatorTemplate>
        <span class='e-bicons e-arrow'></span>
    </ng-template>
</ejs-breadcrumb>
</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));

```

The following overflow modes are available in the Breadcrumb component.

- Collapsed
- Menu
- Wrap
- Scroll
- Hidden
- None

Collapsed

Collapsed mode shows the first and last Breadcrumb items and hides the remaining items with a collapsed icon. When the collapsed icon is clicked, all items become visible and navigable.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { BreadcrumbModule } from '@syncfusion/ej2-angular-navigations'
import { Component } from '@angular/core';
import { enableRipple } from '@syncfusion/ej2-base';
enableRipple(true);
@Component({
  imports: [ BreadcrumbModule],
  standalone: true,
  selector: 'app-root',
  template: `<div class="e-section-control">
    <!-- To Render Breadcrumb. -->
    <ejs-breadcrumb [enableNavigation]="false" [maxItems]=3
overflowMode="Collapsed" >
      <e-breadcrumb-items>

```

```

        <e-breadcrumb-item text="Home" url=".."></e-breadcrumb-
item>
        <e-breadcrumb-item text="Getting"
url="./breadcrumb/getting-started"></e-breadcrumb-item>
        <e-breadcrumb-item text="Data-Binding"
url="./breadcrumb/data-binding"></e-breadcrumb-item>
        <e-breadcrumb-item text="Icons"
url="./breadcrumb/icons"></e-breadcrumb-item>
        <e-breadcrumb-item text="Navigations"
url="./breadcrumb/navigations"></e-breadcrumb-item>
        <e-breadcrumb-item text="Templates"
url="./breadcrumb/templates"></e-breadcrumb-item>
        <e-breadcrumb-item text="Overflow"
url="./breadcrumb/overflow"></e-breadcrumb-item>
    </e-breadcrumb-items>
    <ng-template #separatorTemplate>
        <span class='e-bicons e-arrow'></span>
    </ng-template>
</ejs-breadcrumb>
</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));

```

Menu

Menu mode shows the number of Breadcrumb items that can be accommodated within the container space and creates a submenu with the remaining items.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { BreadcrumbModule } from '@syncfusion/ej2-angular-navigations';
import { Component } from '@angular/core';
import { enableRipple } from '@syncfusion/ej2-base';
enableRipple(true);
@Component({
  imports: [ BreadcrumbModule ],
  standalone: true,
  selector: 'app-root',
  template: `<div class="e-section-control">
    <!-- To Render Breadcrumb. -->
    <ejs-breadcrumb [enableNavigation]="false" [maxItems]=3
overflowMode="Menu" >
      <e-breadcrumb-items>
        <e-breadcrumb-item text="Home" url=".."></e-breadcrumb-
item>
        <e-breadcrumb-item text="Getting"
url="./breadcrumb/getting-started"></e-breadcrumb-item>

```

```

        <e-breadcrumb-item text="Data-Binding"
url="./breadcrumb/data-binding"></e-breadcrumb-item>
        <e-breadcrumb-item text="Icons"
url="./breadcrumb/icons"></e-breadcrumb-item>
        <e-breadcrumb-item text="Navigations"
url="./breadcrumb/navigations"></e-breadcrumb-item>
        <e-breadcrumb-item text="Templates"
url="./breadcrumb/templates"></e-breadcrumb-item>
        <e-breadcrumb-item text="Overflow"
url="./breadcrumb/overflow"></e-breadcrumb-item>
    </e-breadcrumb-items>
    <ng-template #separatorTemplate>
        <span class='e-bicons e-arrow'></span>
    </ng-template>
</ejs-breadcrumb>
</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));

```

Wrap

Wrap mode wraps the items to multiple lines when the Breadcrumb's width exceeds the container space.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { BreadcrumbModule } from '@syncfusion/ej2-angular-navigations'
import { Component } from '@angular/core';
import { enableRipple } from '@syncfusion/ej2-base';
enableRipple(true);
@Component({
  imports: [ BreadcrumbModule],
  standalone: true,
  selector: 'app-root',
  template: `<div class="e-section-control">
    <!-- To Render Breadcrumb. -->
    <ejs-breadcrumb [enableNavigation]="false" [maxItems]=3
overflowMode="Wrap" >
      <e-breadcrumb-items>
        <e-breadcrumb-item text="Home" url=".."></e-breadcrumb-
item>
        <e-breadcrumb-item text="Getting"
url="./breadcrumb/getting-started"></e-breadcrumb-item>
        <e-breadcrumb-item text="Data-Binding"
url="./breadcrumb/data-binding"></e-breadcrumb-item>
        <e-breadcrumb-item text="Icons"
url="./breadcrumb/icons"></e-breadcrumb-item>

```

```

        <e-breadcrumb-item text="Navigations"
url="./breadcrumb/navigations"></e-breadcrumb-item>
        <e-breadcrumb-item text="Templates"
url="./breadcrumb/templates"></e-breadcrumb-item>
        <e-breadcrumb-item text="Overflow"
url="./breadcrumb/overflow"></e-breadcrumb-item>
    </e-breadcrumb-items>
    <ng-template #separatorTemplate>
        <span class='e-bicons e-arrow'></span>
    </ng-template>
</ejs-breadcrumb>
</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));

```

Scroll

Scroll mode shows an HTML scroll bar when the Breadcrumb's width exceeds the container space.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { BreadcrumbModule } from '@syncfusion/ej2-angular-navigations'
import { Component } from '@angular/core';
import { enableRipple } from '@syncfusion/ej2-base';
enableRipple(true);
@Component({
  imports: [ BreadcrumbModule],
  standalone: true,
  selector: 'app-root',
  template: `<div class="e-section-control">
    <!-- To Render Breadcrumb. -->
    <ejs-breadcrumb [enableNavigation]="false" [maxItems]=3
overflowMode="Scroll" >
      <e-breadcrumb-items>
        <e-breadcrumb-item text="Home" url=".."></e-breadcrumb-
item>
        <e-breadcrumb-item text="Getting"
url="./breadcrumb/getting-started"></e-breadcrumb-item>
        <e-breadcrumb-item text="Data-Binding"
url="./breadcrumb/data-binding"></e-breadcrumb-item>
        <e-breadcrumb-item text="Icons"
url="./breadcrumb/icons"></e-breadcrumb-item>
        <e-breadcrumb-item text="Navigations"
url="./breadcrumb/navigations"></e-breadcrumb-item>
        <e-breadcrumb-item text="Templates"
url="./breadcrumb/templates"></e-breadcrumb-item>
        <e-breadcrumb-item text="Overflow"
url="./breadcrumb/overflow"></e-breadcrumb-item>

```

```

        </e-breadcrumb-items>
        <ng-template #separatorTemplate>
            <span class='e-bicons e-arrow'></span>
        </ng-template>
    </ejs-breadcrumb>
</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));

```

Hidden

Hidden mode shows the maximum number of items possible in the container space and hides the remaining items. Clicking on a previous item will make the hidden item visible.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { BreadcrumbModule } from '@syncfusion/ej2-angular-navigations'
import { Component } from '@angular/core';
import { enableRipple } from '@syncfusion/ej2-base';
enableRipple(true);
@Component({
  imports: [ BreadcrumbModule],
  standalone: true,
  selector: 'app-root',
  template: `<div class="e-section-control">
    <!-- To Render Breadcrumb. -->
    <ejs-breadcrumb [enableNavigation]="false" [maxItems]=3
overflowMode="Hidden" >
      <e-breadcrumb-items>
        <e-breadcrumb-item text="Home" url=".."></e-breadcrumb-
item>
        <e-breadcrumb-item text="Getting"
url="./breadcrumb/getting-started"></e-breadcrumb-item>
        <e-breadcrumb-item text="Data-Binding"
url="./breadcrumb/data-binding"></e-breadcrumb-item>
        <e-breadcrumb-item text="Icons"
url="./breadcrumb/icons"></e-breadcrumb-item>
        <e-breadcrumb-item text="Navigations"
url="./breadcrumb/navigations"></e-breadcrumb-item>
        <e-breadcrumb-item text="Templates"
url="./breadcrumb/templates"></e-breadcrumb-item>
        <e-breadcrumb-item text="Overflow"
url="./breadcrumb/overflow"></e-breadcrumb-item>
      </e-breadcrumb-items>
      <ng-template #separatorTemplate>
        <span class='e-bicons e-arrow'></span>
      </ng-template>
    </div>`
})

```

```

        </ejs-breadcrumb>
      </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));

```

None

None mode shows all the items on a single line.

Accessibility in Angular Breadcrumb component

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

The accessibility compliance for the Breadcrumb component is outlined below.

| Accessibility Criteria | Compatibility |

| -- | -- |

| [WCAG 2.2](#) Support | |

| [Section 508](#) Support | |

| Screen Reader Support | |

| Right-To-Left Support | |

| Color Contrast | |

| Mobile Device Support | |

| Keyboard Navigation Support | |

| [Accessibility Checker](#) Validation | |

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

<style>

.post .post-content img {

display: inline-block;

```
margin: 0.5em 0;
```

```
}
```

```
</style>
```

```
<div> - 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 Breadcrumb component followed the [WAI-ARIA](#) patterns to meet the accessibility. The following ARIA attributes are used in the Breadcrumb component:

Attributes	Purpose
------------	---------

---	---
-----	-----

aria-label	Indicates the breadcrumb item text.
------------	-------------------------------------

aria-disabled	Indicates the state of breadcrumb item whether it is disabled.
---------------	--

Keyboard interaction

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

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

---	---
-----	-----

Tab	Navigate to the next item and also next item in the popup of menu type overflow.
-----	--

Shift + Tab	Navigate to the previous item also previous item in the popup of menu type overflow.
-------------	--

Enter key in normal mode	Select the breadcrumb item.
--------------------------	-----------------------------

Enter key in normal mode	To open the popup of menu type overflow mode when you press enter on collapsed button and It will expand the items of collapsed type overflow mode when you press enter on collapsed button.
--------------------------	--

Ensuring accessibility

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

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

See also

- [Accessibility in Syncfusion Angular components](#)

Bullet Chart

Getting started with Angular Bullet chart component

This section explains you the steps required to create a simple Bullet Chart and demonstrate the basic usage of the Bullet Chart component in an Angular environment.

Setup Angular Environment

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

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

Create an Angular Application

Start a new Angular application using below Angular CLI command.

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

Installing Syncfusion BulletChart package

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

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

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

Ivy library distribution package

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

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

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

Angular compatibility compiled package(ngcc)

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

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

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

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

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

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

Registering Bullet Chart Module

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

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

- Modify the template in `app.component.ts` file to render the `ej2-angular-charts` component

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

```
`typescript
import { Component, ViewEncapsulation } from '@angular/core';
@Component({
  selector: 'app-container',
  // specifies the template string for the Bullet Charts component
  template: <ejs-bulletchart id='container'></ejs-bulletchart>
```

```
encapsulation: ViewEncapsulation.None
})
```

```
export class AppComponent { }
```

```
,
```

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

```
,
```

```
<app-container></app-container>
```

```
,
```

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

```
,
```

```
npm start
```

```
,
```

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { BulletChartModule } from '@syncfusion/ej2-angular-charts'
import { Component } from '@angular/core';
@Component({
  imports: [
    BulletChartModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the Chart component
  template: `<ejs-bulletchart id="chart-container"></ejs-bulletchart>`
})
export class AppComponent {
}
```

MAIN.TS

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

Module Injection

Bullet Chart component are segregated into individual feature-wise modules. In order to use a particular feature, you need to inject its feature service in the AppModule. Please find relevant feature service name and description as follows.

- **BulletTooltipService** - Inject this provider to use tooltip feature.

These modules should be injected to the provider section as follows,

```
`typescript
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import { BulletChartComponent } from '@syncfusion/ej2-angular-charts';
import { BulletTooltipService } from '@syncfusion/ej2-angular-charts';
@NgModule({
  imports: [
    BrowserModule,
  ],
  declarations: [AppComponent, BulletChartComponent],
  bootstrap: [AppComponent],
  providers: [ BulletTooltipService ]
})
`
```

Bullet Chart with Data

This section explains how to plot local data to the bullet chart.

```
`typescript
export class AppComponent implements OnInit {
  public data: Object[];
  ngOnInit(): void {
    // Data for bullet chart
    this.data = [
      { value: 100, target: 80 },
      { value: 200, target: 180 },
      { value: 300, target: 280 },
      { value: 400, target: 380 },
      { value: 500, target: 480 },
    ];
  }
}
`
```

Now assign the local data to [dataSource](#) property. **value** and **target** values should be mapped with [valueField](#) and [targetField](#) respectively.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { BulletChartModule } from '@syncfusion/ej2-angular-charts'
import { Component } from '@angular/core';
@Component({
  imports: [
    BulletChartModule
  ],
  providers: [ ],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-bulletchart valueField='value' targetField='target'
    [minimum]='minimum' [maximum]='maximum' [interval]='interval'
    [dataSource]='data'>
    </ejs-bulletchart>`
})
export class AppComponent {
  public minimum: number = 0;
  public maximum: number = 300;
  public interval: number = 50;
  public data: Object[] = [
    { value: 100, target: 80 },
    { value: 200, target: 180 },
    { value: 300, target: 280 },
    { value: 400, target: 380 },
    { value: 500, target: 480 },
  ];
}
```

MAIN.TS

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

Add Bullet Chart Title

You can add a title using [title](#) property to the bullet chart to provide quick information to the user about the data plotted in the bullet chart.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { BulletChartModule } from '@syncfusion/ej2-angular-charts'
import { Component } from '@angular/core';
@Component({
  imports: [
    BulletChartModule
  ],
  providers: [ ],
  standalone: true,
  selector: 'app-container',
```

```

    template: `<ejs-bulletchart valueField='value' targetField='target'
    title='Revenue'
                [minimum]='minimum' [maximum]='maximum' [interval]='interval'
    [dataSource]='data'>
        </ejs-bulletchart>`
  })
  export class AppComponent {
    public minimum: number = 0;
    public maximum: number = 300;
    public interval: number = 50;
    public data: Object[] = [
      { value: 270, target: 250 }
    ];
  }
}

```

MAIN.TS

```

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

```

Ranges

You can add a range using [ranges](#) property to the bullet chart.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { BulletChartModule } from '@syncfusion/ej2-angular-charts';
import { Component } from '@angular/core';
@Component({
  imports: [
    BulletChartModule
  ],
  providers: [],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-bulletchart valueField='value' targetField='target'
    title='Revenue'
                [minimum]='minimum' [maximum]='maximum' [interval]='interval'
    [dataSource]='data'>
        <e-bullet-range-collection>
          <e-bullet-range end='100' color='#ebeb'></e-bullet-range>
          <e-bullet-range end='200' color='#d8d8d8'></e-bullet-range>
          <e-bullet-range end='300' color='#7f7f7f'></e-bullet-range>
        </e-bullet-range-collection>
      </ejs-bulletchart>`
  })
  export class AppComponent {
    public minimum: number = 0;
    public maximum: number = 300;
    public interval: number = 50;
    public data: Object[] = [
      { value: 270, target: 250 }
    ];
  }
}

```

```
}

```

MAIN.TS

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

```

Enable Tooltip

The tooltip is useful when you cannot display information by using the data labels due to space constraints. You can enable tooltip by setting the enable property as true in [tooltip](#) object and by injecting `BulletTooltipService` into the `@NgModule.providers`.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { BulletChartModule } from '@syncfusion/ej2-angular-charts'
import { BulletTooltipService } from '@syncfusion/ej2-angular-charts'
import { Component } from '@angular/core';
@Component({
  imports: [
    BulletChartModule
  ],
  providers: [ BulletTooltipService ],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-bulletchart valueName='value' targetName='target'
    title='Revenue'
    [dataSource]='data'
    [minimum]='minimum' [maximum]='maximum' [interval]='interval'
    [tooltip]='tooltip'>
    <e-bullet-range-collection>
    <e-bullet-range end='100' color='#ebeb'></e-bullet-range>
    <e-bullet-range end='200' color='#d8d8d8'></e-bullet-range>
    <e-bullet-range end='300' color='#7f7f7f'></e-bullet-range>
    </e-bullet-range-collection>
  </ejs-bulletchart>`
})
export class AppComponent {
  public minimum: number = 0;
  public maximum: number = 300;
  public interval: number = 50;
  public tooltip: Object = {enable: true};
  public data: Object[] = [
    { value: 270, target: 250 }
  ];
}
```

MAIN.TS

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

```

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

Bullet chart dimensions in Angular Bullet chart component

Size for Container

The size of the Bullet Chart is determined by the container size, and it can be changed inline or via CSS as following.

```
<div style="width:650px; height:350px;">
<ejs-bulletchart id="app-container"></ejs-bulletchart>
</div>
```

```
`javascript
import { Component, OnInit } from '@angular/core';
@Component({
  selector: 'app-container',
  template:
    `<div style="width:650px; height:350px;">
    <ejs-bulletchart id="app-container"></ejs-bulletchart>
    </div>`
})
export class AppComponent {
  constructor(){
    /*
    */
  }
}
```

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { BulletChartModule } from '@syncfusion/ej2-angular-charts'
import { Component } from '@angular/core';
import { AnimationModel } from '@syncfusion/ej2-charts';
@Component({
  imports: [
    BulletChartModule
  ],
  providers: [ ],
```



```

standalone: true,
  selector: 'app-container',
  template: `<ejs-bulletedchart valueField='value' targetField='target'
[minimum]='minimum' [maximum]='maximum'
  [interval]='interval' [dataSource]='data' [animation]='animation'
width='650px' height='350px'>
    <e-bullet-range-collection>
      <e-bullet-range end='20' color='#ebebeb'></e-bullet-range>
      <e-bullet-range end='25' color='#d8d8d8'></e-bullet-range>
      <e-bullet-range end='30' color='#7f7f7f'></e-bullet-range>
    </e-bullet-range-collection>
  </ejs-bulletedchart>`
})
export class AppComponent {
  public minimum: number = 0;
  public maximum: number = 30;
  public interval: number = 5;
  public data: Object[] = [
    { value: 23, target: 22 }
  ];
  public animation: AnimationModel = { enable: false };
}

```

MAIN.TS

```

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

```

<!-- markdownlint-disable MD036 -->

Size for Bullet Chart

<!-- markdownlint-disable MD036 -->

The [width](#) and [height](#) properties are used to adjust the size of the Bullet Chart.

Pixel

Can set the size of the Bullet Chart in pixels as shown below.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { BulletChartModule } from '@syncfusion/ej2-angular-charts';
import { Component } from '@angular/core';
import { AnimationModel } from '@syncfusion/ej2-charts';
@Component({
  imports: [
    BulletChartModule
  ],
  providers: [ ],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-bulletedchart valueField='value' targetField='target'
[minimum]='minimum' [maximum]='maximum'

```

```

    [interval]='interval' [dataSource]='data' [animation]='animation'
    width='650' height='350'>
    <e-bullet-range-collection>
      <e-bullet-range end='20' color='#ebebeb'></e-bullet-range>
      <e-bullet-range end='25' color='#d8d8d8'></e-bullet-range>
      <e-bullet-range end='30' color='#7f7f7f'></e-bullet-range>
    </e-bullet-range-collection>
  </ejs-bulletchart>`
  })
  export class AppComponent {
    public minimum: number = 0;
    public maximum: number = 30;
    public interval: number = 5;
    public data: Object[] = [
      { value: 23, target: 22 }
    ];
    public animation: AnimationModel = { enable: false };
  }

```

MAIN.TS

```

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

```

Percentage

By setting a value in percentage, the Bullet Chart gets its dimension with respect to its container. For example, when the height is **50%**, the Bullet Chart renders to half of the container's height.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { BulletChartModule } from '@syncfusion/ej2-angular-charts';
import { Component } from '@angular/core';
import { AnimationModel } from '@syncfusion/ej2-charts';
@Component({
  imports: [
    BulletChartModule
  ],
  providers: [ ],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-bulletchart valueField='value' targetField='target'
    [minimum]='minimum' [maximum]='maximum'
    [interval]='interval' [dataSource]='data' [animation]='animation'
    width='70%' height='60%'>
    <e-bullet-range-collection>
      <e-bullet-range end='20' color='#ebebeb'></e-bullet-range>
      <e-bullet-range end='25' color='#d8d8d8'></e-bullet-range>
      <e-bullet-range end='30' color='#7f7f7f'></e-bullet-range>
    </e-bullet-range-collection>
  </ejs-bulletchart>`
})

```

```
export class AppComponent {
  public minimum: number = 0;
  public maximum: number = 30;
  public interval: number = 5;
  public data: Object[] = [
    { value: 23, target: 22 }
  ];
  public animation: AnimationModel = { enable: false };
}
```

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 size is not specified, the Bullet Chart will be rendered with a height of **126px** and a width of the window.

Axis customization in Angular Bullet chart component

MajorTickLines and MinorTickLines Customization

The following properties can be used to customize [majorTicklines](#) and [minorTicklines](#).

- **width** - Specifies the width of ticklines.
- **height** - Specifies the height of ticklines.
- **color** - Specifies the color of ticklines.
- **useRangeColor** - Specifies the color of ticklines and represents the color from corresponding range colors.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { BulletChartModule } from '@syncfusion/ej2-angular-charts';
import { Component } from '@angular/core';
import { AnimationModel } from '@syncfusion/ej2-charts';
@Component({
  imports: [
    BulletChartModule
  ],
  providers: [ ],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-bulletchart valueField='value' targetField='target'
title='Revenue' [minimum]='minimum' [maximum]='maximum'
[interval]='interval' [dataSource]='data' [animation]='animation'
[majorTickLines]='majorTickLines' [minorTickLines]='minorTickLines'>
  <e-bullet-range-collection>
    <e-bullet-range end='100' color='#ebebeb'></e-bullet-range>
    <e-bullet-range end='200' color='#d8d8d8'></e-bullet-range>
    <e-bullet-range end='300' color='#7f7f7f'></e-bullet-range>
  </e-bullet-range-collection>
```

```

</ejs-bulletedchart>`
  })
  export class AppComponent {
    public minimum: number = 0;
    public maximum: number = 300;
    public interval: number = 50;
    public data: Object[] = [
      { value: 270, target: 250 }
    ];
    public majorTickLines: Object = { color: 'blue', width: 5 };
    public minorTickLines: Object = { color: 'red', width: 5 };
    animation: any;
  }

```

MAIN.TS

```

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

```

Tick Placement

The major and the minor ticks can be placed **inside** or **outside** the ranges using the [tickPosition](#) property.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { BulletChartModule } from '@syncfusion/ej2-angular-charts'
import { Component } from '@angular/core';
import { AnimationModel } from '@syncfusion/ej2-charts';
@Component({
  imports: [
    BulletChartModule
  ],
  providers: [ ],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-bulletedchart valueField='value' targetField='target'
title='Revenue' [minimum]='minimum' [maximum]='maximum'
[interval]='interval' [dataSource]='data' [animation]='animation'
tickPosition='Inside'>
  <e-bullet-range-collection>
    <e-bullet-range end='20' color='#ebeb'></e-bullet-range>
    <e-bullet-range end='25' color='#d8d8d8'></e-bullet-range>
    <e-bullet-range end='30' color='#7f7f7f'></e-bullet-range>
  </e-bullet-range-collection>
</ejs-bulletedchart>`
  })
  export class AppComponent {
    public minimum: number = 0;
    public maximum: number = 30;
    public interval: number = 5;
    public data: Object[] = [

```

```

    { value: 23, target: 22 }
  ];
  public animation: AnimationModel = { enable: false };
}

```

MAIN.TS

```

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

```

Label Format

Axis numeric labels can be formatted by using the [labelFormat](#) property. Axis labels support all globalize formats.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { BulletChartModule } from '@syncfusion/ej2-angular-charts';
import { Component } from '@angular/core';
import { AnimationModel } from '@syncfusion/ej2-charts';
@Component({
  imports: [
    BulletChartModule
  ],
  providers: [],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-bulletedchart valueField='value' targetField='target'
title='Revenue' [minimum]='minimum' [maximum]='maximum'
[interval]='interval' [dataSource]='data' [animation]='animation'
labelFormat='c'>
  <e-bullet-range-collection>
    <e-bullet-range end='500' color='#ebebeb'></e-bullet-range>
    <e-bullet-range end='1000' color='#d8d8d8'></e-bullet-range>
    <e-bullet-range end='1500' color='#7f7f7f'></e-bullet-range>
  </e-bullet-range-collection>
</ejs-bulletedchart>`
})
export class AppComponent {
  public minimum: number = 0;
  public maximum: number = 1500;
  public interval: number = 500;
  public data: Object[] = [
    { value: 1100, target: 950 }
  ];
  public animation: AnimationModel = { enable: false };
}

```

MAIN.TS

```

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

```

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

The following table describes the result of applying some commonly used formats to numeric axis labels.

<!-- markdownlint-disable MD033 -->

Label Value	Label Format property value	Result	Description
1000	n1	1000.0	The Number is rounded to 1 decimal place
1000	n2	1000.00	The Number is rounded to 2 decimal place
1000	n3	1000.000	The Number is rounded to 3 decimal place
0.01	p1	1.0%	The Number is converted to percentage with 1 decimal place
0.01	p2	1.00%	The Number is converted to percentage with 2 decimal place
0.01	p3	1.000%	The Number is converted to percentage with 3 decimal place
1000	c1	\$1000.0	The Currency symbol is appended to number and number is rounded to 1 decimal place
1000	c2	\$1000.00	The Currency symbol is appended to number and number is rounded to 2 decimal place

GroupingSeparator

To separate the groups of thousands, set the [enableGroupSeparator](#) property to **true**.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { BulletChartModule } from '@syncfusion/ej2-angular-charts'
import { Component } from '@angular/core';
import { AnimationModel } from '@syncfusion/ej2-charts';
@Component({
  imports: [
    BulletChartModule
  ],
  providers: [ ],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-bulletchart valueField='value' targetField='target'
  title='Revenue' [minimum]='minimum' [maximum]='maximum'
  [interval]='interval' [dataSource]='data' [animation]='animation'
  enableGroupSeparator=true>
  <e-bullet-range-collection>
    <e-bullet-range end='500' color='#ebebeb'></e-bullet-range>
    <e-bullet-range end='1000' color='#d8d8d8'></e-bullet-range>
```

```

    <e-bullet-range end='1500' color='#7f7f7f'></e-bullet-range>
  </e-bullet-range-collection>
</ejs-bulletchart>`
  })
  export class AppComponent {
    public minimum: number = 0;
    public maximum: number = 1500;
    public interval: number = 500;
    public data: Object[] = [
      { value: 1100, target: 950 }
    ];
    public animation: AnimationModel = { enable: false };
  }

```

MAIN.TS

```

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

```

Custom Label Format

Using the [labelFormat](#) property, axis labels can be specified with a custom defined format in addition to the axis value. The label format uses a placeholder such as **\$(value)K**, which represents the axis label.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { BulletChartModule } from '@syncfusion/ej2-angular-charts'
import { Component } from '@angular/core';
import { AnimationModel } from '@syncfusion/ej2-charts';
@Component({
  imports: [
    BulletChartModule
  ],
  providers: [ ],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-bulletchart valueField='value' targetField='target'
title='Revenue' [minimum]='minimum' [maximum]='maximum'
[interval]='interval' [dataSource]='data' [animation]='animation'
[labelFormat]='labelFormat'>
  <e-bullet-range-collection>
    <e-bullet-range end='500' color='#ebebeb'></e-bullet-range>
    <e-bullet-range end='1000' color='#d8d8d8'></e-bullet-range>
    <e-bullet-range end='1500' color='#7f7f7f'></e-bullet-range>
  </e-bullet-range-collection>
</ejs-bulletchart>`
})
export class AppComponent {
  public minimum: number = 0;
  public maximum: number = 1500;
  public interval: number = 500;
  public data: Object[] = [

```

```

    { value: 1100, target: 950 }
  ];
  public labelFormat: string = '${value}K';
  public animation: AnimationModel = { enable: false };
}

```

MAIN.TS

```

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

```

Label Placement

Label can be placed **Inside** or **Outside** of the ranges using the [labelPosition](#) property.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { BulletChartModule } from '@syncfusion/ej2-angular-charts';
import { Component } from '@angular/core';
import { AnimationModel } from '@syncfusion/ej2-charts';
@Component({
  imports: [
    BulletChartModule
  ],
  providers: [ ],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-bulletchart valueField='value' targetField='target'
title='Revenue' [minimum]='minimum' [maximum]='maximum'
[interval]='interval' [dataSource]='data' [animation]='animation'
labelPosition='Inside'>
  <e-bullet-range-collection>
    <e-bullet-range end='500' color='#ebebeb'></e-bullet-range>
    <e-bullet-range end='1000' color='#d8d8d8'></e-bullet-range>
    <e-bullet-range end='1500' color='#7f7f7f'></e-bullet-range>
  </e-bullet-range-collection>
</ejs-bulletchart>`
})
export class AppComponent {
  public minimum: number = 0;
  public maximum: number = 1500;
  public interval: number = 500;
  public data: Object[] = [
    { value: 1100, target: 950 }
  ];
  public animation: AnimationModel = { enable: false };
}

```

MAIN.TS

```

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

```



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

Opposed Position

To place an axis opposite to its original position, set the [opposedPosition](#) property to **true**.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { BulletChartModule } from '@syncfusion/ej2-angular-charts'
import { Component } from '@angular/core';
import { AnimationModel } from '@syncfusion/ej2-charts';
@Component({
  imports: [
    BulletChartModule
  ],
  providers: [ ],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-bulletchart valueField='value' targetField='target'
title='Revenue' [minimum]='minimum' [maximum]='maximum'
[interval]='interval' [dataSource]='data' [animation]='animation'
opposedPosition=true>
  <e-bullet-range-collection>
    <e-bullet-range end='500' color='#ebebeb'></e-bullet-range>
    <e-bullet-range end='1000' color='#d8d8d8'></e-bullet-range>
    <e-bullet-range end='1500' color='#7f7f7f'></e-bullet-range>
  </e-bullet-range-collection>
</ejs-bulletchart>`
})
export class AppComponent {
  public minimum: number = 0;
  public maximum: number = 1500;
  public interval: number = 500;
  public data: Object[] = [
    { value: 1100, target: 950 }
  ];
  public animation: AnimationModel = { enable: false };
}
```

MAIN.TS

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

Category Label

The Bullet Chart supports X-axis label by specifying the property from the data source to the [categoryField](#). It helps to understand the input data in a more efficient way.

APP.COMPONENT.TS

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

```

import { BrowserModule } from '@angular/platform-browser'
import { BulletChartModule } from '@syncfusion/ej2-angular-charts'
import { Component } from '@angular/core';
import { AnimationModel } from '@syncfusion/ej2-charts';
@Component({
  imports: [
    BulletChartModule
  ],
  providers: [ ],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-bulletchart valueField='value' targetField='target'
title='Profit' [minimum]='minimum' [maximum]='maximum'
[interval]='interval' [dataSource]='data' [animation]='animation'
categoryField='category'>
  <e-bullet-range-collection>
    <e-bullet-range end='500' color='#ebebeb'></e-bullet-range>
    <e-bullet-range end='1000' color='#d8d8d8'></e-bullet-range>
    <e-bullet-range end='1500' color='#7f7f7f'></e-bullet-range>
  </e-bullet-range-collection>
</ejs-bulletchart>`
})
export class AppComponent {
  public minimum: number = 0;
  public maximum: number = 1500;
  public interval: number = 500;
  public data: Object[] = [
    { value: 1100, target: 950, category: 'Product A' }
  ];
  public animation: AnimationModel = { enable: false };
}

```

MAIN.TS

```

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

```

Category Label Customization

The label color, opacity, font size, font family, font weight, and font style can be customized by using the [categoryLabelStyle](#) setting for category and the [labelStyle](#) setting for axis label. The [useRangeColor](#) property specifies the color of the axis label and represents the color from the corresponding range colors.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { BulletChartModule } from '@syncfusion/ej2-angular-charts'
import { Component } from '@angular/core';
import { AnimationModel } from '@syncfusion/ej2-charts';
@Component({
  imports: [
    BulletChartModule
  ],
  providers: [ ],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-bulletchart valueField='value' targetField='target'
title='Profit' [minimum]='minimum' [maximum]='maximum'
[interval]='interval' [dataSource]='data' [animation]='animation'
categoryField='category'>
  <e-bullet-range-collection>
    <e-bullet-range end='500' color='#ebebeb'></e-bullet-range>
    <e-bullet-range end='1000' color='#d8d8d8'></e-bullet-range>
    <e-bullet-range end='1500' color='#7f7f7f'></e-bullet-range>
  </e-bullet-range-collection>
</ejs-bulletchart>`
})
export class AppComponent {
  public minimum: number = 0;
  public maximum: number = 1500;
  public interval: number = 500;
  public data: Object[] = [
    { value: 1100, target: 950, category: 'Product A' }
  ];
  public animation: AnimationModel = { enable: false };
}

```

```

    ],
    providers: [ ],
    standalone: true,
    selector: 'app-container',
    template: `<ejs-bulletchart valueField='value' targetField='target'
title='Profit' [minimum]='minimum' [maximum]='maximum'
[interval]='interval' [dataSource]='data' [animation]='animation'
categoryField='category' [categoryLabelStyle]='categoryLabelStyle'>
    <e-bullet-range-collection>
        <e-bullet-range end='500' color='#ebebeb'></e-bullet-range>
        <e-bullet-range end='1000' color='#d8d8d8'></e-bullet-range>
        <e-bullet-range end='1500' color='#7f7f7f'></e-bullet-range>
    </e-bullet-range-collection>
</ejs-bulletchart>`
  })
  export class AppComponent {
    public minimum: number = 0;
    public maximum: number = 1500;
    public interval: number = 500;
    public data: Object[] = [
      { value: 1100, target: 950, category: 'Product A' }
    ];
    public animation: AnimationModel = { enable: false };
    public categoryLabelStyle: Object = { color: 'Orange' };
  }
}

```

MAIN.TS

```

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

```

<!-- markdownlint-disable MD036 -->

Data binding in Angular Bullet chart component

Bullet Chart can visualise data bound from local or remote data.

Local Data

The [dataSource](#) property accepts a collection of values as input that helps to display measures, and compares them to a target bar. To display the actual and target bar, specify the property from the datasource into the [valueField](#) and [targetField](#) respectively.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { BulletChartModule } from '@syncfusion/ej2-angular-charts'
import { Component } from '@angular/core';
import { AnimationModel } from '@syncfusion/ej2-charts';
@Component({
  imports: [
    BulletChartModule
  ],
  providers: [ ],

```

```

standalone: true,
  selector: 'app-container',
  template: `<ejs-bulletedchart id='localData' valueField='completedStories'
targetField='requiredStories'
category='name' height='400px' [minimum]='minimum' [maximum]='maximum'
[interval]='interval'
title='Sprint Planning' titlePosition='Top' [dataSource]='data'
[animation]='animation'>
<e-bullet-range-collection>
  <e-bullet-range end='25' color='#ebebeb'></e-bullet-range>
  <e-bullet-range end='37' color='#d8d8d8'></e-bullet-range>
  <e-bullet-range end='45' color='#7f7f7f'></e-bullet-range>
</e-bullet-range-collection>
</ejs-bulletedchart>`
  })
export class AppComponent {
  public minimum: number = 5;
  public maximum: number = 45;
  public interval: number = 5;
  public animation: AnimationModel = { enable: false }
  public data: Object[] = [
    {
      requiredStories: 20,
      completedStories: 25,
      name: 'David'
    },
    {
      requiredStories: 25,
      completedStories: 20,
      name: 'Asif'
    },
    {
      requiredStories: 15,
      completedStories: 10,
      name: 'Thomas'
    },
    {
      requiredStories: 40,
      completedStories: 39,
      name: 'Rohit'
    },
    {
      requiredStories: 35,
      completedStories: 40,
      name: 'Virat'
    },
    {
      requiredStories: 28,
      completedStories: 25,
      name: 'Jude'
    },
    {
      requiredStories: 10,
      completedStories: 18,
      name: 'Warner'
    }
  ]
}

```

```

        requiredStories: 30,
        completedStories: 28,
        name: 'Malik'
    }
];
}

```

MAIN.TS

```

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

```

<!-- markdownlint-disable MD036 -->

Ranges in Angular Bullet chart component

Ranges represent the quality of a specific range such as **Good**, **Bad** and **Satisfactory** in the Bullet Chart scale. The ending point of a qualitative range is specified in the [end](#) property in [ranges](#). The [minimum](#) value of a quantitative scale is considered the starting point of the first range or the previous range end point.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { BulletChartModule } from '@syncfusion/ej2-angular-charts'
import { Component } from '@angular/core';
import { AnimationModel } from '@syncfusion/ej2-charts';
@Component({
  imports: [
    BulletChartModule
  ],
  providers: [ ],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-bulletchart targetField='target' valueField='value'
categoryField='category' [minimum]='minimum' [maximum]='maximum'
[interval]='interval' [dataSource]='data' [animation]='animation'
height='400' [categoryLabelStyle]='categoryLabelStyle'
title='Sales Rate'>
  <e-bullet-range-collection>
    <e-bullet-range end='35' color='#ebeb'></e-bullet-range>
    <e-bullet-range end='70' color='#d8d8d8'></e-bullet-range>
    <e-bullet-range end='100' color='#7f7f7f'></e-bullet-range>
  </e-bullet-range-collection>
</ejs-bulletchart>`
})
export class AppComponent {
  public minimum: number = 0;
  public maximum: number = 100;
  public interval: number = 20;
  public data: Object[] = [
    { value: 55, target: 75, category: 'Year 1' },
    { value: 70, target: 70, category: 'Year 2' },

```

```

    { value: 85, target: 75, category: 'Year 3' }
  ];
  public animation: AnimationModel = { enable: false };
  public categoryLabelStyle: Object = { color: 'red', size: '13', fontWeight:
'bold'};
}

```

MAIN.TS

```

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

```

Color Customization

Enhance the readability of ranges with color and opacity. It can be applied using the [color](#) and [opacity](#) properties in [ranges](#).

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { BulletChartModule } from '@syncfusion/ej2-angular-charts'
import { Component } from '@angular/core';
import { AnimationModel } from '@syncfusion/ej2-charts';
@Component({
  imports: [
    BulletChartModule
  ],
  providers: [ ],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-bulletchart targetField='target' valueField='value'
categoryField='category' [minimum]='minimum' [maximum]='maximum'
[interval]='interval' [dataSource]='data' [animation]='animation'
height='400' [categoryLabelStyle]='categoryLabelStyle'
title='Sales Rate'>
  <e-bullet-range-collection>
    <e-bullet-range end='35' color='darkred'></e-bullet-range>
    <e-bullet-range end='50' color='red'></e-bullet-range>
    <e-bullet-range end='75' color='blue'></e-bullet-range>
    <e-bullet-range end='90' color='lightgreen'></e-bullet-range>
    <e-bullet-range end='100' color='green'></e-bullet-range>
  </e-bullet-range-collection>
</ejs-bulletchart>`
})
export class AppComponent {
  public minimum: number = 0;
  public maximum: number = 100;
  public interval: number = 20;
  public data: Object[] = [
    { value: 55, target: 75, category: 'Year 1' },
    { value: 70, target: 70, category: 'Year 2' },
    { value: 85, target: 75, category: 'Year 3' }
  ];
}

```

```

    public animation: AnimationModel = { enable: false };
    public categoryLabelStyle: Object = { color: 'red', size: '13', fontWeight:
'bold'};
}

```

MAIN.TS

```

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

```

<!-- markdownlint-disable MD036 -->

Value bar in Angular Bullet chart component

To display the primary data or the current value of the data being measured known as the **Feature Measure** that should be encoded as a bar. This is called as the **Actual Bar** or the **Feature Bar** in the Bullet Chart, and to display the actual bar the [valueField](#) should be mapped to the appropriate field from the data source.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { BulletChartModule } from '@syncfusion/ej2-angular-charts';
import { Component } from '@angular/core';
import { AnimationModel } from '@syncfusion/ej2-charts';
@Component({
  imports: [
    BulletChartModule
  ],
  providers: [ ],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-bulletchart valueField='value' [minimum]='minimum'
[maximum]='maximum'
[interval]='interval' [dataSource]='data' [animation]='animation'
targetWidth=15 [tooltip]='tooltip'>
  <e-bullet-range-collection>
    <e-bullet-range end='35' color='#ebebcb'></e-bullet-range>
    <e-bullet-range end='70' color='#d8d8d8'></e-bullet-range>
    <e-bullet-range end='100' color='#7f7f7f'></e-bullet-range>
  </e-bullet-range-collection>
</ejs-bulletchart>`
})
export class AppComponent {
  public minimum: number = 0;
  public maximum: number = 100;
  public interval: number = 20;
  public data: Object[] = [
    { value: 55, target: 75 }
  ];
  public animation: AnimationModel = { enable: false };
  public tooltip: Object = { enable: false };
}

```

MAIN.TS

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

Types of actual bar

The shape of the actual bar can be customized using the [type](#) property of the Bullet Chart. The actual bar contains **Rect** and **Dot** shapes. By default, the actual bar shape is Rect.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { BulletChartModule } from '@syncfusion/ej2-angular-charts'
import { Component } from '@angular/core';
import { AnimationModel } from '@syncfusion/ej2-charts';
@Component({
  imports: [
    BulletChartModule
  ],
  providers: [ ],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-bulletchart valueField='value' [minimum]='minimum'
[maximum]='maximum'
[interval]='interval' [dataSource]='data' [animation]='animation' type=
'Dot' [tooltip]='tooltip'>
  <e-bullet-range-collection>
    <e-bullet-range end='35' color='#ebebeb'></e-bullet-range>
    <e-bullet-range end='70' color='#d8d8d8'></e-bullet-range>
    <e-bullet-range end='100' color='#7f7f7f'></e-bullet-range>
  </e-bullet-range-collection>
</ejs-bulletchart>`
})
export class AppComponent {
  public minimum: number = 0;
  public maximum: number = 100;
  public interval: number = 20;
  public data: Object[] = [
    { value: 55, target: 75 }
  ];
  public animation: AnimationModel = { enable: false };
  public tooltip: Object = { enable: false };
}
```

MAIN.TS

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


Actual bar customization

Border customization

Using the [valueBorder](#) property of the bullet chart, you can customize the border [color](#) and [width](#) of the actual bar.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { BulletChartModule } from '@syncfusion/ej2-angular-charts'
import { Component } from '@angular/core';
import { AnimationModel } from '@syncfusion/ej2-charts';
@Component({
  imports: [
    BulletChartModule
  ],
  providers: [ ],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-bulletchart valueField='value' [minimum]='minimum'
[maximum]='maximum'
[interval]='interval' [dataSource]='data'
[animation]='animation' [valueBorder]='valueBorder' [tooltip]='tooltip'>
  <e-bullet-range-collection>
    <e-bullet-range end='35' color='#ebebeb'></e-bullet-range>
    <e-bullet-range end='70' color='#d8d8d8'></e-bullet-range>
    <e-bullet-range end='100' color='#7f7f7f'></e-bullet-range>
  </e-bullet-range-collection>
</ejs-bulletchart>`
})
export class AppComponent {
  public minimum: number = 0;
  public maximum: number = 100;
  public interval: number = 20;
  public data: Object[] = [
    { value: 55, target: 75 }
  ];
  public animation: AnimationModel = { enable: false };
  public valueBorder: Object = { color: 'red', width: 3 };
  public tooltip: Object = { enable: false };
}

```

MAIN.TS

```

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

```

Fill color and height Customization

Customize the fill color and height of the actual bar using the [valueFill](#) and [valueHeight](#) properties of the bullet chart. Also, you can bind the color for the actual bar from [dataSource](#) for the bullet chart using [valueFill](#) property.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { BulletChartModule } from '@syncfusion/ej2-angular-charts'
import { Component } from '@angular/core';
import { AnimationModel } from '@syncfusion/ej2-charts';
@Component({
  imports: [
    BulletChartModule
  ],
  providers: [ ],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-bulletchart valueField='value' [minimum]='minimum'
[maximum]='maximum'
[interval]='interval' [dataSource]='data' [animation]='animation'
valueFill='color' [tooltip]='tooltip'
[valueHeight]='valueHeight'>
  <e-bullet-range-collection>
    <e-bullet-range end='35' color='#ebebcb'></e-bullet-range>
    <e-bullet-range end='70' color='#d8d8d8'></e-bullet-range>
    <e-bullet-range end='100' color='#7f7f7f'></e-bullet-range>
  </e-bullet-range-collection>
</ejs-bulletchart>`
})
export class AppComponent {
  public minimum: number = 0;
  public maximum: number = 100;
  public interval: number = 20;
  public data: Object[] = [
    { value: 55, target: 75, color: 'blue' }
  ];
  public animation: AnimationModel = { enable: false };
  public valueBorder: Object = { color: 'red', width: 3 };
  public tooltip: Object = { enable: false };
  public valueHeight: number = 15;
}

```

MAIN.TS

```

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

```

<!-- markdownlint-disable MD036 -->

Comparative bar in Angular Bullet chart component

The line marker that runs perpendicular to the orientation of the graph is known as the **Comparative Measure** and it is used as a target marker to compare against the feature measure value. This is also called as the **Target Bar** in the Bullet Chart. To display the target bar, the [targetField](#) should be mapped to the appropriate field from the datasource.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'

```

```

import { BrowserModule } from '@angular/platform-browser'
import { BulletChartModule } from '@syncfusion/ej2-angular-charts'
import { Component } from '@angular/core';
import { AnimationModel } from '@syncfusion/ej2-charts';
@Component({
  imports: [
    BulletChartModule
  ],
  providers: [ ],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-bulletchart targetField='target' [minimum]='minimum'
[maximum]='maximum'
[interval]='interval' [dataSource]='data' [animation]='animation'
[tooltip]='tooltip'>
  <e-bullet-range-collection>
    <e-bullet-range end='35' color='#ebebeb'></e-bullet-range>
    <e-bullet-range end='70' color='#d8d8d8'></e-bullet-range>
    <e-bullet-range end='100' color='#7f7f7f'></e-bullet-range>
  </e-bullet-range-collection>
</ejs-bulletchart>`
})
export class AppComponent {
  public minimum: number = 0;
  public maximum: number = 100;
  public interval: number = 20;
  public data: Object[] = [
    { value: 55, target: 75 }
  ];
  public animation: AnimationModel = { enable: false };
  public tooltip: Object = { enable: false };
}

```

MAIN.TS

```

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

```

Types of target bar

The shape of the target bar can be customized using the [targetTypes](#) property and it supports **Circle**, **Cross**, and **Rect** shapes. The default type of the target bar is **Rect**.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { BulletChartModule } from '@syncfusion/ej2-angular-charts'
import { Component } from '@angular/core';
import { AnimationModel } from '@syncfusion/ej2-charts';
@Component({
  imports: [
    BulletChartModule
  ],

```

```

providers: [ ],
standalone: true,
  selector: 'app-container',
  template: `<ejs-bulletedchart targetField='target' [minimum]='minimum'
[maximum]='maximum'
  [interval]='interval' [dataSource]='data' [animation]='animation'
[targetTypes]='targetTypes' [tooltip]='tooltip'>
    <e-bullet-range-collection>
      <e-bullet-range end='35' color='#ebeb'></e-bullet-range>
      <e-bullet-range end='70' color='#d8d8d8'></e-bullet-range>
      <e-bullet-range end='100' color='#7f7f7f'></e-bullet-range>
    </e-bullet-range-collection>
  </ejs-bulletedchart>`
})
export class AppComponent {
  public minimum: number = 0;
  public maximum: number = 100;
  public interval: number = 20;
  public data: Object[] = [
    { value: 55, target: 75 }
  ];
  public targetTypes: string[] = ['Circle'];
  public animation: AnimationModel = { enable: false };
  public tooltip: Object = { enable: false };
}

```

MAIN.TS

```

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

```

Target bar customization

The following properties can be used to customize the target bar. Also, you can bind the color for the target bar from [dataSource](#) for the bullet chart.

- [targetColor](#) - Specifies the fill color of Target Bar.
- [targetWidth](#) - Specifies the width of Target Bar.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { BulletChartModule } from '@syncfusion/ej2-angular-charts';
import { Component } from '@angular/core';
import { AnimationModel } from '@syncfusion/ej2-charts';
@Component({
  imports: [
    BulletChartModule
  ],
  providers: [ ],
  standalone: true,
  selector: 'app-container',

```

```

    template: `<ejs-bulletchart targetField='target' [minimum]='minimum'
[maximum]='maximum'
    [interval]='interval' [dataSource]='data' [animation]='animation'
targetColor='color' targetWidth=15 [tooltip]='tooltip'>
    <e-bullet-range-collection>
        <e-bullet-range end='35' color='#ebebeb'></e-bullet-range>
        <e-bullet-range end='70' color='#d8d8d8'></e-bullet-range>
        <e-bullet-range end='100' color='#7f7f7f'></e-bullet-range>
    </e-bullet-range-collection>
</ejs-bulletchart>`
  })
  export class AppComponent {
    public minimum: number = 0;
    public maximum: number = 100;
    public interval: number = 20;
    public data: Object[] = [
      { value: 55, target: 75, color: 'red' }
    ];
    public animation: AnimationModel = { enable: false };
    public tooltip: Object = { enable: false };
  }

```

MAIN.TS

```

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

```

<!-- markdownlint-disable MD036 -->

Title in Angular Bullet chart component

Title

The title of the Bullet Chart displays the information about the data plotted by specifying it in the [title](#) property.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { BulletChartModule } from '@syncfusion/ej2-angular-charts'
import { Component } from '@angular/core';
import { AnimationModel } from '@syncfusion/ej2-charts';
@Component({
  imports: [
    BulletChartModule
  ],
  providers: [ ],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-bulletchart valueField='value' targetField='target'
[minimum]='minimum' [maximum]='maximum'
    [interval]='interval' [dataSource]='data' [animation]='animation'
title='Sales Rate'>
    <e-bullet-range-collection>

```

```

    <e-bullet-range end='35' color='#ebebeb'></e-bullet-range>
    <e-bullet-range end='70' color='#d8d8d8'></e-bullet-range>
    <e-bullet-range end='100' color='#7f7f7f'></e-bullet-range>
  </e-bullet-range-collection>
</ejs-bulletchart>`
})
export class AppComponent {
  public minimum: number = 0;
  public maximum: number = 100;
  public interval: number = 20;
  public data: Object[] = [
    { value: 55, target: 45 }
  ];
  public animation: AnimationModel = { enable: false };
}

```

MAIN.TS

```

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

```

Subtitle

To show additional information about the data plotted, the Bullet Chart can also be given a subtitle using the [subtitle](#) property.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { BulletChartModule } from '@syncfusion/ej2-angular-charts'
import { Component } from '@angular/core';
import { AnimationModel } from '@syncfusion/ej2-charts';
@Component({
  imports: [
    BulletChartModule
  ],
  providers: [ ],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-bulletchart valueField='value' targetField='target'
[minimum]='minimum' [maximum]='maximum'
[interval]='interval' [dataSource]='data' [animation]='animation'
title='Sales Rate' subtitle='(in dollars $)'>
  <e-bullet-range-collection>
    <e-bullet-range end='35' color='#ebebeb'></e-bullet-range>
    <e-bullet-range end='70' color='#d8d8d8'></e-bullet-range>
    <e-bullet-range end='100' color='#7f7f7f'></e-bullet-range>
  </e-bullet-range-collection>
</ejs-bulletchart>`
})
export class AppComponent {
  public minimum: number = 0;
  public maximum: number = 100;
}

```

```

public interval: number = 20;
public data: Object[] = [
  { value: 55, target: 45 }
];
public animation: AnimationModel = { enable: false };
}

```

MAIN.TS

```

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

```

Title and SubTitle Position

The title and the subtitle positions can be customized using the [titlePosition](#) property. Possible positions are **Left**, **Right**, **Top**, and **Bottom**.

Position as Left

By setting the [titlePosition](#) to **Left**, you can display the title and subtitle at the left side of the Bullet Chart.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { BulletChartModule } from '@syncfusion/ej2-angular-charts';
import { Component } from '@angular/core';
import { AnimationModel } from '@syncfusion/ej2-charts';
@Component({
  imports: [
    BulletChartModule
  ],
  providers: [ ],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-bulletchart valueField='value' targetField='target'
[minimum]='minimum' [maximum]='maximum'
[interval]='interval' [dataSource]='data' [animation]='animation'
title='Sales Rate' subtitle='(in dollars $)'
titlePosition='Left'>
<e-bullet-range-collection>
  <e-bullet-range end='35' color='#ebebeb'></e-bullet-range>
  <e-bullet-range end='70' color='#d8d8d8'></e-bullet-range>
  <e-bullet-range end='100' color='#7f7f7f'></e-bullet-range>
</e-bullet-range-collection>
</ejs-bulletchart>`
})
export class AppComponent {
  public minimum: number = 0;
  public maximum: number = 100;
  public interval: number = 20;
  public data: Object[] = [
    { value: 55, target: 45 }
  ];
}

```

```
public animation: AnimationModel = { enable: false };
}
```

MAIN.TS

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

Position as Right

By setting the [titlePosition](#) to **Right**, you can display the title and subtitle at the right side of the Bullet Chart.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { BulletChartModule } from '@syncfusion/ej2-angular-charts';
import { Component } from '@angular/core';
import { AnimationModel } from '@syncfusion/ej2-charts';
@Component({
  imports: [
    BulletChartModule
  ],
  providers: [],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-bulletchart valueField='value' targetField='target'
[minimum]='minimum' [maximum]='maximum'
[interval]='interval' [dataSource]='data' [animation]='animation'
title='Sales Rate' subtitle='(in dollars $)'
titlePosition='Right'>
<e-bullet-range-collection>
  <e-bullet-range end='35' color='#ebebeb'></e-bullet-range>
  <e-bullet-range end='70' color='#d8d8d8'></e-bullet-range>
  <e-bullet-range end='100' color='#7f7f7f'></e-bullet-range>
</e-bullet-range-collection>
</ejs-bulletchart>`
})
export class AppComponent {
  public minimum: number = 0;
  public maximum: number = 100;
  public interval: number = 20;
  public data: Object[] = [
    { value: 55, target: 45 }
  ];
  public animation: AnimationModel = { enable: false };
}
```

MAIN.TS

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



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

Position as Top

By setting the [titlePosition](#) to **Top**, you can display the title and subtitle at the top of the Bullet Chart. The default title and subtitle positions of the Bullet Chart is **Top**.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { BulletChartModule } from '@syncfusion/ej2-angular-charts'
import { Component } from '@angular/core';
import { AnimationModel } from '@syncfusion/ej2-charts';
@Component({
  imports: [
    BulletChartModule
  ],
  providers: [ ],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-bulletchart valueField='value' targetField='target'
[minimum]='minimum' [maximum]='maximum'
[interval]='interval' [dataSource]='data' [animation]='animation'
title='Sales Rate' subtitle='(in dollars $)'
titlePosition='Top'>
<e-bullet-range-collection>
  <e-bullet-range end='35' color='#ebebeb'></e-bullet-range>
  <e-bullet-range end='70' color='#d8d8d8'></e-bullet-range>
  <e-bullet-range end='100' color='#7f7f7f'></e-bullet-range>
</e-bullet-range-collection>
</ejs-bulletchart>`
})
export class AppComponent {
  public minimum: number = 0;
  public maximum: number = 100;
  public interval: number = 20;
  public data: Object[] = [
    { value: 55, target: 45 }
  ];
  public animation: AnimationModel = { enable: false };
}
```

MAIN.TS

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

Position as Bottom

By setting the [titlePosition](#) to **Bottom**, you can display the title and subtitle at the bottom of the Bullet Chart.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { BulletChartModule } from '@syncfusion/ej2-angular-charts'
import { Component } from '@angular/core';
import { AnimationModel } from '@syncfusion/ej2-charts';
@Component({
  imports: [
    BulletChartModule
  ],
  providers: [ ],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-bulletchart valueField='value' targetField='target'
[minimum]='minimum' [maximum]='maximum'
[interval]='interval' [dataSource]='data' [animation]='animation'
title='Sales Rate' subtitle='(in dollars $)'
titlePosition='Bottom'>
<e-bullet-range-collection>
  <e-bullet-range end='35' color='#ebebcb'></e-bullet-range>
  <e-bullet-range end='70' color='#d8d8d8'></e-bullet-range>
  <e-bullet-range end='100' color='#7f7f7f'></e-bullet-range>
</e-bullet-range-collection>
</ejs-bulletchart>`
})
export class AppComponent {
  public minimum: number = 0;
  public maximum: number = 100;
  public interval: number = 20;
  public data: Object[] = [
    { value: 55, target: 45 }
  ];
  public animation: AnimationModel = { enable: false };
}

```

MAIN.TS

```

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

```

Title Customization

The title color, opacity, font size, font family, font weight, and font style can be customized using the [titleStyle](#) property.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { BulletChartModule } from '@syncfusion/ej2-angular-charts'
import { Component } from '@angular/core';
import { AnimationModel } from '@syncfusion/ej2-charts';
@Component({
  imports: [
    BulletChartModule
  ],
  providers: [ ],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-bulletchart valueField='value' targetField='target'
[minimum]='minimum' [maximum]='maximum'
[interval]='interval' [dataSource]='data' [animation]='animation'
title='Sales Rate' subtitle='(in dollars $)'
titlePosition='Bottom'>
<e-bullet-range-collection>
  <e-bullet-range end='35' color='#ebebcb'></e-bullet-range>
  <e-bullet-range end='70' color='#d8d8d8'></e-bullet-range>
  <e-bullet-range end='100' color='#7f7f7f'></e-bullet-range>
</e-bullet-range-collection>
</ejs-bulletchart>`
})
export class AppComponent {
  public minimum: number = 0;
  public maximum: number = 100;
  public interval: number = 20;
  public data: Object[] = [
    { value: 55, target: 45 }
  ];
  public animation: AnimationModel = { enable: false };
}

```

```

    ],
    providers: [ ],
    standalone: true,
    selector: 'app-container',
    template: `<ejs-bulletchart valueField='value' targetField='target'
[minimum]='minimum' [maximum]='maximum'
[interval]='interval' [dataSource]='data' [animation]='animation'
title='Sales Rate' subtitle='(in dollars $)'
[titleStyle]='titleStyle'>
    <e-bullet-range-collection>
        <e-bullet-range end='35' color='#ebebeb'></e-bullet-range>
        <e-bullet-range end='70' color='#d8d8d8'></e-bullet-range>
        <e-bullet-range end='100' color='#7f7f7f'></e-bullet-range>
    </e-bullet-range-collection>
</ejs-bulletchart>`
  ))
export class AppComponent {
  public minimum: number = 0;
  public maximum: number = 100;
  public interval: number = 20;
  public data: Object[] = [
    { value: 55, target: 45 }
  ];
  public animation: AnimationModel = { enable: false };
  public titleStyle: Object = { size: '22px', color: 'red', fontFamily:
'cursive', fontWeight: 'Bold' };
}

```

MAIN.TS

```

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

```

SubTitle Customization

The sub-title color, opacity, font size, font family, font weight, and font style can be customized using the [subtitleStyle](#) property.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { BulletChartModule } from '@syncfusion/ej2-angular-charts'
import { Component } from '@angular/core';
import { AnimationModel } from '@syncfusion/ej2-charts';
@Component({
  imports: [
    BulletChartModule
  ],
  providers: [ ],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-bulletchart valueField='value' targetField='target'
[minimum]='minimum' [maximum]='maximum'

```

```

    [interval]='interval' [dataSource]='data' [animation]='animation'
    title='Sales Rate' subtitle='(in dollars $)'
    [subtitleStyle]='subtitleStyle'>
    <e-bullet-range-collection>
      <e-bullet-range end='35' color='#ebebeb'></e-bullet-range>
      <e-bullet-range end='70' color='#d8d8d8'></e-bullet-range>
      <e-bullet-range end='100' color='#7f7f7f'></e-bullet-range>
    </e-bullet-range-collection>
  </ejs-bulletchart>`
  })
  export class AppComponent {
    public minimum: number = 0;
    public maximum: number = 100;
    public interval: number = 20;
    public data: Object[] = [
      { value: 55, target: 45 }
    ];
    public animation: AnimationModel = { enable: false };
    public subtitleStyle: Object = { size: '22px', color: 'red', fontFamily:
    'cursive', fontWeight: 'Bold' };
  }

```

MAIN.TS

```

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

```

<!-- markdownlint-disable MD036 -->

Customization in Angular Bullet chart component

Orientation

The Bullet Chart can be rendered in different orientations such as **Horizontal** or **Vertical** via the [orientation](#) property. By default, the Bullet Chart is rendered in the **Horizontal** orientation.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { BulletChartModule } from '@syncfusion/ej2-angular-charts'
import { Component } from '@angular/core';
import { AnimationModel } from '@syncfusion/ej2-charts';
@Component({
  imports: [
    BulletChartModule
  ],
  providers: [ ],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-bulletchart targetField='target' valueField='value'
  [minimum]='minimum' [maximum]='maximum'
  [interval]='interval' [dataSource]='data' [animation]='animation'
  title='Sales Rate in dollars'
  subtitle='(in dollars $)' width='25%' orientation='Vertical'>

```

```

    <e-bullet-range-collection>
      <e-bullet-range end='35' color='#ebebeb'></e-bullet-range>
      <e-bullet-range end='70' color='#d8d8d8'></e-bullet-range>
      <e-bullet-range end='100' color='#7f7f7f'></e-bullet-range>
    </e-bullet-range-collection>
  </ejs-bulletchart>`
  })
  export class AppComponent {
    public minimum: number = 0;
    public maximum: number = 100;
    public interval: number = 20;
    public data: Object[] = [
      { value: 55, target: 75 }
    ];
    public animation: AnimationModel = { enable: false };
  }

```

MAIN.TS

```

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

```

Right-to-left (RTL)

The Bullet Chart supports the right-to-left rendering that can be enabled by setting the [enableRtl](#) property to **true**.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { BulletChartModule } from '@syncfusion/ej2-angular-charts'
import { Component } from '@angular/core';
import { AnimationModel } from '@syncfusion/ej2-charts';
@Component({
  imports: [
    BulletChartModule
  ],
  providers: [ ],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-bulletchart targetField='target' valueField='value'
[minimum]='minimum' [maximum]='maximum'
[interval]='interval' [dataSource]='data' [animation]='animation'
title='Sales Rate in dollars'
subtitle='(in dollars $)' enableRtl=true>
    <e-bullet-range-collection>
      <e-bullet-range end='35' color='#ebebeb'></e-bullet-range>
      <e-bullet-range end='70' color='#d8d8d8'></e-bullet-range>
      <e-bullet-range end='100' color='#7f7f7f'></e-bullet-range>
    </e-bullet-range-collection>
  </ejs-bulletchart>`
})
export class AppComponent {

```

```

public minimum: number = 0;
public maximum: number = 100;
public interval: number = 20;
public data: Object[] = [
  { value: 55, target: 75 }
];
public animation: AnimationModel = { enable: false };
}

```

MAIN.TS

```

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

```

Animation

The actual and the target bar supports the linear animation via the [animation](#) setting. The speed and the delay are controlled using the [duration](#) and [delay](#) properties respectively.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { BulletChartModule } from '@syncfusion/ej2-angular-charts';
import { Component } from '@angular/core';
import { AnimationModel } from '@syncfusion/ej2-charts';
@Component({
  imports: [
    BulletChartModule
  ],
  providers: [ ],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-bulletchart valueField='value' targetField='target'
[minimum]='minimum' [maximum]='maximum'
[interval]='interval' [dataSource]='data' [animation]='animation'>
<e-bullet-range-collection>
  <e-bullet-range end='35' color='#ebebeb'></e-bullet-range>
  <e-bullet-range end='70' color='#d8d8d8'></e-bullet-range>
  <e-bullet-range end='100' color='#7f7f7f'></e-bullet-range>
</e-bullet-range-collection>
</ejs-bulletchart>`
})
export class AppComponent {
  public minimum: number = 0;
  public maximum: number = 100;
  public interval: number = 20;
  public data: Object[] = [
    { value: 55, target: 75 }
  ];
  public animation: AnimationModel = { enable: true };
}

```

MAIN.TS

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

Theme

The Bullet Chart supports different type of themes via the [theme](#) property.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { BulletChartModule } from '@syncfusion/ej2-angular-charts'
import { Component } from '@angular/core';
import { AnimationModel } from '@syncfusion/ej2-charts';
@Component({
  imports: [
    BulletChartModule
  ],
  providers: [ ],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-bulletchart targetField='target' valueField='value'
[minimum]='minimum' [maximum]='maximum'
[interval]='interval' [dataSource]='data' [animation]='animation'
title='Sales Rate in dollars'
subtitle='(in dollars $)' theme='HighContrast'>
<e-bullet-range-collection>
  <e-bullet-range end='35' color='#ebebeb'></e-bullet-range>
  <e-bullet-range end='70' color='#d8d8d8'></e-bullet-range>
  <e-bullet-range end='100' color='#7f7f7f'></e-bullet-range>
</e-bullet-range-collection>
</ejs-bulletchart>`
})
export class AppComponent {
  public minimum: number = 0;
  public maximum: number = 100;
  public interval: number = 20;
  public data: Object[] = [
    { value: 55, target: 75 }
  ];
  public animation: AnimationModel = { enable: false };
}
```

MAIN.TS

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

<!-- markdownlint-disable MD036 -->

Data label in Angular Bullet chart component

Data Labels are used to identify the value of actual bar in the Bullet Chart component. The Data Labels will be shown by specifying the [dataLabel](#) setting's [enable](#) property to **true**.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { BulletChartModule } from '@syncfusion/ej2-angular-charts'
import { Component } from '@angular/core';
import { AnimationModel } from '@syncfusion/ej2-charts';
@Component({
  imports: [
    BulletChartModule
  ],
  providers: [ ],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-bulletedchart id='localData' valueField='completedStories'
targetField='requiredStories'
category='name' height='400px' [minimum]='minimum' [maximum]='maximum'
[interval]='interval'
title='Sprint Planning' titlePosition='Top' [dataSource]='data'
[animation]='animation' [dataLabel]='dataLabel'>
<e-bullet-range-collection>
  <e-bullet-range end='25' color='#ebebeb'></e-bullet-range>
  <e-bullet-range end='37' color='#d8d8d8'></e-bullet-range>
  <e-bullet-range end='45' color='#7f7f7f'></e-bullet-range>
</e-bullet-range-collection>
</ejs-bulletedchart>`
})
export class AppComponent {
  public minimum: number = 5;
  public maximum: number = 45;
  public interval: number = 5;
  public animation: AnimationModel = { enable: false };
  public dataLabel: Object = { enable: true };
  public data: Object[] = [
    {
      requiredStories: 20,
      completedStories: 25,
      name: 'David'
    },
    {
      requiredStories: 25,
      completedStories: 20,
      name: 'Asif'
    },
    {
      requiredStories: 15,
      completedStories: 10,
      name: 'Thomas'
    },
    {
      requiredStories: 40,
      completedStories: 39,
      name: 'Rohit'
    }
  ]
}
```



```

    },
    {
      requiredStories: 35,
      completedStories: 40,
      name: 'Virat'
    },
    {
      requiredStories: 28,
      completedStories: 25,
      name: 'Jude'
    },
    {
      requiredStories: 10,
      completedStories: 18,
      name: 'Warner'
    },
    {
      requiredStories: 30,
      completedStories: 28,
      name: 'Malik'
    }
  ];
}

```

MAIN.TS

```

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

```

Data Label Customization

Data Labels color, opacity, font size, font family, font weight, and font style can be customized using the [labelStyle](#).

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { BulletChartModule } from '@syncfusion/ej2-angular-charts';
import { Component } from '@angular/core';
import { AnimationModel } from '@syncfusion/ej2-charts';
@Component({
  imports: [
    BulletChartModule
  ],
  providers: [ ],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-bulletchart id='localData' valueField='completedStories'
targetField='requiredStories'
category='name' height='400px' [minimum]='minimum' [maximum]='maximum'
[interval]='interval'
title='Sprint Planning' titlePosition='Top' [dataSource]='data'
[animation]='animation' [dataLabel]='dataLabel'>

```

```

<e-bullet-range-collection>
  <e-bullet-range end='25' color='#ebeb'></e-bullet-range>
  <e-bullet-range end='37' color='#d8d8d8'></e-bullet-range>
  <e-bullet-range end='45' color='#7f7f7f'></e-bullet-range>
</e-bullet-range-collection>
</ejs-bulletedchart>`
}))
export class AppComponent {
  public minimum: number = 5;
  public maximum: number = 45;
  public interval: number = 5;
  public animation: AnimationModel = { enable: false };
  public dataLabel: Object = { enable: true, labelStyle: { color: 'yellow',
size: '20' } };
  public data: Object[] = [
    {
      requiredStories: 20,
      completedStories: 25,
      name: 'David'
    },
    {
      requiredStories: 25,
      completedStories: 20,
      name: 'Asif'
    },
    {
      requiredStories: 15,
      completedStories: 10,
      name: 'Thomas'
    },
    {
      requiredStories: 40,
      completedStories: 39,
      name: 'Rohit'
    },
    {
      requiredStories: 35,
      completedStories: 40,
      name: 'Virat'
    },
    {
      requiredStories: 28,
      completedStories: 25,
      name: 'Jude'
    },
    {
      requiredStories: 10,
      completedStories: 18,
      name: 'Warner'
    },
    {
      requiredStories: 30,
      completedStories: 28,
      name: 'Malik'
    }
  ];
}

```

MAIN.TS

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

Tool tip in Angular Bullet chart component

When the mouse is hovered over a bar in the Bullet Chart, the tooltip displays important summary about the actual and the target bar values.

Default Tooltip

The tooltip is not visible by default. To make it visible, set the [enable](#) property in the [tooltip](#) to **true** and injecting `BulletTooltip` module using `BulletChart.Inject(BulletTooltip)`.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { BulletChartModule } from '@syncfusion/ej2-angular-charts'
import { Component } from '@angular/core';
import { AnimationModel } from '@syncfusion/ej2-charts';
@Component({
  imports: [
    BulletChartModule
  ],
  providers: [ ],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-bulletchart valueField='value' targetField='target'
[minimum]='minimum' [maximum]='maximum'
[interval]='interval' [dataSource]='data' [animation]='animation'
title='Sales Rate' subtitle='(in dollars $)'
[tooltip]='tooltip'>
  <e-bullet-range-collection>
    <e-bullet-range end='35' color='#ebebeb'></e-bullet-range>
    <e-bullet-range end='70' color='#d8d8d8'></e-bullet-range>
    <e-bullet-range end='100' color='#7f7f7f'></e-bullet-range>
  </e-bullet-range-collection>
</ejs-bulletchart>`
})
export class AppComponent {
  public minimum: number = 0;
  public maximum: number = 100;
  public interval: number = 20;
  public data: Object[] = [
    { value: 55, target: 45 }
  ];
  public animation: AnimationModel = { enable: false };
  public tooltip: Object = { enable: true };
}
```

MAIN.TS

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

Tooltip Template

Any HTML elements can be displayed in the tooltip by using the [template](#) property of the [tooltip](#). You can use the **\${target}** and **\${value}** as place holders in the HTML element to display the value and target values from the data source of corresponding data point.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { BulletChartModule } from '@syncfusion/ej2-angular-charts'
import { BulletTooltipService } from '@syncfusion/ej2-angular-charts'
import { Component } from '@angular/core';
import { AnimationModel } from '@syncfusion/ej2-charts';
@Component({
  imports: [
    BulletChartModule
  ],
  providers: [ BulletTooltipService ],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-bulletchart valueField='value' targetField='target'
[minimum]='minimum' [maximum]='maximum'
[interval]='interval' [dataSource]='data' [animation]='animation'
title='Sales Rate' subtitle='(in dollars $)'
[tooltip]='tooltip' height='110px'>
  <e-bullet-range-collection>
    <e-bullet-range end='35' color='#ebebcb'></e-bullet-range>
    <e-bullet-range end='70' color='#d8d8d8'></e-bullet-range>
    <e-bullet-range end='100' color='#7f7f7f'></e-bullet-range>
  </e-bullet-range-collection>
</ejs-bulletchart>`
})
export class AppComponent {
  public minimum: number = 0;
  public maximum: number = 100;
  public interval: number = 20;
  public data: Object[] = [
    { value: 55, target: 45 }
  ];
  public animation: AnimationModel = { enable: false };
  public tooltip: Object = { enable: true, template : '#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 Customization

The following properties can be used to customize the Bullet Chart tooltip.

- [fill](#) - Specifies the color of tooltip.
- [border](#) - Specifies the tooltip border color and width.
- [textStyle](#) - Specifies the tooltip font family, font style, font weight, color and size.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { BulletChartModule } from '@syncfusion/ej2-angular-charts'
import { Component } from '@angular/core';
import { AnimationModel } from '@syncfusion/ej2-charts';
@Component({
  imports: [
    BulletChartModule
  ],
  providers: [ ],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-bulletedchart valueField='value' targetField='target'
[minimum]='minimum' [maximum]='maximum'
[interval]='interval' [dataSource]='data' [animation]='animation'
title='Sales Rate' subtitle='(in dollars $)'
[tooltip]='tooltip'>
  <e-bullet-range-collection>
    <e-bullet-range end='35' color='#ebeb'></e-bullet-range>
    <e-bullet-range end='70' color='#d8d8d8'></e-bullet-range>
    <e-bullet-range end='100' color='#7f7f7f'></e-bullet-range>
  </e-bullet-range-collection>
</ejs-bulletedchart>`
})
export class AppComponent {
  public minimum: number = 0;
  public maximum: number = 100;
  public interval: number = 20;
  public data: Object[] = [
    { value: 55, target: 45 }
  ];
  public animation: AnimationModel = { enable: false };
  public tooltip: Object = { enable: true, fill: 'red', border:{ color:
'green', width: 10 } };
}
```

MAIN.TS

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


Accessibility in Angular Bullet chart component

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

The accessibility compliance for the Bullet chart component is outlined below.

| Accessibility Criteria | Compatibility |


| -- | -- |

| [WCAG 2.2](#) Support |  |

| [Section 508](#) Support |  |

| Screen Reader Support |  |

| Right-To-Left Support |  |

| Color Contrast |  |

| Mobile Device Support |  |

| Keyboard Navigation Support |  |

| [Accessibility Checker](#) Validation |  |

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

<style>

.post .post-content img {

display: inline-block;

margin: 0.5em 0;

}

</style>

<div> - All features of the component meet the requirement.</div>

<div> - Some features of the component do not meet the requirement.</div>

<div> - The component does not meet the requirement.</div>

WAI-ARIA attributes

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

- `img` (role)
- `button` (role)
- `aria-label` (attribute)
- `aria-pressed` (attribute)

Keyboard interaction

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

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

| --- | --- |

| **Tab** | Moves the focus to the next element in the Bullet chart. |

| **Shift + Tab** | Moves the focus to the previous element in the Bullet chart. |

| **Ctrl + P** | Prints the Bullet chart. |

Ensuring accessibility

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

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

See also

- [Accessibility in Syncfusion Angular components](#)

ButtonGroup

Getting started with Angular Button group component

This section explains how to create a simple CSS ButtonGroup, and demonstrate the basic usage of the CSS ButtonGroup component in an Angular environment.

Dependencies

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

```
`js
```

```
|-- @syncfusion/ej2-splitbuttons
```

```
,
```

Setup Angular environment

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

`
npm install -g @angular/cli
`

Create an Angular application

Start a new Angular application using below Angular CLI command.

`
ng new my-app
cd my-app
`

Installing Syncfusion ButtonGroup package

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

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

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

Ivy library distribution package

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

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

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

Angular compatibility compiled package(ngcc)

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

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

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

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

`bash
@syncfusion/ej2-angular-buttons:"20.2.38-ngcc"
`

Note: If the ngcc tag is not specified while installing the package, the Ivy Library Package will be installed and this package will throw a warning.

Adding Button module

Import 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 button module from buttons package.
import { ButtonModule } from '@syncfusion/ej2-angular-buttons';
import { AppComponent } from './app.component';
@NgModule({
  imports: [ BrowserModule, ButtonModule ], // Registering EJ2 Button Module
  declarations: [ AppComponent ],
  bootstrap: [ AppComponent ]
})
export class AppModule { }
```

Adding Syncfusion ButtonGroup component

Modify the template in `app.component.ts` file to render the ButtonGroup Component.

```
`typescript
import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
  template: `<div class='e-btn-group'>
<button ej-button>HTML</button>
<button ej-button>CSS</button>
<button ej-button>Javascript</button>
</div>`
})
export class AppComponent { }
```

To render Button in CSS ButtonGroup component, import Button module into the angular application(app.module.ts) from the package `@syncfusion/ej2-angular-buttons` and its styles in `style.css`.

Adding CSS reference

Add ButtonGroup component's styles as given below in `style.css`.

```
`css
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css';
`
```

Running the application

Run the application in the browser using the following command:

```
`
ng serve
`
```

The following example shows a basic ButtonGroup component.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component } from '@angular/core';
@Component({
  imports: [
    ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<div class="e-section-control">
    <!-- To render ButtonGroup. -->
    <div class="e-btn-group">
      <button ej2-button>HTML</button>
      <button ej2-button>CSS</button>
      <button ej2-button>Javascript</button>
    </div>
  </div>`
})
export class AppComponent { }
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
```

```
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Orientation

ButtonGroup can be arranged both in a vertical and horizontal orientation. By default, it is horizontally oriented.

Vertical Orientation

ButtonGroup can be aligned vertically by using the built-in CSS `e-vertical` to the target element.

The following example illustrates how to achieve vertical orientation in ButtonGroup.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component } from '@angular/core';
@Component({
  imports: [
    ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<div class="e-section-control">
    <!-- To render ButtonGroup. -->
    <div class="e-btn-group e-vertical">
      <button ej-button>HTML</button>
      <button ej-button>CSS</button>
      <button ej-button>Javascript</button>
    </div>
  </div>`
})
export class AppComponent { }
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

ButtonGroup does not support SplitButton component nesting in a vertical orientation.

Types and styles in Angular Button group component

This section explains about different types and styles of ButtonGroup.

ButtonGroup types

Outline ButtonGroup

An Outline ButtonGroup has a border with transparent background. To create Outline ButtonGroup, `e-outline` class should be added to the target element and to each button elements using `cssClass` property.

The following sample illustrates how to achieve an Outline ButtonGroup,

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component } from '@angular/core';
@Component({
  imports: [

    ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<div class="e-section-control">
    <!-- To render ButtonGroup. -->
    <div class='e-btn-group e-outline'>
      <button ejs-button cssClass='e-outline'>HTML</button>
      <button ejs-button cssClass='e-outline'>CSS</button>
      <button ejs-button cssClass='e-outline'>Javascript</button>
    </div>
  </div>`
})
export class AppComponent { }
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

ButtonGroup does not have support for **flat** and **round** types.

ButtonGroup styles

The Essential JS 2 ButtonGroup has the following predefined styles. This can be achieved by adding corresponding class name in each button elements using **cssClass** property.

Class	Description
-----	-----
e-primary	Used to represent a primary action.
e-success	Used to represent a positive action.
e-info	Used to represent an informative action.
e-warning	Used to represent an action with caution.
e-danger	Used to represent a negative action.

The following example illustrates how to achieve predefined styles in ButtonGroup.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component } from '@angular/core';
@Component({
  imports: [
    ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<div class="e-section-control">
    <!-- To render ButtonGroup. -->
    <div class='e-btn-group'>
      <button ejs-button cssClass='e-info'>View</button>
      <button ejs-button>Edit</button>
      <button ejs-button cssClass='e-danger'>Delete</button>
    </div>
  </div>`
})
export class AppComponent { }
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Predefined ButtonGroup styles provide only the visual indication. So,

ButtonGroup content should define the ButtonGroup style for the users of assistive technologies such as screen readers.

See Also

- [ButtonGroup with icons](#)
- [Create ButtonGroup with rounded corner](#)

Selection in Angular Button group component

Selection

Single selection

ButtonGroup supports radio type selection in which only one button can be selected. This can be achieved by adding input element along with `id` attribute with its corresponding label along with `for` attribute inside the target element. In this ButtonGroup, the type of the input element should be `radio` and `e-btn` is added to the `label` element.

The following example illustrates the single selection behavior in ButtonGroup.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
```

```

import { BrowserModule } from '@angular/platform-browser'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component } from '@angular/core';
@Component({
  imports: [

    ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<div class="e-section-control">
    <!-- To render ButtonGroup. -->
    <div class="e-btn-group">
      <input type="radio" id="radioleft" name="font"
value="left"/>
      <label class="e-btn" for="radioleft">Left</label>
      <input type="radio" id="radiomiddle" name="font"
value="middle"/>
      <label class="e-btn" for="radiomiddle">Center</label>
      <input type="radio" id="radiatoright" name="font"
value="right"/>
      <label class="e-btn" for="radiatoright">Right</label>
    </div>
  </div>`
})
export class AppComponent { }

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Multiple selection

ButtonGroup supports checkbox type selection in which multiple button can be selected. This can be achieved by adding input element along with **id** attribute with its corresponding label along with **for** attribute inside the target element. In this ButtonGroup, the type of the input element should be **checkbox** and **e-btn** is added to the **label** element.

The following example illustrates the multiple selection behavior in ButtonGroup.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component } from '@angular/core';
@Component({
  imports: [

    ButtonModule
  ],

```

```

standalone: true,
  selector: 'app-root',
  template: `<div class="e-section-control">
    <!-- To render ButtonGroup. -->
    <div class="e-btn-group">
      <input type="checkbox" id="check_bold" name="align"
value="bold"/>
      <label class="e-btn" for="check_bold">Bold</label>
      <input type="checkbox" id="check_italic" name="align"
value="italic"/>
      <label class="e-btn" for="check_italic">Italic</label>
      <input type="checkbox" id="check_underline" name="align"
value="underline"/>
      <label class="e-btn"
for="check_underline">Underline</label>
    </div>
  </div>`
  })
export class AppComponent { }

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Nesting

Nesting with other components can be possible in ButtonGroup. The following components can be nested in ButtonGroup,

- DropDownButton
- SplitButton

For nesting support, [SplitButton dependencies](#) should be configured and added in `system.config.js`.

DropDownButton

To initialize DropDownButton component, refer [DropDownButton Getting Started documentation](#).

In the following example, the DropDownButton component is rendered in `app.component.ts` and `DropDownButtonModule` is imported in `app.module.ts` file.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { enableRipple } from '@syncfusion/ej2-base'
import { DropDownButtonModule } from '@syncfusion/ej2-angular-splitbuttons'
import { Component } from '@angular/core';
import { ItemModel } from '@syncfusion/ej2-angular-splitbuttons';
@Component({
  imports: [

```

```

        DropDownButtonModule,
        ButtonModule
    ],
    standalone: true,
    selector: 'app-root',
    template: `<div class="e-section-control">
        <!-- To render ButtonGroup. -->
        <div class='e-btn-group'>
            <button class='e-btn'>HTML</button>
            <button class='e-btn'>CSS</button>
            <button class='e-btn'>Javascript</button>
            <button ejs-dropdownbutton [items]='items'
content='More'></button>
        </div>
    </div>`
    ))
export class AppComponent {
    public items: ItemModel[] = [
        {
            text: 'Learn SQL'
        },
        {
            text: 'Learn PHP'
        },
        {
            text: 'Learn Bootstrap'
        }
    ];
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

SplitButton

To initialize SplitButton component refer [SplitButton Getting Started documentation](#).

In the following example, the SplitButton component is rendered in `app.component.ts` and `SplitButtonModule` is imported in `app.module.ts` file.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { enableRipple } from '@syncfusion/ej2-base'
import { SplitButtonModule } from '@syncfusion/ej2-angular-splitbuttons'
import { Component } from '@angular/core';
import { ItemModel } from '@syncfusion/ej2-angular-splitbuttons';
@Component({
    imports: [

        SplitButtonModule,

```



```

    ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<div class="e-section-control">
    <!-- To render ButtonGroup. -->
    <div class='e-btn-group'>
      <button class='e-btn'>Cut</button>
      <button class='e-btn'>Copy</button>
      <ejs-splitbutton content="Paste" [items]='items'></ejs-
splitbutton>
    </div>
  </div>`
})
export class AppComponent {
  public items: ItemModel[] = [
    {
      text: 'Paste'
    },
    {
      text: 'Paste Text'
    },
    {
      text: 'Paste Special'
    }
  ];
}

```

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 ButtonGroup selected state on initial render](#)

Accessibility in Angular Button group component

The Button group component followed the accessibility guidelines and standards, including [ADA](#), [Section 508](#), [WCAG 2.2](#) standards, and [WCAG roles](#) that are commonly used to evaluate accessibility.

The accessibility compliance for the Button group component is outlined below.

| Accessibility Criteria | Compatibility |

| -- | -- |

| [WCAG 2.2](#) Support | |

| [Section 508](#) Support | |

```

| Screen Reader Support |  |

| Right-To-Left Support |  |

| Color Contrast |  |

| Mobile Device Support |  |

| Keyboard Navigation Support |  |

| Accessibility Checker Validation |  |

| Axe-core Accessibility Validation |  |

<style>
.post .post-content img {
display: inline-block;
margin: 0.5em 0;
}
</style>

<div> - All
features of the component meet the requirement.</div>

<div> - Some features of the component do not meet the requirement.</div>

<div> - The component does not meet the requirement.</div>

```

Keyboard interaction

The Button group component followed the [keyboard interaction](#) guideline, making it easy for people who use assistive technologies (AT) and those who completely rely on keyboard navigation. The following keyboard shortcuts are supported by the Button group component.

Normal behavior

| **Press** | **To do this** |

| --- | --- |

| **Tab** | **Focuses the next button in the ButtonGroup.** |

| **Enter/Space** | **Activates the focussed button in the ButtonGroup.** |

Checkbox behavior

| **Press** | **To do this** |

| --- | --- |

| Tab | Focuses the next button in the ButtonGroup. |

| Space | Activates the focussed button in the ButtonGroup. |

Radiobutton behavior

| Press | To do this |

| --- | --- |

| Tab | Focuses the active button in the ButtonGroup. |

| Right | Activates next/previous button in the ButtonGroup. |

Ensuring accessibility

The Button group component's accessibility levels are ensured through an [accessibility-checker](#) and [axe-core](#) software tools during automated testing.

The accessibility compliance of the Button group component is shown in the following sample. Open the [sample](#) in a new window to evaluate the accessibility of the Button group component with accessibility tools.

See also

- [Accessibility in Syncfusion Angular components](#)

How To

Create buttongroup with icons in Angular Button group component

To create ButtonGroup with icons, `iconCss` property of the button component can be used.

The following example illustrates how to create ButtonGroup with icons.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component } from '@angular/core';
@Component({
  imports: [
    ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<div class="e-section-control">
    <!-- To render ButtonGroup. -->
    <div class='e-btn-group'>
      <button ejs-button iconCss='e-icons e-left-
icon'>Left</button>
      <button ejs-button iconCss='e-icons e-middle-
icon'>Right</button>
      <button ejs-button iconCss='e-icons e-right-
icon'>Middle</button>
    </div>
  </div>`
})
```

```

  })
  export class AppComponent { }

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Create buttongroup with rounded corner in Button group component

The ButtonGroup with rounded corner has round edges on both side. In the ButtonGroup with rounded corner, **e-round-corner** class is to be added to the target element.

The following example illustrates how to create ButtonGroup with rounded corner.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component } from '@angular/core';
@Component({
  imports: [

    ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<div class="e-section-control">
    <!-- To render ButtonGroup. -->
    <div class='e-btn-group e-round-corner'>
      <button ej2-button>HTML</button>
      <button ej2-button>CSS</button>
      <button ej2-button>Javascript</button>
    </div>
  </div>`
})
export class AppComponent { }

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Disable in Angular Button group component

Particular button

To disable a particular button in a ButtonGroup, **disabled** attribute should be added to corresponding button element.

Whole ButtonGroup

To disable whole ButtonGroup, `disabled` attribute should be added to all the button elements.

The following example illustrates how to disable the particular and the whole ButtonGroup.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component } from '@angular/core';
@Component({
  imports: [

    ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<div class="e-section-control">
    <!-- To render ButtonGroup. -->
    <div class="e-btn-group">
      <button ej2-button>HTML</button>
      <button ej2-button [disabled]="true">CSS</button>
      <button ej2-button>Javascript</button>
    </div>
    <div class="e-btn-group">
      <button ej2-button [disabled]="true">HTML</button>
      <button ej2-button [disabled]="true">CSS</button>
      <button ej2-button [disabled]="true">Javascript</button>
    </div>
  </div>`
})
export class AppComponent { }

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

To disable radio/checkbox type ButtonGroup, the `disabled` attribute should be added to the particular input element.

Enable ripple in Angular Button group component

Ripple can be enabled by importing `enableRipple` method from `ej2-base` and set `enableRipple` as `true`.

The following example illustrates how to enable ripple for ButtonGroup.

<!-- markdownlint-disable MD033 -->

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'

```

```
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component } from '@angular/core';
@Component({
  imports: [

    ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<div class="e-section-control">
    <!-- To render ButtonGroup. -->
    <div class="e-btn-group">
      <button ejs-button>HTML</button>
      <button ejs-button>CSS</button>
      <button ejs-button>Javascript</button>
    </div>
  </div>`
})
export class AppComponent { }
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Enable rtl in Angular Button group component

ButtonGroup supports RTL functionality. This can be achieved by adding **e-rtl** class to the target element.

The following example illustrates how to create ButtonGroup with RTL support.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component } from '@angular/core';
@Component({
  imports: [

    ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<div class="e-section-control">
    <!-- To render ButtonGroup with RTL. -->
    <div class="e-btn-group e-rtl">
      <button ejs-button>HTML</button>
      <button ejs-button>CSS</button>
      <button ejs-button>Javascript</button>
    </div>
  </div>`
})
```

```

  })
  export class AppComponent { }

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Form submit in Angular Button group component

The name attribute of the input element is used to group radio/checkbox type ButtonGroup. When the radio/checkbox type are grouped in form, the checked items value attribute will be posted to server on form submit that can be retrieved through the name. The disabled radio/checkbox type value will not be sent to the server on form submit.

In the following code snippet, the radio type ButtonGroup is explained with male value as checked.

Now, the value that is in checked state will be sent on form submit.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component } from '@angular/core';
@Component({
  imports: [

    ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<div class="e-section-control">
    <!-- To render ButtonGroup. -->
    <form>
      <div class="e-btn-group">
        <input type="radio" id="male" name="gender"
value="male" checked/>
        <label class="e-btn" for="male">Male</label>
        <input type="radio" id="female" name="gender"
value="female"/>
        <label class="e-btn" for="female">Female</label>
        <input type="radio" id="transgender" name="gender"
value="transgender"/>
        <label class="e-btn"
for="transgender">Transgender</label>
      </div>
      <button class="e-btn e-primary">Submit</button>
    </form>
  </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));
```

Initialize buttongroup using util function in Button group component

Though, it is a CSS component for easy initialization of ButtonGroup `createButtonGroup` util function can be used.

To use `createButtonGroup` util function, [SplitButton dependencies](#) should be configured and it should be added in `system.config.js`.

Using `createButtonGroup` method, the Button options, selector, and cssClass is passed and the corresponding classes is added to the elements.

Create basic ButtonGroup

To create a basic ButtonGroup, the target element along with the button elements should be created and `createButtonGroup` should be imported from `ej2-splitbuttons`.

Create radio type ButtonGroup

To create a radio type ButtonGroup, the target element along with the input elements should be created with type radio.

Create checkbox type ButtonGroup

Checkbox type ButtonGroup creation is similar to radio type ButtonGroup, instead the type of the input elements should be checkbox.

The following example illustrates how to create ButtonGroup using `createButtonGroup` function for basic, checkbox, and radio

type behaviors.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component } from '@angular/core';
import { createButtonGroup } from '@syncfusion/ej2-splitbuttons';
@Component({
  imports: [
    ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<div class="e-section-control">
    <!-- To render ButtonGroup. -->
    <h5>Normal behavior</h5>
    <div id='basic'>
      <button></button>
```



```

        <button></button>
        <button></button>
    </div>
    <h5>Checkbox type behavior</h5>
    <div id='checkbox'>
        <input type="checkbox" id="checkbold" name="font"
value="bold" />
        <input type="checkbox" id="checkitalic" name="font"
value="italic" />
        <input type="checkbox" id="checkunderline" name="font"
value="underline" />
    </div>
    <h5>Radiobutton type behavior</h5>
    <div id='radio'>
        <input type="radio" id="radioleft" name="align"
value="left" />
        <input type="radio" id="radiomiddle" name="align"
value="middle" />
        <input type="radio" id="radiatoright" name="align"
value="right" />
    </div>`
    })
    export class AppComponent {
    ngOnInit() {
    createButtonGroup('#basic', {
        buttons: [
            { content: 'HTML' },
            { content: 'CSS' },
            { content: 'Javascript' }
        ]
    });
    createButtonGroup('#checkbox', {
        buttons: [
            { content: 'Bold' },
            { content: 'Italic' },
            { content: 'Undeline' }
        ]
    });
    createButtonGroup('#radio', {
        buttons: [
            { content: 'Left' },
            { content: 'Center' },
            { content: 'Right' }
        ]
    });
    }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

If null value is passed in button options, then that particular button will be skipped from processing in `createButtonGroup` util function.

Show `buttongroup` selected state on initial render in Button group component

To show selected state on initial render, `checked` property should be added to the corresponding input element.

The following example illustrates how to show selected state on initial render.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component } from '@angular/core';
@Component({
  imports: [

    ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<div class="e-section-control">
    <!-- To render ButtonGroup. -->
    <div class="e-btn-group">
      <input type="checkbox" id="checkbold" name="font"
value="bold" checked/>
      <label class="e-btn" for="checkbold">Bold</label>
      <input type="checkbox" id="checkitalic" name="font"
value="italic" />
      <label class="e-btn" for="checkitalic">Italic</label>
      <input type="checkbox" id="checkline" name="font"
value="underline"/>
      <label class="e-btn" for="checkline">Underline</label>
    </div>
  </div>`
})
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));
```

Button

Getting started with Angular Button component

This section explains how to create a simple Button and demonstrate the basic usage of the Button module in an Angular environment.

Dependencies

The list of dependencies required to use the Button module in your application is given below:

```
`typescript
|-- @syncfusion/ej2-angular-buttons
|-- @syncfusion/ej2-angular-base
|-- @syncfusion/ej2-base
|-- @syncfusion/ej2-buttons
\
```

Setup Angular environment

You can use [Angular CLI](#) to setup your Angular applications. To install Angular CLI use the following command.

```
\
npm install -g @angular/cli
\
```

Create an Angular application

Start a new Angular application using below Angular CLI command.

```
\
ng new my-app
cd my-app
\
```

Installing Syncfusion Button package

Syncfusion packages are distributed in npm as `@syncfusion` scoped packages. You can get all the Angular Syncfusion package from npm [link](#).

Currently, Syncfusion provides two types of package structures for Angular components,

1. Ivy library distribution package [format](#)
2. Angular compatibility compiler(Angular's legacy compilation and rendering pipeline) package.

Ivy library distribution package

Syncfusion Angular packages(`>=20.2.36`) has been moved to the Ivy distribution to support the Angular [Ivy](#) rendering engine and the package are compatible with Angular version 12 and above. To download the package use the below command.

Add [@syncfusion/ej2-angular-buttons](#) package to the application.

```
`bash
npm install @syncfusion/ej2-angular-buttons --save
\
```

Angular compatibility compiled package(ngcc)

For Angular version below 12, you can use the legacy (ngcc) package of the Syncfusion Angular components. To download the ngcc package use the below.

Add [@syncfusion/ej2-angular-buttons@ngcc](#) package to the application.

```
`bash
npm install @syncfusion/ej2-angular-buttons@ngcc --save
`
```

To mention the ngcc package in the `package.json` file, add the suffix `-ngcc` with the package version as below.

```
`bash
@syncfusion/ej2-angular-buttons:"20.2.38-ngcc"
`
```

Note: If the ngcc tag is not specified while installing the package, the Ivy Library Package will be installed and this package will throw a warning.

Adding Button module

Import 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 button module from buttons package.
import { ButtonModule } from '@syncfusion/ej2-angular-buttons';
import { AppComponent } from './app.component';
@NgModule({
  imports: [ BrowserModule, ButtonModule ], // Registering EJ2 Button Module
  declarations: [ AppComponent ],
  bootstrap: [ AppComponent ]
})
export class AppModule { }
`
```

Adding Syncfusion Button component

Modify the template in `app.component.ts` file to render the Button module.

```
`typescript
import { Component } from '@angular/core';
@Component({
```

```

selector: 'app-root',
template: `<!-- To render Button. -->
<button ej2-button>Button</button>`
})
export class AppComponent { }
`

```

Adding CSS reference

Add 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 following example shows a basic Button component.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component } from '@angular/core';
@Component({
  imports: [
    ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<div class="e-section-control">
    <!-- To render Button. -->
    <button ej2-button>Button</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));

```

Change Button type

To change the default Button to flat Button, set the [cssClass](#) property with `e-flat` and text content of the Button is set using [content](#) property.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component } from '@angular/core';
@Component({
  imports: [

    ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<div class="e-section-control">
    <!-- To change the Button type. -->
    <button ej2-button cssClass="e-flat"
content="Button"></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));
```

Types and styles in Angular Button component

This section explains the different styles and types of Buttons.

Button styles

The Essential JS 2 Button has the following predefined styles that can be defined using the [cssClass](#) property.

Class	Description
-----	-----
e-primary	Used to represent a primary action.
e-success	Used to represent a positive action.
e-info	Used to represent an informative action.
e-warning	Used to represent an action with caution.
e-danger	Used to represent a negative action.
e-link	Changes the appearance of the Button like a hyperlink.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component } from '@angular/core';
@Component({
  imports: [

    ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  styleUrls: ['./style.css'],
  template: `<div class="e-section-control">
    <!-- Primary Button - Used to represent a primary action. -->
    <button ej2-button cssClass="e-primary">Primary</button>
    <!-- Success Button - Used to represent a positive action. --
  >

    <button ej2-button cssClass="e-success">Success</button>
    <!-- Info Button - Used to represent an informative action --
  >

    <button ej2-button cssClass="e-info">Info</button>
    <!-- Warning Button - Used to represent an action with
  caution. -->

    <button ej2-button cssClass="e-warning">Warning</button>
    <!-- Danger Button - Used to represent a negative action. -->
    <button ej2-button cssClass="e-danger">Danger</button>
    <!-- Link Button - Changes the appearance of the Button like
  a hyperlink. -->

    <button ej2-button cssClass="e-link">Link</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));
```

Predefined Button styles provide only the visual indication. So, Button content should define the Button style for the users of assistive technologies such as screen readers.

Primary action button can also be achieved by setting [isPrimary](#) property as `true`.

Button types

The types of Essential JS 2 Button are as follows:

- Basic types
- Flat Button
- Outline Button

- Round Button
- Toggle Button

Basic types

The basic Button types are explained below.

Type	Description
Button	Defines a click Button.
Submit	This Button submits the form data to the server.
Reset	This Button resets all the controls in the form elements to their initial values.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component } from '@angular/core';
@Component({
  imports: [
    ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  styleUrls: ['./style.css'],
  template: `<div class="e-section-control">
    <form>
      <!-- Submit type button -->
      <button type="submit" ejs-button>Submit</button>
      <!-- Reset type button -->
      <button type="reset" ejs-button>Reset</button>
    </form>
  </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));
```

Flat Button

The Flat Button is styled with no background color. To create a flat Button, set the [cssClass](#) property to `e-flat`.

Outline Button

An outline Button has a border with transparent background. To create an outline Button, set the [cssClass](#) property to `e-outline`.

Round Button

A round Button is shaped like a circle. Usually, it contains an icon representing its action.

To create a round Button, set the [cssClass](#) property to `e-round`.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component } from '@angular/core';
@Component({
  imports: [
    ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  styleUrls: ['./style.css'],
  template: `<div class="e-section-control">
    <!-- Flat Button. -->
    <button ej2-button cssClass="e-flat">Flat</button>
    <!-- Outline Button. -->
    <button ej2-button cssClass="e-outline">Outline</button>
    <!-- Round Button - Icon can be loaded by setting "e-icons e-
plus-icon" in "iconCss" property.-->
    <!-- Use "e-icons" class name to load Essential JS 2 built-in
icons.-->
    <!-- Use "e-plus-icon" class name to load plus icon that you
can refer in "styles.css" -->
    <button ej2-button cssClass="e-round" iconCss="e-icons e-
plus-icon" [isPrimary]="true"></button>
  </div>`
})
export class AppComponent { }
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Toggle Button

A toggle Button allows you to change between the two states. The Button is active in toggled state and can be recognized through the `e-active` class. The functionality of the toggle Button is handled by click event. To create a toggle Button, set the [isToggle](#)

property to `true`. In the following code snippet, the toggle Button text changes to play/pause based on the state of the Button with the use of click event.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component, ViewChild, HostListener } from '@angular/core';
import { ButtonComponent } from '@syncfusion/ej2-angular-buttons';
@Component({
  imports: [
    ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  styleUrls: ['./style.css'],
  template: `<div class="e-section-control">
    <!-- Button is active in toggled state. -->
    <button #togglebtn ej2-button cssClass="e-flat"
iconCss="e-btn-sb-icon e-play-icon" [isToggle]="true"
content="Play"></button>
  </div>`
})
// Click event handled using HostListener.
export class AppComponent {
  @ViewChild('togglebtn') togglebtn: ButtonComponent | any;
  @HostListener('click', ['togglebtn'])
  btnClick() {
    if(this.togglebtn.element.classList.contains('e-active')){
      this.togglebtn.content = 'Pause';
      this.togglebtn.iconCss = 'e-btn-sb-icon e-pause-icon';
    }
    else {
      this.togglebtn.content = 'Play';
      this.togglebtn.iconCss = 'e-btn-sb-icon e-play-icon';
    }
  }
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Icons

Button with font icons

The Button can have an icon to provide the visual representation of the action. To place the icon on a Button, set the `iconCss` property with the required icon CSS. By default, the icon is positioned to the left side of the Button. You can customize the icon's position

by using the [iconPosition](#) property.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component } from '@angular/core';
@Component({
  imports: [

    ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  styleUrls: ['./style.css'],
  template: `<div class="e-section-control">
      <!-- To position the icon to the left of the text on a
Button. -->
      <button ejs-button iconCss="e-btn-sb-icon e-prev-
icon">Previous</button>
      <!-- To position the icon to the right of the text on a
Button. -->
      <button ejs-button iconCss= "e-btn-sb-icon e-stop-icon"
iconPosition="Right">Stop</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));
```

Button with SVG image

SVG image can be added to the Button using [iconCss](#) property.

In the following example, SVG image is added using the iconCss class `e-search-icon` by setting `height` and `width`.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component } from '@angular/core';
@Component({
  imports: [

    ButtonModule
  ],
  standalone: true,
```

```

    selector: 'app-root',
    styleUrls: ['styles.css'],
    template: `<div class="e-section-control">
        <button ej-button iconCss= "e-search-icon"></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));

```

The Essential JS 2 provides a set of icons that can be loaded by applying **e-icons** class name to the element. You can also use third party icons on the Button using the [iconCss](#) property.

Button size

The two types of Button sizes are default and small. To change the size of the default Button to small Button, set the [cssClass](#) property to **e-small**.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component } from '@angular/core';
@Component({
  imports: [
    BrowserModule,
    ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  styleUrls: ['./style.css'],
  template: `<div class="e-section-control">
    <!-- Small Button. -->
    <button ej-button cssClass="e-small">Small</button>
    <!-- Normal Button. -->
    <button ej-button>Normal</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));

```

See Also

- [Customize Button appearance](#)
- [How to create block button](#)
- [How to create repeat button](#)

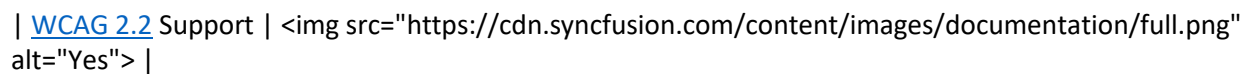
Accessibility in Angular Button component

The Button component followed the accessibility guidelines and standards, including [ADA, Section 508](#), [WCAG 2.2](#) standards, and [WCAG roles](#) that are commonly used to evaluate accessibility.

The accessibility compliance for the Button component is outlined below.

| Accessibility Criteria | Compatibility |

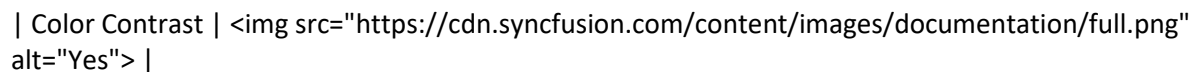
| -- | -- |

| [WCAG 2.2](#) Support |  |

| [Section 508](#) Support |  |

| Screen Reader Support |  |

| Right-To-Left Support |  |

| Color Contrast |  |

| Mobile Device Support |  |

| Keyboard Navigation Support |  |

| [Accessibility Checker](#) Validation |  |

| [Axe-core](#) Accessibility Validation |  |

<style>

.post .post-content img {

display: inline-block;

margin: 0.5em 0;

}

</style>

<div> - All features of the component meet the requirement.</div>

```
<div> - Some features of the component do not meet the requirement.</div>
```

```
<div> - The component does not meet the requirement.</div>
```

WAI-ARIA attributes

The Button component followed the [WAI-ARIA](#) patterns to meet the accessibility. The following ARIA attributes are used in the Button component:

| Attributes | Purpose |

| --- | --- |

| `aria-label` | Provides an accessible name for the icon only button. |

Keyboard interaction

The Button component followed the [keyboard interaction](#) guideline, making it easy for people who use assistive technologies (AT) and those who completely rely on keyboard navigation. The following keyboard shortcuts are supported by the Button component.

| Press | To do this |

| --- | --- |

| Space | When the button has focus, pressing the space key changes the state of the button. |

Ensuring accessibility

The Button component's accessibility levels are ensured through an [accessibility-checker](#) and [axe-core](#) software tools during automated testing.

The accessibility compliance of the Button component is shown in the following sample. Open the [sample](#) in a new window to evaluate the accessibility of the Button component with accessibility tools.

See also

- [Accessibility in Syncfusion Angular components](#)

How To

Add link to a button in Angular Button component

Link can be added to the Button by adding `e-link` using `cssClass` property and `<a>` tag with `href` attribute should be added inside the button element.

The following example illustrates how to add link to a Button.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component } from '@angular/core';
@Component({
  imports: [
    ButtonModule
```

```

    ],
    standalone: true,
    selector: 'app-root',
    template: `<div class="e-section-control">
        <!-- To render Button. -->
        <button ejs-button cssClass='e-link'><a
href="https://www.google.com/" target='_blank'>Go to google</a></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));

```

Create a block button in Angular Button component

You can customize a button into a block button that will span the entire width of its parent element.

To create a block button, set the [cssClass](#) property as `e-block`.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component } from '@angular/core';
@Component({
  imports: [

    ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  styleUrls:['./style.css'],
  template: `
    <div class="e-section-control">
      <!-- Block button. -->
      <button ejs-button cssClass="e-block">Block
Button</button>

      <!-- Primary block button. -->
      <button ejs-button cssClass="e-block e-primary">Block
Button</button>

      <!-- Success block button. -->
      <button ejs-button cssClass="e-block e-success">Block
Button</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));
```

Customize button appearance in Angular Button component

You can customize the appearance of the button by using the Cascading Style Sheets (CSS). Define the CSS according to your requirement, and assign the class name to the `cssClass` property.

In the following code snippet the background color, text color, height, width, and sharp corner of the button can be customized through the `e-custom` class for all states (hover, focus, and active).

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component } from '@angular/core';
@Component({
  imports: [
    BrowserModule,
    ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  styleUrls: ['./style.css'],
  template: `<div class="e-section-control">
    <!-- To customize Button appearance. -->
    <!-- Refer the "e-custom" class details in "styles.css"-->
    <button ej2-button cssClass="e-custom">Custom</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));
```

Customize input and anchor elements in Angular Button component

You can customize the appearance of the input and anchor elements using predefined styles through the class property. In the following code snippet, the input element is customized as a link button by setting the `e-btn e-link` class, and the anchor element is customized as a primary button by setting the `e-btn e-primary` class.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component } from '@angular/core';
```



```

@Component({
  imports: [

    ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<div class="e-section-control">
    <div>
      <input type="button" id="inputbtn" value="Input Button"
class="e-btn e-link">
    </div><br>
    <div>
      <a id="anchorbtn" class="e-btn e-primary"
href="#">Google</a>
    </div>
  </div>`
})
export class AppComponent { }

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Repeat button in Angular Button component

The repeat button is a type of button in that the click event is triggered at regular time interval from the pressed state till the released state.

The following example explains about how to achieve repeat button in mouse and touch events.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component, ViewChild } from '@angular/core';
import { ButtonComponent } from '@syncfusion/ej2-angular-buttons';
@Component({
  imports: [

    ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<div class="e-section-control">
    <div id='container'>
      <div class='btncontainer'>
        <button #btn ej2-button (mousedown)="mousedown()"
(mouseup)="mouseup()" (touchstart)="touchstart()" (touchend)="touchend()"
(click)="onclick()">Button</button>
      </div>
      <div class='event' style="height:auto;">

```

```

        <table title='Event Trace' style="width:100%">
            <tbody>
                <tr>
                    <th>Event Trace</th>
                </tr>
                <tr>
                    <td>
                        <div class="eventarea" style="height:
250px;overflow: auto">
                            <span id="eventlog" style="word-
break: normal;"></span>
                        </div>
                    </td>
                </tr>
                <tr>
                    <td>
                        <div class="evtbtn"
style="padding:20px 0 0 20px">
                            <button #clear
(click)="clearEvt()" ejc-button>Clear</button>
                        </div>
                    </td>
                </tr>
            </tbody>
        </table>
    </div>
</div>`
})
export class AppComponent {
    @ViewChild('btn')
    public btn: ButtonComponent | any;
    @ViewChild('clear')
    public clear: ButtonComponent | any;
    public timeout: any;
    public spanElem: HTMLElement | any;
    public log: HTMLElement | any;
    mousedown() {
        this.timeout = setInterval(() => {
            this.spanElem = document.createElement('span');
            this.spanElem.innerHTML = 'Button click event triggered.<hr>';
            this.log = document.getElementById('eventlog');
            this.log.insertBefore(this.spanElem, this.log.firstChild);
        }, 200);
    };
    mouseup() {
        clearInterval(this.timeout);
    };
    touchstart() {
        this.timeout = setInterval(() => {
            this.spanElem = document.createElement('span');
            this.spanElem.innerHTML = 'Button click event triggered.<hr>';
            this.log = document.getElementById('eventlog');
            this.log.insertBefore(this.spanElem, this.log.firstChild);
        }, 200);
    };
    touchend() {

```

```

clearInterval(this.timeout);
};
onclick() {
  this.spanElem = document.createElement('span');
  this.spanElem.innerHTML = 'Button click event triggered.<hr>';
  this.log = document.getElementById('eventlog');
  this.log.insertBefore(this.spanElem, this.log.firstChild);
};
clearEvt() {
  document.getElementById('eventlog')!.innerHTML = '';
};
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Right to left in Angular Button component

Button component has RTL support. This can be achieved by setting [enableRtl](#) as `true`.

The following example illustrates how to enable right-to-left support in Button component.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component } from '@angular/core';
@Component({
  imports: [

    ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  styleUrls: ['./style.css'],
  template: `<div class="e-section-control">
    <button ej2-button iconCss="e-btn-icons e-setting-icon"
    [enableRtl]="true">Settings</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));

```

Set the disabled state in Angular Button component

Button component can be enabled/disabled by giving [disabled](#) property. To disable Button component, the `disabled` property can be set as `true`.

The following example demonstrates button in `disabled` state.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component } from '@angular/core';
@Component({
  imports: [

    ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<div class="e-section-control">
    <button ej2-button [disabled]="true">Disabled</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));
```

Tooltip for button in Angular Button component

Tooltip can be shown on button hover and it can be achieved by setting `title` attribute.

The following snippets illustrates how to show tooltip on button hover.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component, ViewChild, OnInit } from '@angular/core';
import { ButtonComponent } from '@syncfusion/ej2-angular-buttons';
@Component({
  imports: [

    ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  styleUrls:['./style.css'],
  template: `<div class="e-section-control">
```

```

        <button #btn ej-button
[isPrimary]="true">Button</button></div>`
    })
    export class AppComponent implements OnInit {
        @ViewChild('btn')
        private btn: ButtonComponent | any;
        ngOnInit() {
            this.btn.element.setAttribute("title", "Primary Button")
        }
    }
}

```

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 Button component

This article describes the API migration process of Button component from Essential JS 1 to Essential JS 2.

Properties

| Behavior | API in Essential JS 1 | API in Essential JS 2 |

| --- | --- | --- |

| Specifies the text of the button | **Property:** *text*

 <button ej-button id="btn" text="Button"></button> | **Property:** *content*

 <button ej-button id="btn" content="Button"></button> |

| Specifies the content type of the button | **Property:** *ContentType*

 <button ej-button id="btn" text="Save" contentType="TextAndImage" prefixIcon="e-icon e-save"></button> | Not applicable |

| Specifies icon for the button | **Property:** *prefixIcon*

 <button ej-button id="btn" contentType="ImageOnly" prefixIcon="e-icon e-save"></button> | **Property:** *iconCss*

 <button ej-button id="btn" iconCss="e-icons e-save"></button> |

| Positioning icon in the button | **Property:** *imagePosition*

 <button ej-button id="btn" contentType="TextAndImage" text="Save" prefixIcon="e-icon e-save" imagePosition="ImageRight"></button> | **Property:** *iconPosition*

 <button ej-button id="btn" content="Save" iconCss="e-icons e-save" iconPosition="Right"></button> |

| Specifies secondary icon for the button | **Property:** *suffixIcon*

 <button ej-button id="btn" text="FileSearch" contentType="ImageTextImage" prefixIcon="e-icon e-search" suffixIcon="e-icon e-file-html"></button> | Not applicable |

| Adding custom class | **Property:** *cssClass*

 <button ej-button id="btn" text="Button" cssClass="custom-class"></button> | **Property:** *cssClass*

 <button ej-button id="btn" cssClass="custom-class" content="Button"></button> |

| Specifies the size of the button | **Property:** *size*

 <button ej-button id="btn" text="Button" size="small"></button> | **Property:** *cssClass*

 <button ej-button id="btn" cssClass="e-small" content="Button"></button> |

| Triggers click event repeatedly while pressing the button | **Property:** *repeatButton*

 <button ej-button id="btn" text="Click" [repeatButton]="true"></button> | Not applicable |

| Sets time interval between two consecutive click event on the repeat button | **Property:** *timeInterval*

 <button ej-button id="btn" text="Click" [repeatButton]="true" timeInterval="100"></button> | Not applicable |

| Specifies the type of the button | **Property:** *type*

 <button ej-button id="btn" text="Button" type="Button"></button> | Not applicable |

| Changes normal button into rounded corner | **Property:** *showRoundedCorner*

 <button ej-button id="btn" text="Button" [showRoundedCorner]="true"></button> | Not applicable |

| Specifies the width of the button | **Property:** *width*

 <button ej-button id="btn" text="Button" width="150px"></button> | Not applicable |

| Specifies the height of the button | **Property:** *height*

 <button ej-button id="btn" text="Button" height="50px"></button> | Not applicable |

| Adds HTML attributes to the button | **Property:** *htmlAttributes*

 <button ej-button id="btn" text="Button" [htmlAttributes]="attributes"></button> | Not applicable |

| Allows the appearance of the Button to be enhanced and visually appealing | Not applicable | **Property:** *isPrimary*

 <button ej-button id="btn" isPrimary="true" content="Button"></button> |

| Makes the button toggle from normal to active state | **Property:** *isToggle*

 Not applicable | **Property:** *isToggle*

 <button ej-button id="btn" isToggle="true" content="Play"></button> |

| Specifies the disabled state of the button | **Property:** *enabled*

 <button ej-button id="btn" text="Button" [enabled]="false"></button> | **Property:** *disabled*

 <button ej-button id="btn" [disabled]="true" content="Button"></button> |

| Enable or disable rendering component in right to left direction | **Property:** *enableRTL*

 <button ej-button id="btn" contentType="TextAndImage" text="Save" prefixIcon="e-icon e-save" [enableRTL]="true"></button> | **Property:** *enableRtl*

 <button ej-button id="btn" [enableRtl]="true" content="Save" iconCss="e-icons e-save"></button> |

Methods

| Behavior | API in Essential JS 1 | API in Essential JS 2 |

| --- | --- | --- |

| Destroys the control | **Methods:** *destroy*

 <button ej-button #btn id="btn" text="Button"></button>
 @ViewChild('btn')
 public btnObj: ButtonComponent;
 this.btnObj.destroy(); | **Methods:** *destroy*

 <button ej-button #btn id="btn" content="Button"></button>
 @ViewChild('btn')
 public btnObj: ButtonComponent;
 this.btnObj.destroy(); |

| Disables the button | **Methods:** *disable*

 <button ej-button #btn id="btn" text="Button"></button>
 @ViewChild('btn')
 public btnObj: ButtonComponent;
 this.btnObj.disable(); | Not applicable |

| Enables the button | **Methods:** *enable*

 <button ej-button #btn id="btn" text="Button"></button>
 @ViewChild('btn')
 public btnObj: ButtonComponent;
 this.btnObj.enable(); | Not applicable |

Events

| Behavior | API in Essential JS 1 | API in Essential JS 2 |

| --- | --- | --- |

| Triggers while clicking the button | **Events:** *click*

 <button ej-button id="btn" text="Button" (click)="btnClick(\$event)"></button>
 btnClick(args) {
/ code block */
} | Not applicable |

| Triggers once the component rendering is completed | **Events:** *create*

 <button ej-button id="btn" text="Button" (click)="create(\$event)"></button>
 create(args) {
/ code block /
} | **Events:** *created*

 <button ej-button id="btn" content="Button" (created)="created()"></button>
 created() {
/ code block /
} |

| Triggers once the component is destroyed | **Events:** *destroy*

 <button ej-button id="btn" text="Button" (destroy)="destroy(\$event)"></button>
 destroy(args) {
/ code block */
} | Not applicable |