

USER GUIDE

Essential Studio for Blazor

Version - v19.4.0.38 | Release Date - December 17, 2021

Welcome to Syncfusion Blazor Components	48
How to best read this user guide	48
Components List	48
table	48
Getting Help	50
See Also	50
System requirements for Blazor Components.....	50
Integrated Development Environment.....	50
Framework & SDK	50
Browser Compatibility in Blazor.....	51
Blazor WebAssembly	51
Blazor Server Side	51
See Also	51
NuGet Packages for Syncfusion Blazor UI components.....	52
Benefits of using individual NuGet packages.....	52
Available NuGet packages.....	52
Syncfusion.Blazor.Core	52
Syncfusion.Blazor.BarcodeGenerator	52
Syncfusion.Blazor.BulletChart.....	53
Syncfusion.Blazor.Buttons	53
Syncfusion.Blazor.Calendars	53
Syncfusion.Blazor.Cards	54
Syncfusion.Blazor.Charts.....	54
Syncfusion.Blazor.CircularGauge	54
Syncfusion.Blazor.Data	54
Syncfusion.Blazor.DataVizCommon	55
Syncfusion.Blazor.Diagrams	55
Syncfusion.Blazor.DropDowns	55
Syncfusion.Blazor.FileManager	56
Syncfusion.Blazor.Gantt.....	56
Syncfusion.Blazor.Grid	57
Syncfusion.Blazor.HeatMap	57
Syncfusion.Blazor.InPlaceEditor.....	57
Syncfusion.Blazor.Inputs	58
Syncfusion.Blazor.Kanban	58

Syncfusion.Blazor.Layouts.....	58
Syncfusion.Blazor.LinearGauge.....	59
Syncfusion.Blazor.Lists	59
Syncfusion.Blazor.Maps	59
Syncfusion.Blazor.Navigations	59
Syncfusion.Blazor.Notifications	60
Syncfusion.Blazor.PdfViewer	60
Syncfusion.Blazor.PivotTable	61
Syncfusion.Blazor.Popups	62
Syncfusion.Blazor.ProgressBar.....	62
Syncfusion.Blazor.QueryBuilder	62
Syncfusion.Blazor.RangeNavigator	63
Syncfusion.Blazor.RichTextEditor	63
Syncfusion.Blazor.Schedule	63
Syncfusion.Blazor.SmithChart.....	64
Syncfusion.Blazor.Sparkline	64
Syncfusion.Blazor.Spinner.....	64
Syncfusion.Blazor.SplitButtons	65
Syncfusion.Blazor.StockChart	65
Syncfusion.Blazor.Themes	65
Syncfusion.Blazor.TreeGrid.....	66
Syncfusion.Blazor.TreeMap	66
Syncfusion.Blazor.WordProcessor	66
Getting Started.....	67
Getting Started with Blazor Server Side App in Visual Studio	67
Prerequisites	67
Create a new Blazor Server App in Visual Studio.....	67
Install Syncfusion Blazor Packages in the App	67
Add Style Sheet	67
Add Script Reference	68
Register Syncfusion Blazor Service.....	68
Add Syncfusion Blazor component	69
Getting Started with Blazor WebAssembly App in Visual Studio.....	70
Prerequisites	70
Create a new Blazor WebAssembly App in Visual Studio	70

Install Syncfusion Blazor Packages in the App	70
Add Style Sheet	70
Add Script Reference	71
Register Syncfusion Blazor Service.....	71
Adding Syncfusion Blazor component and running the application.....	73
Getting started with Blazor WebAssembly App in VS for Mac	74
Prerequisites	74
Create a Blazor WebAssembly project in Visual Studio for Mac	75
Installing Syncfusion Blazor packages in the application.....	78
Adding Syncfusion Blazor component and running the application.....	81
Getting started with Blazor ASP.NET Core Hosted App in Visual Studio	82
Prerequisites	83
Create a Blazor ASP.NET Core Hosted project in Visual Studio	83
Installing Syncfusion Blazor packages in the application.....	83
Adding Syncfusion Blazor component and running the application.....	87
See Also	89
Getting started with Blazor ASP.NET Core Hosted App in .NET Core CLI	89
Prerequisites	89
Create a Blazor ASP.NET Core Hosted project using .NET Core CLI	89
Adding Syncfusion Blazor component and running the application.....	91
See Also	93
Creating Razor Class Library (RCL) using Syncfusion Blazor components.....	93
Prerequisites	93
Create a Razor Class Library using Syncfusion Blazor components in Visual Studio 2019	93
Create a Razor Class Library using Syncfusion Blazor components in Visual Studio 2022	97
Create a Blazor Server project in Visual Studio with Razor Class Library (RCL)	105
Create a Blazor WebAssembly project in Visual Studio with Razor Class Library (RCL)	110
License Key.....	116
Syncfusion Licensing Overview	116
Generate Syncfusion Blazor License key.....	117
Register Syncfusion License key in Blazor application	117
Register Syncfusion license key in a Razor Class Library application.....	119
Syncfusion Licensing Errors.....	120
Syncfusion Licensing FAQ.....	124
Appearance	125

Blazor Themes in Syncfusion Components	125
Reference themes in Blazor application	126
Static Web Assets	126
CDN Reference	127
NPM Packages.....	128
LibMan	129
Change theme dynamically	130
Size Mode for Blazor Components	135
Size mode for application	135
Size mode for a control	135
Change size mode for application at runtime.....	136
Change size mode for a control at runtime	138
See Also	140
Theme Studio in Blazor Components.....	141
Customizing theme color from theme studio.....	141
Import previously changed settings into theme studio.....	149
Common Variables	151
Blazor Icons Library	164
Icon component	164
Icon integration with Button component	167
Using icons directly in HTML element.....	167
Icons list	168
Common.....	168
Version Compatibility of Syncfusion Blazor Components.....	168
Version Information	169
See also	169
Reference scripts in Blazor Application	169
JavaScript isolation.....	169
Disable JavaScript isolation	169
Import previously generated settings into CRG.....	172
Accessibility support with ADA compliance in Syncfusion Blazor components	173
Section 508.....	173
Keyboard navigation	173
Localization (Multi-Language) support in Syncfusion Blazor components.....	174
How to enable Localization in Blazor application	174

How Syncfusion Blazor UI Component supports Localization	174
Enable Localization in Blazor WebAssembly application	179
Globalization	183
Right-To-Left support in Syncfusion Blazor Components	183
Enable RTL for all components	183
Enable RTL to individual component	185
State Persistence.....	186
State Persistence Supported Components and Properties	187
Input Form Validation and Data Annotation	187
How to Validate Syncfusion Blazor UI Components	187
Blazor Form Validation Supported Components	191
Apply Custom Validation Attributes	204
See Also	207
Default HTML Attributes and DOM Events.....	207
Using HTML Attributes and DOM Events in the Input Element.....	207
Using HTML Attributes and DOM Events in the Root Element.....	210
Deployment in Blazor.....	210
Publish Blazor Application with Visual Studio 2019.....	210
Publish Blazor Application with CLI.....	212
Data Binding.....	212
Bind data to the Syncfusion Blazor components using WebApiAdaptor of SfDataManager and perform CRUD operations.....	221
SQL server data binding and performing CRUD operations	231
Bind data from SQL server to Syncfusion Blazor components	239
How to bind data using Dapper and perform CRUD operations	248
How to bind data to the Syncfusion Blazor components using gRPC service.....	257
How To	266
Extend, Customize, and Reuse Components	266
How to Add Syncfusion Blazor Component on an Existing ASP.NET Core MVC Application.....	274
How to Render the Blazor Server Application in IE11 Web Browser.....	278
How to upgrade Syncfusion Blazor Components to the latest version	280
How to upgrade Syncfusion Blazor Components to 18.1.0.36 version	281
How to Create a Blazor WebAssembly Application with Prerendering.....	283
How to Configure Syncfusion Blazor Component in bUnit Testing	289
How to Use Syncfusion Blazor ReportViewer	309

How to troubleshoot server and client exceptions in Blazor.....	309
Installation	314
Web Installer	314
Downloading Syncfusion Blazor web installer	314
Installing Syncfusion Blazor web installer	317
Offline Installer	328
Downloading Syncfusion Blazor offline installer.....	328
Installing Syncfusion Blazor offline installer	332
Mac Installer	346
Downloading Syncfusion Blazor Mac installer	346
Installing Syncfusion Blazor Mac installer	349
Install Syncfusion Blazor NuGet packages	357
Overview	357
Installation using Package Manager UI	358
Installation using Dotnet (.NET) CLI	360
Upgrade.....	361
Visual Studio Code Integration	361
Visual Studio Code Extension.....	361
Visual Studio Integration.....	361
VS 2019 Extension.....	361
Accordion	361
Adding component package to the application	362
Add SyncfusionBlazor service in Startup file.....	362
Adding Accordion component to the application.....	363
Run the application	363
Initialize Accordion using Template	364
See Also	364
Data binding in Blazor Accordion Component.....	364
Expand Mode in Blazor Accordion Component.....	366
Single	366
Multiple	367
Expanding the items.....	367
Animations in Blazor Accordion Component.....	369
Accessibility in Blazor Accordion Component.....	371
ARIA attributes	371

Keyboard interaction	372
Style and Appearance in Blazor Accordion Component	372
Customizing Accordion	372
Customizing the Accordion's items.....	373
Customizing Accordion's header.....	373
Customizing Accordion's expand and collapse icons.....	373
Customizing the hover state of Accordion control	373
Customizing selected item of Accordion control	373
Content Render Mode in Blazor Accordion Component	374
How To	374
Add/Remove Accordion items in Blazor Accordion Component	374
Show/Hide Accordion item in Blazor Accordion Component	376
Enable or Disable item in Blazor Accordion Component	379
Add Icon to Header in Blazor Accordion Component	381
Prevent the Expand or Collapse item in Blazor Accordion Component	387
Add Nested Accordion in Blazor Accordion Component	389
Create Wizard in Blazor Accordion Component	390
Treeview Integration in Blazor Accordion Component.....	395
Accumulation Chart	398
Blazor Accumulation Chart in Server Side App using Visual Studio	398
Importing Syncfusion Blazor component in the application.....	399
Adding component package to the application	399
Add SyncfusionBlazor service in Startup.cs	399
Add Accumulation Chart	401
Add Title	402
Add Data Label	403
Enable Tooltip	404
Enable Legend.....	406
See also	407
Chart Types	407
Pie and Doughnut in Blazor Accumulation Chart Component.....	407
Pyramid in Blazor Accumulation Chart Component	419
Funnel in Blazor Accumulation Chart Component.....	426
Data Label in Blazor Accumulation Chart Component	434
Position	436

Smart Labels.....	437
Connector Line	439
Text Mapping	441
See Also	442
Grouping in Blazor Accumulation Chart Component	442
Pie Grouping.....	444
Empty Points in Blazor Accumulation Chart Component	448
Customization	450
See Also	452
Annotation in Blazor Accumulation Chart Component	452
Region	453
Co-ordinate Units	454
See Also	456
Animation in Blazor Accumulation Chart Component.....	456
Tooltip in Blazor Accumulation Chart Component	457
Header.....	458
Tooltip Format	460
Tooltip Customization	461
Tooltip Text Mapping.....	463
See Also	464
Legend in Blazor Accumulation Chart Component.....	464
Position and Alignment.....	466
Legend Shape	467
Legend Size.....	469
Legend Shape Size.....	470
Paging for Legend.....	471
Title and Subtitle in Blazor Accumulation Chart Component	473
Title Customization	474
Subtitle	475
Subtitle Customization	477
See Also	478
Print and Export in Blazor Accumulation Chart Component	479
Print.....	479
Export.....	480
See Also	481

Events in Blazor Accumulation Chart Component	481
OnDataLabelRender	482
OnLegendItemRender	483
OnPointRender	483
OnExportComplete	484
OnPrintComplete	485
SizeChanged	486
Loaded	487
OnPointClick	487
TooltipRender	488
How To	489
Text placing center of the Blazor Doughnut Chart Component	489
AutoComplete	491
Getting Started with Blazor AutoComplete Component	491
Importing Syncfusion Blazor component in the application	491
Adding component package to the application	491
Add SyncfusionBlazor service in Startup.cs	492
Adding AutoComplete component to the application	492
Run the application	493
Binding data source	493
Custom values	494
Configure the suggestion list	495
See Also	496
Data Binding in Blazor AutoComplete Component	496
Data Source in Blazor AutoComplete Component	496
Bind to local data	497
Bind to remote data	499
Binding ExpandoObject	504
Binding DynamicObject	505
Binding ObservableCollection	506
Entity Framework	507
Grouping in Blazor AutoComplete Component	509
Filtering in Blazor AutoComplete Component	510
Change the filter type	511
Filter item count	512

Limit the minimum filter character	513
Case sensitive filtering	514
Custom Filtering	514
Templates in Blazor AutoComplete Component	515
Item template	515
Group template.....	516
Header template	517
Footer template	518
No records template	520
Action failure template	520
Localization in Blazor AutoComplete Component	521
Blazor server side	521
Blazor WebAssembly	524
Style and appearance in Blazor AutoComplete Component	526
Customizing the appearance of wrapper element	526
Customizing the dropdown icon's color	526
Customizing the focus color	526
Customizing the outline theme's focus color	526
Customizing the disabled component's text color	527
Customizing the float label element's focusing color	527
Customizing the color of the placeholder text	527
Customizing the placeholder to add mandatory indicator(*).....	527
Customizing the text selection color.....	527
Customizing the background color of focus, hover, and active item's	528
Customizing the appearance of pop-up element	528
Adding search icon in the Blazor AutoComplete component.....	528
Virtualization in Blazor AutoComplete Component	529
Native Events in Blazor AutoComplete Component	530
Bind native events to AutoComplete.....	530
Pass event data to event handler	531
List of Native events supported	531
Accessibility in Blazor AutoComplete Component	532
ARIA attributes.....	532
Keyboard interaction	532
Events in Blazor AutoComplete Component	534

Blur	534
ValueChange	534
Closed.....	535
Created.....	535
Destroyed.....	536
Focus	536
OnOpen	537
OnClose	537
DataBound	538
Filtering	538
OnActionBegin	539
OnActionFailure	540
OnValueSelect.....	541
Opened.....	541
Avatar.....	542
Getting Started with Blazor Avatar Component	542
Importing Syncfusion Blazor component in the application.....	542
Adding Avatar component to the application	543
Run the application	543
Badge	543
Getting Started with Blazor Badge Component.....	543
Importing Syncfusion Blazor component in the application.....	544
Adding Badge component to the application	544
Run the application	544
Barcode	545
Getting Started with Blazor Barcode Component	545
Importing Syncfusion Blazor component in the application.....	545
Adding component package to the application	545
Add SyncfusionBlazor service in Startup.cs	546
Adding BarcodeGenerator component to the Application.....	546
Adding QR Generator control	547
Adding Data Matrix Generator control	547
See Also	548
Barcode Generator in Blazor Barcode Component	548
Code39	548

Code39 Extended	548
Code 11	549
Codabar	549
Code 32	550
Code 93	550
Code 93 Extended	551
Code 128	551
Customizing the Barcode color	552
Customizing the Barcode dimension	552
Customizing the text	553
Event	554
QR Code generator in Blazor Barcode Component	554
QR Code	554
Customizing the Barcode color	555
Customizing the Barcode dimension	555
Customizing the text	555
Event	556
Data Matrix generator in Blazor Barcode Component	556
Data Matrix	556
Customizing the Barcode color	557
Customizing the Barcode dimension	557
Customizing the text	557
Event	558
Export in Blazor Barcode Component.....	558
Export.....	558
Breadcrumb	559
Getting Started with Blazor Breadcrumb Component.....	559
Prerequisites	559
Create a new Blazor App in Visual Studio	559
Install Syncfusion Blazor Navigations NuGet in the App.....	559
Add Style Sheet	559
Add Script Reference	560
Register Syncfusion Blazor Service.....	560
Add Syncfusion Blazor Breadcrumb component	562
Add items to the Breadcrumb component	563

Enable or disable navigation	563
See Also	563
Breadcrumb items in Blazor Breadcrumb component	563
Items based on current URL.....	564
Absolute URL.....	564
Icons in Blazor Breadcrumb component.....	565
Breadcrumb with font icon	565
Breadcrumb with image.....	565
Breadcrumb with SVG image	566
Icon only.....	567
Show icon only for first item.....	567
Navigation in Blazor Breadcrumb component.....	567
Relative URL	567
Absolute URL.....	568
Enable navigation for last Breadcrumb item	569
Overflow mode in Blazor Breadcrumb component	569
Templates in Blazor Breadcrumb component	570
Template context	570
Item template	570
Separator template	571
Bullet Chart	572
Getting Started with Blazor Bullet Chart Component	572
Importing Syncfusion Blazor Bullet Chart component in the application.....	572
Adding component package to the application.....	572
Adding SyncfusionBlazor service in Startup.cs	572
Adding Bullet Chart component	573
Adding Title	573
Adding Ranges.....	574
Adding Tooltip.....	575
Bullet Chart Dimensions in Blazor Bullet Chart Component.....	575
Size for Container.....	575
Size for Bullet Chart.....	576
Margin.....	576
Axis Customization in Blazor Bullet Chart Component.....	577
MajorTickLines and MinorTickLines Customization.....	577

Tick Placement	578
Label Format	578
Grouping Separator.....	579
Custom Label Format.....	580
Label Placement.....	580
Opposed Position	581
Category Label	581
Axis Label and Category Label Customization	582
Working with Data in Blazor Bullet Chart Component	582
Ranges in Blazor Bullet Chart Component.....	583
Color Customization.....	584
Actual Bar in Blazor Bullet Chart Component.....	585
Types of Actual Bar	585
Actual Bar Customization.....	586
Target Bar in Blazor Bullet Chart Component.....	586
Types of Target Bar	587
Target Bar Customization.....	587
Title and Subtitle in Blazor Bullet Chart Component	588
Title	588
Subtitle	588
Title and Subtitle Position	589
Title and Subtitle Customization.....	590
Customization in Blazor Bullet Chart Component	590
Orientation.....	590
Right-to-left (RTL).....	591
Animation.....	592
Theme	592
Border	593
Data Labels in Blazor Bullet Chart Component.....	593
Data Label Customization	594
Tooltip in Blazor Bullet Chart Component	595
Default Tooltip	595
Tooltip Customization	595
Tooltip Template.....	596
Legend in Blazor Bullet Chart Component.....	597

Legend items from color mapping	598
Legend size	599
Legend with paging support	599
Position and Alignment.....	600
Customization	602
Events in Blazor Bullet Chart Component.....	603
Loaded.....	603
OnPrintComplete	603
TooltipRender	604
LegendRender	605
Button	605
Getting Started with Blazor Button Component.....	606
Importing Syncfusion Blazor component in the application.....	606
Adding component package to the application.....	606
Add SyncfusionBlazor service in Startup.cs	606
Adding component package to the application.....	607
Adding Button component to the application	607
Run the application	607
See Also	608
Native Events in Blazor Button Component	608
List of Native events supported	608
How to bind click event to Button	608
Types and Styles in Blazor Button Component.....	609
Button styles	609
Button types.....	609
Icons	611
Button size	612
Styles and Appearances in Blazor Button Component	612
How To	613
Create a Block Button in Blazor Button Component	613
Customize Button Appearance in Blazor Button Component	613
HTML Attribute Support in Blazor Button Component.....	613
Repeat Button in Blazor Button Component	614
Right-To-Left in Blazor Button Component	614
Set the disabled state in Blazor Button Component.....	615

Tooltip for Button in Blazor Button Component.....	615
ButtonGroup	616
Getting Started with Blazor ButtonGroup Component	616
Importing Syncfusion Blazor component in the application.....	616
Adding component package to the application.....	616
Add SyncfusionBlazor service in Startup.cs	617
Adding component package to the application	617
Adding ButtonGroup component to the application.....	617
Run the application	618
See Also	618
Types and Styles in Blazor ButtonGroup Component.....	618
ButtonGroup styles	618
ButtonGroup types	619
Icons	620
ButtonGroup size	620
Selection and Nesting in Blazor ButtonGroup Component	621
Single selection	621
Multiple selection	621
Nesting	621
Styles and Appearances in Blazor ButtonGroup Component	623
Calendar	623
Getting Started with Blazor Calendar Component	623
Importing Syncfusion Blazor component in the application.....	623
Adding component package to the application.....	624
Add SyncfusionBlazor service in Program.cs	624
Adding Calendar component to the application.....	625
Run the application	625
Setting the Value, Min, and Max dates.....	625
See Also	626
Data Binding in Blazor Calendar Component.....	626
One-Way Binding	626
Two-Way Data Binding.....	627
Dynamic Value Binding	627
Data Range in Blazor Calendar Component.....	627
Multi Selection in Blazor Calendar Component.....	628

Calendar Views in Blazor Calendar Component	629
Set the initial view.....	629
View Restriction	630
Accessibility in Blazor Calendar Component.....	631
Keyboard interaction	631
Globalization in Blazor Calendar Component	632
Blazor server side	632
Blazor WebAssembly	634
Customize the localized text	636
Right-To-Left	637
Events in Blazor Calendar Component.....	637
OnRenderDayCell	638
ValueChanged	638
Created.....	638
Destroyed.....	639
Navigated	639
Special Dates in Blazor Calendar Component.....	639
How To	641
Show Dates of Other Months in Blazor Calendar Component	641
Week Number in Blazor Calendar Component.....	642
Change the First Day of Week in Blazor Calendar Component	644
Card.....	645
Getting Started with Blazor Card Component	645
Importing Syncfusion Blazor component in the application.....	645
Adding component package to the application.....	646
Add SyncfusionBlazor service in Startup.cs	646
Adding Card component	646
Adding a header and content	647
Run the application	647
Header and Content in Blazor Card Component	647
Header.....	647
Content	648
Image and Divider in Blazor Card Component.....	649
Images.....	649
Divider	649

Action Buttons in Blazor Card Component	650
Vertical	650
Horizontal Card in Blazor Card Component	650
Stacked cards	650
Style and Appearance in Blazor Card Component	651
Customizing the card	651
Customizing the Header element	652
Customizing the card content	652
Divider used to separate the elements inside the card	652
Including image within card element	652
Including a title or caption for the image	652
To include heading image within the header	652
Customizing the Header main title	653
Customizing the Header subtitle	653
Including action buttons or anchor tags	653
To align card elements horizontally	653
To align elements vertically within the horizontal layout	653
Charts	654
Blazor Charts Component in Server Side App using Visual Studio	654
Importing Syncfusion Blazor component in the application	654
Adding component package to the application	655
Add SyncfusionBlazor service in Startup.cs	655
Add Chart Component	656
Populate Chart with Data	657
Add Titles	658
Add Data Label	659
Enable Tooltip	661
Enable Legend	662
See also	663
Blazor Charts Component in WebAssembly App using Visual Studio	664
Create a Blazor WebAssembly project in Visual Studio 2019	664
Importing Syncfusion Blazor component in the application	666
Adding component package to the application	668
Add SyncfusionBlazor service in Startup.cs	669
Add Chart Component	669

Populate Chart with Data.....	670
Add Titles	671
Add Data Label	673
Enable Tooltip	674
Enable Legend	676
See also	677
Working with Data in Blazor Charts Component.....	677
List binding	678
Remote Data	680
Entity Framework.....	683
Empty points	687
See Also	689
Chart Dimensions in Blazor Charts Component.....	689
Size for Container.....	690
Size for Chart.....	691
See Also	693
Category Axis in Blazor Charts Component	693
Labels Placement	694
Range and Interval	696
Indexed Category Axis.....	697
See Also	698
Numeric Axis in Blazor Charts Component	699
Range and Interval	700
Range Padding.....	701
Label Format	707
Custom Label Format.....	709
See Also	710
DateTime Axis in Blazor Charts Component	710
DateTime Axis	710
DateTime Category Axis	711
Label Format	718
See Also	720
Logarithmic Axis in Blazor Charts Component.....	720
Range	721
Logarithmic Base.....	722

Logarithmic Interval	723
Label Format	724
Custom Label Format	727
See Also	728
Axis Labels in Blazor Charts Component.....	728
Smart Axis Labels	728
Axis Labels Positioning	732
Multilevel Labels	734
Edge Label Placement	741
Labels Customization	743
Trim Label.....	744
Line Break.....	745
Label Format	747
See Also	747
Axis Customization in Blazor Charts Component.....	747
Axis Crossing	747
Title	749
Tick Lines	750
Grid Lines Customization	752
Multiple Axis	753
See also	754
Inversed Axis	754
Opposed Position	756
See Also	757
Stripline in Blazor Charts Component.....	757
Horizontal Striplines.....	758
Vertical Striplines	759
Striplines Customization	760
Text Customization	762
See Also	763
Multiple Panes in Blazor Charts Component	763
Rows.....	763
Columns	766
See Also	769
Chart Types	770

Line Chart in Blazor Charts Component.....	770
Step Line Chart in Blazor Charts Component.....	774
Stacked Line Chart in Blazor Charts Component	777
100% Stacked Line in Blazor Charts Component	781
Spline in Blazor Charts Component	785
Area in Blazor Charts Component.....	790
Range Area in Blazor Charts Component.....	795
Stacked Area in Blazor Charts Component	798
100% Stacked Area in Blazor Charts Component	801
Step Area in Blazor Charts Component.....	805
Spline Area in Blazor Charts Component.....	808
Column Chart in Blazor Charts Component	811
Range Column in Blazor Charts Component.....	815
Stacked Column in Blazor Charts Component	819
100% Stacked Column in Blazor Charts Component	824
Bar Charts in Blazor Charts Component	828
Stacked Bar in Blazor Charts Component	832
100% Stacked Bar in Blazor Charts Component	836
Scatter in Blazor Charts Component.....	840
Bubble in Blazor Charts Component.....	843
Polar in Blazor Charts Component.....	847
Radar in Blazor Charts Component.....	859
Hilo in Blazor Charts Component.....	864
High Low Open Close in Blazor Charts Component	867
Candle in Blazor Charts Component	869
Box and Whisker in Blazor Charts Component	873
Waterfall in Blazor Charts Component	877
Histogram in Blazor Charts Component.....	881
Error Bar in Blazor Charts Component.....	882
Vertical Chart in Blazor Charts Component	888
Pareto in Blazor Charts Component.....	890
Mixed Chart in Blazor Charts Component	891
Multiple Chart Series.....	891
Combination Chart Series	893
See Also	895

Markers in Blazor Charts Component.....	895
Markers	895
Shape.....	897
Images	898
Customization	899
See Also	900
Data Labels in Blazor Charts Component.....	900
Position	901
Template	903
Text Mapping	903
Margin.....	904
Customization	906
See Also	907
Datalabel Template in Blazor Charts Component.....	908
Annotation in Blazor Charts Component.....	909
Region	911
Co-ordinate Units.....	912
See Also	914
Appearance in Blazor Charts Component.....	914
Custom Color Palette	914
Chart Customization	916
Animation.....	920
Chart Title.....	921
Chart Subtitle	922
See Also	924
Legend in Blazor Charts Component.....	924
Enable Legend.....	924
Position and Alignment.....	925
Legend Customization.....	929
Series selection based on legend	935
Hiding legend item	937
See Also	939
Tooltip in Blazor Charts Component.....	939
Enable Tooltip	939
Tooltip Format	940

Tooltip Customization	941
See Also	943
Zooming in Blazor Charts Component	943
Enable Zooming	943
Modes	944
Toolbar	945
Enable Pan.....	947
Enable Scrollbar	948
Auto interval on zooming.....	950
See Also	952
Data Editing in Blazor Charts Component.....	952
See Also	954
Crosshair and Trackball in Blazor Charts Component.....	954
Enable Tooltip	955
Crosshair Customization	957
Trackball.....	958
See Also	960
Selection in Blazor Charts Component	960
Point.....	961
Series.....	962
Cluster	963
Rectangular Selection	965
Multiple Selection	967
Selection via code-behind.....	968
Legend Selection	970
Selection Customization	971
See Also	973
Print and Export in Blazor Charts Component.....	973
Print.....	973
Export.....	975
See Also	976
Technical Indicators in Blazor Charts Component.....	976
Accumulation Distribution	976
Average True Range (ATR)	979
Bollinger Bands	982

Exponential Moving Average (EMA)	989
Momentum	993
Moving Average Convergence Divergence (MACD)	1000
Relative Strength Index (RSI).....	1008
Simple Moving Average (SMA)	1012
Stochastic	1015
Triangular Moving Average (TMA)	1019
See Also	1025
Trendlines in Blazor Charts Component	1026
Linear.....	1026
Exponential	1027
Logarithmic	1029
Polynomial	1031
Power	1033
Moving Average	1034
Forecasting.....	1036
Trendlines Customization	1039
See Also	1039
Internationalization in Blazor Charts Component	1039
Globalization	1039
Label Format	1040
See Also	1041
Localization in Blazor Charts Component	1041
Blazor Server Side	1041
Blazor WebAssembly	1042
See Also	1043
Events in Blazor Charts Component.....	1044
OnZoomStart.....	1045
OnZoomEnd	1045
OnZooming.....	1046
OnLegendItemRender	1047
OnDataLabelRender	1048
OnPointRender.....	1049
OnAxisLabelRender	1050
OnAxisLabelClick	1051

OnAxisActualRangeCalculated	1051
OnAxisMultiLevelLabelRender	1052
SizeChanged	1053
OnScrollChanged.....	1054
OnExportComplete	1055
OnDataEdit.....	1056
OnDataEditCompleted	1057
OnLegendClick.....	1058
OnMultiLevelLabelClick.....	1059
OnSelectionChanged.....	1060
Loaded.....	1061
OnPointClick.....	1061
TooltipRender	1062
SharedTooltipRender	1063
How To	1064
Add or Remove Series in Blazor Charts Component.....	1064
Lazy Loading in Blazor Charts Component.....	1067
Table in Tooltip in Blazor Charts Component	1069
Visualize grid data in chart in Blazor Charts Component.....	1071
Threshold in Chart in Blazor Charts Component	1074
Live Chart in Blazor Charts Component	1076
Hiding Axis in Blazor Charts Component.....	1079
Convert millisecond to date time in Blazor Charts Component	1080
CheckBox.....	1081
Getting Started with Blazor CheckBox Component	1081
Importing Syncfusion Blazor component in the application.....	1081
Adding component package to the application	1082
Add SyncfusionBlazor service in Startup.cs	1082
Adding component package to the application	1082
Adding Checkbox component to the application	1082
Run the application	1083
See Also	1083
Native Events in Blazor CheckBox Component.....	1083
List of Native events supported	1083
How to bind onchange event to Checkbox	1083

Label and Size in Blazor CheckBox Component	1084
Label	1084
Size	1084
See Also	1085
Accessibility in Blazor CheckBox Component	1085
Keyboard interaction	1085
Styles and Appearances in Blazor CheckBox Component.....	1086
How To	1086
Customized Checkbox in Blazor CheckBox Component	1086
Model Binding in Blazor CheckBox Component	1091
Right-To-Left in Blazor CheckBox Component	1092
Chip	1092
Getting Started with Blazor Chip Component	1092
Importing Syncfusion Blazor component in the application.....	1092
Add Syncfusion Blazor service in Startup.cs (Server-side application)	1096
Add Syncfusion Blazor service in Program.cs (Client-side application)	1097
Adding component package to the application	1097
Adding Chip component to the application	1098
Run the application	1098
See Also	1098
Types in Blazor Chip Component	1098
Input Chip.....	1098
Choice Chip	1099
Filter Chip	1099
Action Chip	1100
Customization in Blazor Chip Component	1101
Styles	1101
Leading Icon	1101
Avatar.....	1102
Leading Content	1103
Trailing Icon.....	1103
Outline Chip	1103
CSS Structure in Blazor Chip Component.....	1104
Customizing the chip text	1104
Customizing the chip icon	1104

Customizing the chip delete button.....	1104
Customizing the chip outline	1105
Customizing the chip on selection	1105
Customizing the chip avatar text	1105
Accessibility in Blazor Chip Component.....	1105
Keyboard Interaction	1105
Circular Gauge.....	1106
Getting Started with Blazor Circular Gauge Component	1106
Importing Syncfusion Blazor Circular Gauge component in the application	1106
Adding component package to an application	1107
Adding SyncfusionBlazor Service in Startup.cs	1107
Adding Circular Gauge component.....	1107
Set pointer value.....	1108
Adding title for Circular Gauge.....	1109
Adding ranges in the Circular Gauge.....	1109
See also	1110
Dimensions in Blazor Circular Gauge Component	1110
Size for Circular Gauge	1110
Axes in Blazor Circular Gauge Component	1111
Axis customization	1111
Minimum and maximum.....	1111
Start and end angle	1112
Axis radius	1112
Ticks.....	1113
Labels	1114
Axis direction.....	1120
Multiple axes.....	1121
Ranges in Blazor Circular Gauge Component	1122
Range start and end.....	1122
Start width and end width	1122
Changing color	1123
Range Position	1124
Rounded corners.....	1124
Radius.....	1125
Dragging ranges	1126

Multiple ranges	1127
Gradient Color.....	1127
See also	1132
Pointers in Blazor Circular Gauge Component.....	1132
Needle pointer	1133
Range bar pointer	1136
Marker pointer	1138
Dragging pointer	1141
Multiple pointers	1142
Pointer animation	1143
Gradient Color.....	1144
Annotations in Blazor Circular Gauge Component	1146
Customization	1146
Positioning the annotation	1147
Multiple annotations	1148
See also	1150
Legend in Blazor Circular Gauge Component	1150
Legend customization	1150
Position and alignment	1150
Font of the legend text	1151
Toggle option in legend	1153
Paging support in legend	1154
Legend text customization.....	1155
User Interaction in Blazor Circular Gauge Component.....	1156
Tooltip for pointers	1156
Tooltip for ranges.....	1158
Tooltip for annotations	1159
Dragging pointer	1160
Dragging Range	1161
Print and Export in Blazor Circular Gauge Component.....	1161
Print.....	1161
Export.....	1162
Appearance in Blazor Circular Gauge Component	1164
Circular gauge title	1164
Circular gauge position	1164

Background customization.....	1165
Radius calculation based on angles	1166
Accessibility in Blazor Circular Gauge Component	1167
Globalization in Blazor Circular Gauge Component.....	1167
Events in Blazor Circular Gauge Component	1168
Using events in Circular Gauge component.....	1168
Available events	1169
Methods in Blazor Circular Gauge Component	1172
SetAnnotationValueAsync	1172
SetPointerValueAsync.....	1173
SetRangeValue	1174
RefreshAsync.....	1175
Color Picker	1175
Getting Started with Blazor Color Picker Component	1175
Importing Syncfusion Blazor component in the application.....	1175
Adding component package to the application	1176
Add SyncfusionBlazor service in Startup.cs	1176
Adding Color Picker component to the application.....	1177
Run the application	1177
See Also	1177
Accessibility in Blazor Color Picker Component.....	1177
ARIA attributes.....	1177
Keyboard interaction	1178
Inline Rendering in Blazor Color Picker Component.....	1178
Mode and Value in Blazor Color Picker Component.....	1179
Rendering palette at initial load	1179
Color value	1180
Localization and RTL in Blazor Color Picker Component	1181
Localization	1181
RTL.....	1182
Styles and Appearances in Blazor Color Picker Component	1183
How To	1183
Customize Color Picker in Blazor Color Picker Component	1183
Disable Color Picker in Blazor Color Picker Component	1187
Handle No Color Support in Blazor Color Picker Component.....	1187

Hide control buttons in Blazor Color Picker Component.....	1191
Render palette alone in Blazor Color Picker Component	1191
ComboBox.....	1192
Getting Started with Blazor ComboBox Component	1192
Importing Syncfusion Blazor component in the application.....	1192
Adding component package to the application	1193
Add SyncfusionBlazor service in Startup.cs	1193
Adding ComboBox component to the application.....	1193
Run the application	1194
Binding data source	1194
Custom values.....	1195
Configure the popup list	1196
See Also	1197
Data Binding in Blazor ComboBox Component	1197
Index Value Binding	1198
Data Source in Blazor ComboBox Component	1198
Binding local data.....	1198
Binding remote data	1201
Binding ExpandoObject.....	1206
Binding DynamicObject.....	1207
Binding ObservableCollection.....	1208
Entity Framework.....	1209
Grouping in Blazor ComboBox Component	1211
Filtering in Blazor ComboBox Component.....	1213
Custom Filtering	1213
Templates in Blazor ComboBox Component	1214
Item template	1214
Group template.....	1215
Header template	1217
Footer template	1218
No records template	1219
Action failure template	1220
Localization in Blazor ComboBox Component	1220
Blazor server side	1220
Blazor WebAssembly	1223

Style and appearance in Blazor ComboBox Component	1225
Customizing the appearance of wrapper element	1225
Customizing the dropdown icon's color	1225
Customizing the focus color	1225
Customizing the outline theme's focus color	1225
Customizing the disabled component's text color	1226
Customizing the float label element's focusing color	1226
Customizing the color of the placeholder text	1226
Customizing the placeholder to add mandatory indicator(*).....	1226
Customizing the text selection color.....	1227
Customizing the background color of focus, hover, and active item's	1227
Customizing the appearance of pop-up element	1227
Virtualization in Blazor ComboBox Component	1227
Adding Custom Value to Blazor ComboBox Component	1228
Native Events in Blazor ComboBox Component	1230
Bind native events to ComboBox.....	1230
Pass event data to event handler	1231
List of Native events supported	1231
Accessibility in Blazor ComboBox Component	1232
ARIA attributes.....	1232
Keyboard interaction	1232
Events in Blazor ComboBox Component	1234
Blur	1234
ValueChange	1235
Closed.....	1235
Created.....	1236
Destroyed.....	1236
Focus	1237
OnOpen	1237
OnClose	1238
DataBound	1238
Filtering	1239
OnActionBegin	1239
OnActionComplete.....	1240
OnActionFailure	1241

OnValueSelect.....	1242
Opened.....	1242
How To	1243
DropDownList options with tooltip in Blazor ComboBox Component	1243
ContextMenu	1244
Getting Started with Blazor ContextMenu Component	1244
Importing Syncfusion Blazor component in the application.....	1244
Adding component package to the application	1245
Add SyncfusionBlazor service in Startup.cs	1245
Adding Context Menu component to the application.....	1246
Run the application	1246
See Also	1246
Accessibility in Blazor ContextMenu Component.....	1247
ARIA attributes.....	1247
Keyboard interaction	1247
Customizing and Multilevel nesting in Blazor ContextMenu Component.....	1247
Customizing Context Menu Items.....	1247
Multilevel nesting	1249
Icons and Navigation in Blazor ContextMenu Component.....	1250
Icons	1250
Navigation	1251
Styles and Appearances in Blazor ContextMenu Component	1252
How To	1252
Bind Context Menu Events in Blazor ContextMenu Component	1252
Change animation settings in Blazor ContextMenu Component	1253
Data Binding in Blazor ContextMenu Component.....	1254
Enable/Disable Context Menu items in Blazor ContextMenu Component	1255
Open a dialog on Item Click in Blazor Context Menu	1256
Open Sub Menu on Item Click in Blazor ContextMenu Component	1258
Open and close Context Menu in Blazor ContextMenu Component	1258
Dashboard Layout	1259
Getting Started with Blazor Dashboard Layout Component	1259
Importing Syncfusion Blazor component in the application.....	1259
Add Syncfusion Blazor service in Startup.cs (Server-side application)	1260
Add Syncfusion Blazor service in Program.cs (Client-side application)	1261

Adding Dashboard Layout component namespace to the application	1261
Initialize the Dashboard Layout component.....	1262
Run the application	1262
Defining panels	1262
See Also	1267
Configuring the Grid Layout in Blazor Dashboard Layout Component.....	1267
Modifying cell size.....	1268
Setting cell spacing.....	1268
Graphical representation of grid layout.....	1269
Panels	1270
Size and Position in Blazor Dashboard Layout Component	1270
Header and Content in Blazor Dashboard Layout Component	1273
Interaction With Panels	1274
Drag and Drop in Blazor Dashboard Layout Component.....	1274
Resizing Panels in Blazor Dashboard Layout Component.....	1276
Floating Panels in Blazor Dashboard Layout Component	1277
Responsive and Adaptive Layout in Blazor Dashboard Layout Component.....	1278
CSS Structure in Blazor Dashboard Layout Component	1279
Customizing the dashboard layout panel header	1279
Customizing the dashboard layout panel content.....	1280
Customizing the dashboard layout panel resize icon	1280
Customizing the dashboard layout panel background	1280
Accessibility in Blazor Dashboard Layout Component.....	1280
ARIA attributes.....	1280
DataManager	1280
Getting Started with Blazor DataManager Component	1281
Importing Syncfusion Blazor component in the application.....	1281
Adding component package to the application	1281
Add Data Manager Component	1281
Connection to a data source	1281
Component binding	1283
Data Binding in Blazor DataManager Component.....	1285
Local data binding	1285
Remote data binding.....	1287
Adaptors in Blazor DataManager Component.....	1288

Json adaptor	1288
Url adaptor	1289
ODataV4 adaptor	1291
Web API adaptor	1291
WebMethod adaptor	1292
Writing custom adaptor	1293
Custom Binding in Blazor DataManager Component	1294
How To	1298
Adding custom headers in Blazor DataManager Component	1298
Working in offline mode in Blazor DataManager Component	1299
DataGrid	1300
Getting Started with Blazor DataGrid Component	1300
Importing Syncfusion Blazor component in the application	1300
Adding component package to the application	1301
Add SyncfusionBlazor service in Startup.cs	1301
Add DataGrid Component	1301
Defining Row Data	1301
Defining Columns	1302
Enable Paging	1303
Enable Sorting	1304
Enable Filtering	1304
Enable Grouping	1305
Handling exceptions	1306
See Also	1307
Data Binding in Blazor DataGrid Component	1307
List binding	1308
Remote data	1314
SQL Server data binding(SQL Client)	1323
Entity Framework	1325
HTTP client	1328
Observable Collection	1330
Troubleshoot: DataGrid renders without data even though server returns with correct data	1332
Handling exceptions	1333
Custom Binding in Blazor DataGrid Component	1334
Data binding	1335

Inject service into Custom Adaptor	1337
Custom adaptor as Component	1339
CRUD operation	1342
Handling Aggregates in Custom Adaptor	1345
Handling Grouping in Custom Adaptor	1347
Data Annotation in Blazor DataGrid Component	1349
Columns in Blazor DataGrid Component	1351
Auto generation	1351
Dynamic column building.....	1352
Complex data binding	1353
Foreign key column.....	1358
Header text	1360
Header template	1362
Column type.....	1364
Format.....	1365
Visibility	1366
Width	1367
Autofit	1367
Responsive columns.....	1370
Controlling datagrid actions.....	1371
Show/hide columns by external button	1373
Render boolean values as checkbox	1378
Row in Blazor DataGrid Component	1379
Row template.....	1379
Detail Template.....	1385
Customize rows.....	1397
Styling alternate rows	1398
Row height	1400
Cell in Blazor DataGrid Component	1403
Displaying the HTML content.....	1403
Customize cell styles	1404
Auto wrap	1406
Custom Attributes.....	1408
DataGrid Lines.....	1409
Clip Mode	1410

Templates in Blazor DataGrid Component	1412
Template context	1412
GridTemplates component	1414
Grouping in Blazor DataGrid Component	1416
Initial group	1418
Hide drop area	1419
Group by format	1421
Grouping events	1421
Caption template	1422
Lazy Load Grouping	1424
Filtering in Blazor DataGrid Component	1430
Initial filter	1431
Filter operators	1433
Filter menu	1438
CheckBox Filter	1443
Excel like filter	1445
Initial sort	1449
Multi-column sorting	1450
Sort order	1451
Sorting events	1451
Custom sort comparer	1452
Touch interaction	1454
Searching in Blazor DataGrid Component	1454
Initial search	1455
Search operators	1457
Search by external button	1457
Search specific columns	1458
Disable search for particular column	1459
Immediate Searching	1460
Editing in Blazor DataGrid Component	1461
Toolbar with edit option	1463
Edit Modes	1465
Edit next row or previous row from the current row	1469
Cell edit type	1471
Customizing the default editor controls	1471

Cell Edit Template	1480
Command column	1487
Column validation	1491
Provide new item or edited item using events	1499
Entity Framework.....	1502
Perform CRUD operation using Grid Events	1505
Performing CRUD operations programmatically	1508
Custom external form editing	1510
Dialog template.....	1512
Inline Template	1524
Adding a new row at the bottom of the datagrid.....	1527
Confirmation messages.....	1528
Default column values on adding new record	1531
Disable editing for particular column	1532
Troubleshoot: Editing works only for first row	1534
Event trace while editing	1534
Customize the edit dialog	1536
Perform CRUD operation for complex object using EditTemplate	1538
Paging in Blazor DataGrid Component	1539
Pager with page size dropdown.....	1541
Pager Template	1542
Scrolling in Blazor DataGrid Component	1546
Set width and height.....	1546
Responsive with parent container	1547
Frozen rows and columns	1548
Virtualization in Blazor DataGrid Component	1554
Row Virtualization.....	1555
Column Virtualization	1557
Enable Cell placeholder during Virtualization.....	1560
Frozen Columns Virtualization	1562
Scroll the content by external button.....	1565
Limitations for Virtualization	1568
See Also	1569
Selection in Blazor DataGrid Component	1569
Selection mode	1570

Cell selection	1572
Checkbox selection	1573
Toggle selection	1576
Drag selection	1577
Perform Toggle selection programmatically.....	1579
Select row at initial rendering.....	1580
Get selected row indexes.....	1581
Touch interaction	1582
Multiple selection based on condition	1583
Simple multiple row selection	1585
Aggregates in Blazor DataGrid Component.....	1586
Built-in aggregate types	1586
Footer aggregate.....	1586
How to format aggregate value	1588
Group and caption aggregate	1589
Custom aggregate	1592
Handling Aggregates in Custom Adaptor.....	1593
Page setup.....	1597
Print using an external button	1597
Print the visible page.....	1599
Print large number of columns	1602
Vertical Mode.....	1605
State Management in Blazor DataGrid Component	1607
Enabling persistence in Grid	1607
Handling grid state manually	1608
Globalization in Blazor DataGrid Component.....	1609
Localization	1610
Internationalization.....	1617
Right to left (RTL)	1617
Toolbar in Blazor DataGrid Component.....	1618
Built-in Tool Bar Item	1618
Custom Toolbar Items.....	1620
Built-in and Custom Items in Toolbar	1621
Custom Toolbar.....	1623
Enable/Disable Toolbar Items.....	1627

Customize Toolbar Text	1629
Customize toolbar styles	1630
Pdf Export in Blazor DataGrid Component	1632
Excel Export in Blazor DataGrid Component	1652
To customize excel export	1653
Exporting grouped records	1657
How to export the Grid with specific columns	1659
Customizing columns	1663
Copy to clipboard by external buttons	1665
AutoFill	1666
Paste	1668
Context Menu in Blazor DataGrid Component	1669
Custom context menu items	1671
Built-in and Custom context menu items	1673
Sub context menu items in DataGrid	1674
Disable the Context menu for specific columns in DataGrid	1675
Disable context menu items dynamically in DataGrid	1676
Accessibility in Blazor DataGrid Component	1677
WAI-ARIA	1677
Keyboard navigation	1678
WebAssembly Performance in Blazor DataGrid Component	1679
Avoid unnecessary component renders	1679
Avoid unnecessary component renders after grid events	1681
Use paging or virtualization to load only visible rows	1682
Events in Blazor DataGrid Component	1682
OnActionBegin	1682
OnActionComplete	1683
OnActionFailure	1684
BeforeOpenColumnChooser	1685
Created	1686
OnLoad	1687
Destroyed	1688
OnDataBound	1688
DataBound	1689
RowDataBound	1690

DetailDataBound.....	1691
HeaderCellInfo	1693
QueryCellInfo	1694
OnBeginEdit	1695
OnBatchAdd	1695
OnBatchSave	1696
OnBatchDelete	1697
OnCellEdit.....	1698
OnCellSave	1699
CellSaved	1700
RowSelecting.....	1701
RowSelected.....	1701
RowDeselecting.....	1702
RowDeselected	1703
CellSelecting.....	1704
CellSelected.....	1705
CellDeselecting.....	1706
CellDeselected.....	1706
OnRecordClick.....	1707
OnRecordDoubleClick	1708
OnToolBarClick	1709
CommandClicked	1710
ColumnMenuItemClicked	1711
ContextMenuItemClicked	1712
ContextMenuOpen	1713
OnPdfExport.....	1714
PdfHeaderQueryCellInfoEvent.....	1715
PdfQueryCellInfoEvent.....	1716
PdfAggregateTemplateInfo	1717
OnExcelExport.....	1719
ExcelHeaderQueryCellInfoEvent.....	1720
ExcelQueryCellInfoEvent.....	1721
ExcelAggregateTemplateInfo	1722
ExportComplete	1723
OnResizeStart.....	1724

ResizeStopped.....	1725
OnRowDragStart	1726
RowDropped	1727
How To	1728
Blazor DataGrid Component in WebAssembly App using CLI.....	1728
Blazor DataGrid Component in Server Side App using CLI.....	1734
Blazor DataGrid Component in WebAssembly App using Visual Studio	1740
Show or Hide columns in Dialog editing in Blazor DataGrid Component	1750
Create custom toolbar with drop-down list in Blazor DataGrid Component	1752
Customize Column Styles in Blazor DataGrid Component.....	1754
Access public methods in Blazor DataGrid Component	1756
Change datasource dynamically in Blazor DataGrid Component	1757
Custom control in datagrid toolbar in Blazor DataGrid Component	1759
DataGrid customization in Blazor DataGrid Component.....	1761
Get index value of selected rowcell in Blazor DataGrid Component.....	1763
Select rows based on certain condition in Blazor DataGrid Component.....	1764
Customize column menu icon in Blazor DataGrid Component	1765
How to Group the Column Chooser Items.....	1767
Calculate column value based on other columns in Blazor DataGrid.....	1770
Using dictionary values as datasource in Blazor DataGrid Component.....	1771
Upgrade Application To Latest Version in Blazor DataGrid Component	1774
Custom toolbar items with text name same as default toolbar items	1775
Blazor DataGrid Component inside the Tab with Specific Height	1777
Custom data source & filtering for DropDownList in Blazor DataGrid	1778
Single click editing with Batch mode in Blazor DataGrid Component	1780
Cascading DropDownList in Blazor DataGrid Component Editing	1781
Editing with template column in Blazor DataGrid Component	1783
Hide DataGrid Header in Blazor DataGrid Component.....	1784
Display Custom Tooltip in Blazor DataGrid Cell	1784
Styling and appearance in Blazor DataGrid Component	1786
Customize empty grid display message in Blazor DataGrid Component.....	1787
Saving a new row at a particular index of the blazor datagrid	1788
Filter choice items count for Excel filter in Blazor DataGrid	1789
Custom delete confirmation dialog in Blazor DataGrid Component	1790
Create custom Grid component in Blazor DataGrid Component	1792

Add a range of items into ObservableCollection in Blazor DataGrid	1794
Hide the command column button in a specific record	1795
Hide expand icon in hierarchical Grid when it has no child records.....	1797
DatePicker	1799
Getting Started with Blazor DatePicker Component	1799
Importing Syncfusion Blazor component in the application.....	1799
Adding component package to the application	1800
Add SyncfusionBlazor service in Program.cs	1800
Adding DatePicker component to the application.....	1800
Run the application	1801
Setting the Value and Min and Max dates.....	1801
See Also	1801
Data Binding in Blazor DatePicker Component	1802
One-Way Binding	1802
Two-Way Data Binding.....	1802
Dynamic Value Binding	1802
Date Range in Blazor DatePicker Component.....	1803
Date Format in Blazor DatePicker Component.....	1804
Start and Depth View in Blazor DatePicker Component.....	1805
Start view	1805
Depth view	1806
Globalization in Blazor DatePicker Component.....	1806
Blazor server side	1806
Blazor WebAssembly	1809
Customize the localized text	1810
Right-To-Left	1811
Strict Mode in Blazor DatePicker Component	1812
Native Events in Blazor DatePicker Component	1814
Bind native events to DatePicker	1814
Pass event data to event handler	1814
List of Native events supported	1815
Accessibility in Blazor DatePicker Component	1815
Keyboard interaction	1815
Events in Blazor DatePicker Component	1817
Blur	1817

ValueChange	1817
OnClose	1817
Created.....	1818
Destroyed.....	1818
Focus	1818
Navigated	1819
OnOpen	1819
OnRenderDayCell	1819
Week Number in Blazor DatePicker Component.....	1820
Week Rule	1820
Special Dates in Blazor DatePicker Component.....	1821
How To	1823
Disabled the Blazor DatePicker Component.....	1823
Set the Placeholder in Blazor DatePicker Component.....	1824
Set the Readonly in Blazor DatePicker Component.....	1824
Open the Blazor DatePicker popup on Focus	1825
DateRangePicker	1825
Getting Started with Blazor DateRangePicker Component	1825
Importing Syncfusion Blazor component in the application.....	1826
Adding component package to the application	1826
Add SyncfusionBlazor service in Program.cs	1826
Adding DateRangePicker component to the application.....	1827
Run the application	1827
Setting the Min and Max.....	1827
See Also	1828
Data Binding in Blazor DateRangePicker Component	1828
One-Way Binding	1828
Two-Way Data Binding.....	1829
Dynamic Value Binding	1829
Range Restriction in Blazor DateRangePicker Component.....	1830
Restrict the range within a range.....	1830
Range span.....	1831
Strict mode.....	1831
Globalization in Blazor DateRangePicker Component.....	1833
Blazor server side	1833

Blazor WebAssembly	1836
Customize the localized text	1837
Right-To-Left	1838
Native Events in Blazor DateRangePicker Component	1839
Bind native events to DateRangePicker	1839
Pass event data to event handler	1840
List of Native events supported	1840
Customization in Blazor DateRangePicker Component	1840
First day of week	1841
Accessibility in Blazor DateRangePicker Component	1841
Keyboard interaction	1842
Events in Blazor DateRangePicker Component	1843
Blur	1843
ValueChange	1844
OnClose	1844
Created.....	1844
Destroyed.....	1844
Focus	1845
Navigated	1845
OnOpen	1845
OnRenderDayCell	1846
RangeSelected.....	1846
Week Number in Blazor DateRangePicker Component.....	1846
Week Rule	1847
How To	1848
Disable the Blazor DateRangePicker Component.....	1848
Set the Placeholder in Blazor DateRangePicker Component.....	1849
Customization using CssClass in Blazor DateRangePicker Component	1849
Datetime Picker.....	1851
Getting Started with Blazor DateTime Picker Component	1851
Importing Syncfusion Blazor component in the application.....	1851
Adding component package to the application.....	1852
Add SyncfusionBlazor service in Program.cs	1852
Adding DateTimePicker component to the application.....	1853
Run the application	1853

Setting the Value, Min and Max	1853
See Also	1854
Data Binding in Blazor Datetime Picker Component	1854
One-Way Binding	1854
Two-Way Data Binding.....	1855
Dynamic Value Binding	1855
DateTime Range in Blazor Datetime Picker Component	1855
Globalization in Blazor Datetime Picker Component.....	1857
Blazor server side	1857
Blazor WebAssembly	1859
Customize the localized text	1861
Right-To-Left	1861
Native Events in Blazor Datetime Picker Component.....	1862
Bind native events to DateTimePicker.....	1862
Pass event data to event handler	1863
List of Native events supported	1863
Strict Mode in Blazor Datetime Picker Component.....	1863
Accessibility in Blazor Datetime Picker Component	1865
Keyboard Interaction	1865
Events in Blazor Datetime Picker Component	1867
Blur	1867
ValueChange	1868
OnClose	1868
Created.....	1868
Destroyed.....	1869
Focus	1869
Navigated	1869
OnOpen	1869
OnRenderDayCell	1870
Week Number in Blazor DateTimePicker Component.....	1870
Week Rule	1871
Special Dates in Blazor DateTimePicker Component.....	1872
How To	1874
Disable the Blazor DateTimePicker Component.....	1874
Set the Placeholder in Blazor Datetime Picker Component	1874

Welcome to Syncfusion Blazor Components

Syncfusion Blazor Components is a modern enterprise native UI components library for creating Blazor WebAssembly and Server applications. Syncfusion Blazor components library has been built from the ground up to be lightweight, responsive, modular, and touch-friendly. The Syncfusion Blazor components are native Blazor components and not wrappers over EJ2 components.

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** gives information that is to known before starting to write code. This is the only section that is recommended to be read end-to-end before starting to write code, all other information can be referred to as needed.
- After being familiar with the basics of using the component, the next step would be to start integrating the component into the application. A good starting point would be to refer to the code snippets in the [online sample browser](#) which contains hundreds of code samples, you will likely find a code sample that resembles your intended usage scenario.

Components List

<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;
}
.controlcategory
{
font-size: 14px!important;
text-align: left!important;
font-weight: bold!important;
border:0 !important;
}
</style>

```

| | | | |
|---------------------------------------|-----------------------------------|--------------------------------------|--------------------------------------|
| GRIDS | DATA
VISUALIZATION | CALENDARS | DROPDOWNS |
| DataGrid | Charts | Scheduler | AutoComplete |
| Pivot Table | Stock Chart | Gantt Chart | ListBox |
| TreeGrid | Circular Gauge | Calendar | ComboBox |
| FILE VIEWERS &
EDITORS | Linear Gauge | DatePicker | Dropdown List |
| RichTextEditor | Diagram Component | DateRangePicker | Multiselect DropDown |
| PDF Viewer | HeatMap Chart | DateTime Picker | NAVIGATION |
| Word Processor | Map | TimePicker | Accordion |
| FILE FORMAT
FRAMEWORKS | Range Selector | INPUTS | Context Menu |
| Excel | Smith Chart | TextBox | Menu Bar |
| PDF | Sparkline Charts | Input Mask | Sidebar |
| Word | Barcode | Numeric TextBox | Tabs |
| PowerPoint | TreeMap | RadioButton | Toolbar |
| LAYOUT | Bullet Chart | CheckBox | TreeView |
| Dialog | Kanban | Color Picker | File Manager |
| ListView | BUTTONS | File Upload | Breadcrumb |
| Tooltip | Button | Range Slider | NOTIFICATION |
| Splitter | ButtonGroup | Toggle Switch Button | Toast |
| | Dropdown Menu | Signature [Preview] | Progress Bar |
| | Progress Button | FORMS | Spinner |
| | SplitButton | In-place Editor | Badge |

| | | | |
|---------------------------|-----------------------|-------------------------------|--|
| Dashboard | Chips | Query Builder | |
| Card | Icons | | |
| Avatar | | | |

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 the tag `syncfusion-blazor`.
- Don't see what you need? Please request it in our [feedback portal](#).

See Also

- [Product Development Life Cycle](#)

Visual Studio

- [Getting started with Syncfusion Blazor Components in Server App](#)
- [Getting started with Syncfusion Blazor Components in WebAssembly App](#)

.NET CLI

- [Getting started with Syncfusion Blazor Components in Server App](#)
- [Getting started with Syncfusion Blazor Components in WebAssembly App](#)

Visual Studio for Mac

- [Getting started with Syncfusion Blazor Components in Server App](#)
- [Getting started with Syncfusion Blazor Components in WebAssembly App](#)

System requirements for Blazor Components

Blazor applications can be developed using one of the following IDEs or using [.NET CLI](#).

Integrated Development Environment

Blazor applications can be developed using one of the following IDEs. You can also develop using [.NET CLI](#) without below IDEs.

- [Visual Studio 2022 / Visual Studio 2019](#)
- [Visual Studio Code](#)
- [JetBrains Rider](#)

Framework & SDK

One of the the following .NET SDK is required to develop and run the Blazor application.

- [.NET 6.0 SDK](#)
- [.NET 5.0 SDK](#)
- [.NET Core SDK 3.1.8](#)

If you are planning to use Visual Studio to develop Blazor Applications.

.NET Core SDK 3.1.8 requires Visual Studio 2019 16.7 or later.

.NET 5.0 requires Visual Studio 2019 16.8 or later.

.NET 6.0 requires Visual Studio 2022 17.0 Preview 4.1 or later.

Browser Compatibility in Blazor

Syncfusion Blazor UI components are supported by all modern web browsers on Windows, Linux, and MacOS.

Blazor WebAssembly

| Browser | Versions |
|--|----------------|
| ----- ----- | |
| Google Chrome, including Android & iOS | Latest Version |
| Mozilla Firefox | Latest Version |
| Microsoft Edge | Latest Version |
| Apple Safari, including iOS | Latest Version |
| Opera | Latest Version |
| Microsoft Internet Explorer | Not Supported |

Blazor Server Side

| Browser | Versions |
|--|----------------|
| ----- ----- | |
| Google Chrome, including Android & iOS | Latest Version |
| Mozilla Firefox | Latest Version |
| Microsoft Edge | Latest Version |
| Apple Safari, including iOS | Latest Version |
| Opera | Latest Version |
| Microsoft Internet Explorer | 11 |

Microsoft Internet Explorer requires additional polyfills for [.NET Core 3.1 / lower versions](#). Refer this [documentation](#) to add the polyfills in Blazor server application.

See Also

- [ASP.NET Core Blazor supported platforms](#)

NuGet Packages for Syncfusion Blazor UI components

Starting with v18.4.0.30 (Volume 4, 2020), the Syncfusion Blazor UI components are separately available in individual NuGet packages. The NuGet packages are segregated based on the component usage and its namespace. The complete NuGet package [Syncfusion.Blazor](#) will also be available along with the individual NuGet packages. It means its support is not deprecated yet.

Warning: Do not use both [Syncfusion.Blazor](#) and individual NuGet packages in the same application. It will throw ambiguous errors while compiling the project.

Benefits of using individual NuGet packages

- These individual NuGet packages are extremely useful while rendering Syncfusion Blazor components in Blazor WebAssembly applications. These packages will reduce the initial loading time in Blazor WebAssembly applications.
- While installing [Syncfusion.Blazor](#) NuGet package in a Blazor WebAssembly application, it will load the complete Syncfusion Blazor library in the web browser which takes more initial loading time. Whereas, the individual NuGet package installation will resolve this and load the required components assembly alone in the web browser.
- The [Lazy load assemblies in Blazor WebAssembly](#) functionality can be utilized with the Syncfusion Blazor individual NuGet packages.
- These individual NuGet packages can be used in the Blazor Server application to reduce the application deployment size in production.

Available NuGet packages

[Syncfusion.Blazor.Core](#)

This package contains the base component, common classes, common functionalities, and interfaces for the entire Syncfusion Blazor UI components.

<!-- markdownlint-disable MD033 -->

| NuGet package name | Assembly name | Components | Dependencies |
|--|----------------------------|-----------------|--|
| Syncfusion.Blazor.Core | Syncfusion.Blazor.Core.dll | SfBaseComponent | <ul style="list-style-type: none"> • Microsoft.AspNetCore.Components.Web • Microsoft.CSharp • Newtonsoft.Json • Syncfusion.Blazor.Themes • Syncfusion.Licensing |

[Syncfusion.Blazor.BarcodeGenerator](#)

The Blazor BarcodeGenerator supports the most common 1D and 2D barcode, and complete customization of its appearance.

| NuGet package name | Assembly name | Components | Dependencies |
|--------------------|---------------|------------|--------------|
|--------------------|---------------|------------|--------------|

| | | | |
|--|--|--|--|
| Syncfusion.Blazor.BarcodeGenerator | Syncfusion.Blazor.BarcodeGenerator.dll | <ul style="list-style-type: none"> SfBarcodeGenerator SfDataMatrixGenerator SfQRCodeGenerator | Syncfusion.Blazor.Core |
|--|--|--|--|

[Syncfusion.Blazor.BulletChart](#)

The Blazor Bullet Chart is used to visually compare measures, similar to the commonly used bar chart. A bullet chart displays one or more measures, and compares them with a target value. The measures can be displayed in a range of performance such as poor, satisfactory, and good.

| NuGet package name | Assembly name | Components | Dependencies |
|---|-----------------------------------|---------------|---|
| Syncfusion.Blazor.BulletChart | Syncfusion.Blazor.BulletChart.dll | SfBulletChart | <ul style="list-style-type: none"> Syncfusion.Blazor.Core Syncfusion.Blazor.Data Syncfusion.Blazor.DataVizCommon |

[Syncfusion.Blazor.Buttons](#)

The Blazor buttons package contains UI components such as Button, Checkbox, RadioButton, Switch, and Chip component. It is easy to use and integrate within the form.

| NuGet package name | Assembly name | Components | Dependencies |
|---|-------------------------------|---|--|
| Syncfusion.Blazor.Buttons | Syncfusion.Blazor.Buttons.dll | <ul style="list-style-type: none"> SfButton SfCheckBox SfChip SfRadioButton SfSwitch SfIcon | Syncfusion.Blazor.Core |

[Syncfusion.Blazor.Calendars](#)

The Calendars package contains date and time components such as Calendar, DatePicker, DateRangePicker, DateTimePicker, and TimePicker. These components come with options to disable dates, restrict selection, and show custom events.

| NuGet package name | Assembly name | Components | Dependencies |
|---|---------------------------------|---|---|
| Syncfusion.Blazor.Calendars | Syncfusion.Blazor.Calendars.dll | <ul style="list-style-type: none"> SfCalendar SfDatePicker SfDateRangePicker | <ul style="list-style-type: none"> Syncfusion.Blazor.Core Syncfusion.Blazor.Buttons |

| | | | |
|--|--|--|---|
| | | <ul style="list-style-type: none"> SfDateTimePicker SfTimePicker | <ul style="list-style-type: none"> Syncfusion.Blazor.Inputs Syncfusion.Blazor.Lists |
|--|--|--|---|

Syncfusion.Blazor.Cards

A Blazor Card is a small layout that shows a defined content in an organized structure.

| NuGet package name | Assembly name | Components | Dependencies |
|---|-----------------------------|------------|--|
| Syncfusion.Blazor.Cards | Syncfusion.Blazor.Cards.dll | SfCard | Syncfusion.Blazor.Core |

Syncfusion.Blazor.Charts

The Blazor Chart is a well-crafted charting component to visualize data. It contains a rich gallery of 30+ charts and graphs, ranging from line to financial that cater to all charting scenarios. Its high performance helps to render large amounts of data quickly. It also comes with features such as zooming, panning, tooltip, crosshair, trackball, highlight, and selection.

| NuGet package name | Assembly name | Components | Dependencies |
|--|------------------------------|--|--|
| Syncfusion.Blazor.Charts | Syncfusion.Blazor.Charts.dll | <ul style="list-style-type: none"> SfAccumulationChart SfChart | <ul style="list-style-type: none"> Syncfusion.Blazor.Core Syncfusion.Blazor.Data Syncfusion.Blazor.DataVizCommon Syncfusion.PdfExport.Net.Core |

Syncfusion.Blazor.CircularGauge

The Blazor Circular Gauge is used for visualizing numeric values on a circular scale with features like multiple axes, rounded corners, and more. The appearance of the gauge can be completely customized to simulate a speedometer, meter gauge, analog clock, etc.

| NuGet package name | Assembly name | Components | Dependencies |
|---|-------------------------------------|-----------------|---|
| Syncfusion.Blazor.CircularGauge | Syncfusion.Blazor.CircularGauge.dll | SfCircularGauge | <ul style="list-style-type: none"> Syncfusion.Blazor.Core Syncfusion.PdfExport.Net.Core |

Syncfusion.Blazor.Data

The SfDataManager is a data management package to perform data operations such as grouping, sorting in Blazor applications. It will act as an abstraction for using local data sources like IEnumerable, Observable collections, and remote data sources like web services returning JSON, JSONP, OData.

| NuGet package name | Assembly name | Components | Dependencies |
|--|----------------------------|---------------|--|
| Syncfusion.Blazor.Data | Syncfusion.Blazor.Data.dll | SfDataManager | Syncfusion.Blazor.Core |

[Syncfusion.Blazor.DataVizCommon](#)

The Blazor DataVizCommon is the base package for the svg elements used in the visualization components like charts and range selector.

| NuGet package name | Assembly name | Components | Dependencies |
|---|-------------------------------------|------------|--|
| Syncfusion.Blazor.DataVizCommon | Syncfusion.Blazor.DataVizCommon.dll | - | Syncfusion.Blazor.Core |

[Syncfusion.Blazor.Diagrams](#)

The Blazor Diagram is used for visualization, design, and editing of interactive diagrams such as flowcharts, BPMN diagrams, and mind maps. It has seamless interaction and editing capabilities.

| NuGet package name | Assembly name | Components | Dependencies |
|--|--------------------------------|--|---|
| Syncfusion.Blazor.Diagrams | Syncfusion.Blazor.Diagrams.dll | <ul style="list-style-type: none"> SfDiagram SfOverview SfSymbolPalette | <ul style="list-style-type: none"> Syncfusion.Blazor.Core Syncfusion.Blazor.Data Syncfusion.Blazor.Navigations Syncfusion.Blazor.Popups |

[Syncfusion.Blazor.DropDowns](#)

A package of Blazor Dropdown contains a collection of Dropdown components such as Dropdown List, Combo Box, AutoComplete, Multiselect Dropdown, and List Box. Dropdown components contain specific features such as data binding, grouping, sorting, filtering, and templates.

| NuGet package name | Assembly name | Components | Dependencies |
|---|---------------------------------|--|--|
| Syncfusion.Blazor.DropDowns | Syncfusion.Blazor.DropDowns.dll | <ul style="list-style-type: none"> SfAutoComplete SfComboBox SfDropDownList SfListBox SfMultiSelect | <ul style="list-style-type: none"> Syncfusion.Blazor.Core Syncfusion.Blazor.Buttons Syncfusion.Blazor.Data Syncfusion.Blazor.Inputs Syncfusion.Blazor.Spinner |

Syncfusion.Blazor.FileManager

Blazor File Manager is a graphical user interface component used to manage the file system. It enables the user to perform common file operations such as accessing, editing, uploading, downloading, and sorting files and folders. This component also allows easy navigation for browsing or selecting a file or folder from the file system.

| NuGet package name | Assembly name | Components | Dependencies |
|---|-----------------------------------|---------------|---|
| Syncfusion.Blazor.FileManager | Syncfusion.Blazor.FileManager.dll | SfFileManager | <ul style="list-style-type: none"> • Syncfusion.Blazor.Core • Syncfusion.Blazor.Buttons • Syncfusion.Blazor.Data • Syncfusion.Blazor.Grid • Syncfusion.Blazor.Inputs • Syncfusion.Blazor.Layouts • Syncfusion.Blazor.Lists • Syncfusion.Blazor.Navigations • Syncfusion.Blazor.Popups • Syncfusion.Blazor.Spinner • Syncfusion.Blazor.SplitButtons |

Syncfusion.Blazor.Gantt

The Blazor Gantt is designed to visualize and edit the project schedule, and track the project progress. It helps to organize and schedule the projects, and also the project schedule can be updated through interactions like editing, dragging, and resizing.

| NuGet package name | Assembly name | Components | Dependencies |
|---|-----------------------------|------------|---|
| Syncfusion.Blazor.Gantt | Syncfusion.Blazor.Gantt.dll | SfGantt | <ul style="list-style-type: none"> • Syncfusion.Blazor.Core • Syncfusion.Blazor.Buttons • Syncfusion.Blazor.Calendars • Syncfusion.Blazor.Data • Syncfusion.Blazor.DropDowns • Syncfusion.Blazor.FileManager • Syncfusion.Blazor.Grid • Syncfusion.Blazor.Inputs • Syncfusion.Blazor.Layouts • Syncfusion.Blazor.Lists • Syncfusion.Blazor.Navigations • Syncfusion.Blazor.Popups • Syncfusion.Blazor.RichTextEditor |

| | | | |
|--|--|--|---|
| | | | <ul style="list-style-type: none"> • Syncfusion.Blazor.Spinner • Syncfusion.Blazor.TreeGrid |
|--|--|--|---|

Syncfusion.Blazor.Grid

Blazor DataGrid component is used to display and manipulate the tabular data with configuration options to control the way the data is presented. It can pull data from data sources such as IEnumerable, ObservableCollection, OData web services, or DataManager and binding data fields to columns. It also displays the column header to identify the field with support for grouped records.

| NuGet package name | Assembly name | Components | Dependencies |
|--|-----------------------------|------------|---|
| Syncfusion.Blazor.Grid | Syncfusion.Blazor.Grids.dll | SfGrid | <ul style="list-style-type: none"> • Syncfusion.Blazor.Core • Syncfusion.Blazor.Buttons • Syncfusion.Blazor.Calendars • Syncfusion.Blazor.Data • Syncfusion.Blazor.DropDowns • Syncfusion.Blazor.Inputs • Syncfusion.Blazor.Navigations • Syncfusion.Blazor.Popups • Syncfusion.Blazor.Spinner • Syncfusion.ExcelExport.Net.Core • Syncfusion.PdfExport.Net.Core |

Syncfusion.Blazor.HeatMap

Blazor HeatMap Chart is used to visualize two-dimensional data in which the values are represented in gradient or fixed colors.

| NuGet package name | Assembly name | Components | Dependencies |
|---|-------------------------------|------------|--|
| Syncfusion.Blazor.HeatMap | Syncfusion.Blazor.HeatMap.dll | SfHeatMap | <ul style="list-style-type: none"> • Syncfusion.Blazor.Core • Syncfusion.Blazor.Data |

Syncfusion.Blazor.InPlaceEditor

The Blazor In-place Editor component is most useful for editing a value dynamically within its context (in-place). Its features include inline and pop-up modes, and customizable user interface (UI) and events.

| NuGet package name | Assembly name | Components | Dependencies |
|---|-------------------------------------|-----------------|--|
| Syncfusion.Blazor.InPlaceEditor | Syncfusion.Blazor.InPlaceEditor.dll | SfInPlaceEditor | <ul style="list-style-type: none"> • Syncfusion.Blazor.Core • Syncfusion.Blazor.Buttons • Syncfusion.Blazor.DropDowns • Syncfusion.Blazor.Inputs |

| | | | |
|--|--|--|---|
| | | | <ul style="list-style-type: none"> • Syncfusion.Blazor.Popups • Syncfusion.Blazor.Spinner |
|--|--|--|---|

Syncfusion.Blazor.Inputs

A package of Blazor input components comes with a collection of form components. They can be used to get different input values from the users such as text, numbers, patterns, color, and file inputs.

| NuGet package name | Assembly name | Components | Dependencies |
|--|------------------------------|--|---|
| Syncfusion.Blazor.Inputs | Syncfusion.Blazor.Inputs.dll | <ul style="list-style-type: none"> • SfColorPicker • SfMaskedTextBox • SfNumericTextBox • SfSlider • SfTextBox • SfUploader • SfSignature | <ul style="list-style-type: none"> • Syncfusion.Blazor.Core • Syncfusion.Blazor.Data • Syncfusion.Blazor.Popups • Syncfusion.Blazor.Spinner • Syncfusion.Blazor.SplitButtons |

Syncfusion.Blazor.Kanban

The Blazor Kanban board visually depicts work at various stages of a process using columns, cards, and swimlane.

| NuGet package name | Assembly name | Components | Dependencies |
|--|------------------------------|------------|--|
| Syncfusion.Blazor.Kanban | Syncfusion.Blazor.Kanban.dll | SfKanban | <ul style="list-style-type: none"> • Syncfusion.Blazor.Core • Syncfusion.Blazor.Buttons • Syncfusion.Blazor.Data • Syncfusion.Blazor.DropDowns • Syncfusion.Blazor.Inputs • Syncfusion.Blazor.Lists • Syncfusion.Blazor.Popups • Syncfusion.Blazor.Spinner |

Syncfusion.Blazor.Layouts

The layout package contains Splitter and Dashboard Layout components. The Blazor DashboardLayout is a grid structured layout control that helps to create a dashboard with panels. The splitter is a layout component used to construct different layouts using multiple and nested panes that are resizable and expandable.

| NuGet package name | Assembly name | Components | Dependencies |
|---|-------------------------------|---|--|
| Syncfusion.Blazor.Layouts | Syncfusion.Blazor.Layouts.dll | <ul style="list-style-type: none"> SfDashboardLayout SfSplitter | <ul style="list-style-type: none"> Syncfusion.Blazor.Core Syncfusion.Blazor.Data |

[Syncfusion.Blazor.LinearGauge](#)

The Blazor Linear Gauge is used for visualizing numeric values in a linear scale with features like multiple axes, different orientations, and more. The appearance of the gauge can be completely customized to simulate a thermometer, pressure gauge, ruler, etc.

| NuGet package name | Assembly name | Components | Dependencies |
|---|-----------------------------------|---------------|---|
| Syncfusion.Blazor.LinearGauge | Syncfusion.Blazor.LinearGauge.dll | SfLinearGauge | <ul style="list-style-type: none"> Syncfusion.Blazor.Core Syncfusion.PdfExport.Net.Core |

[Syncfusion.Blazor.Lists](#)

Blazor ListView component allows to select an item or multiple items from a list-like interface and represents the data in an interactive hierarchical structure across different layouts or views. Lists are used for displaying data, data navigation, and data entry.

| NuGet package name | Assembly name | Components | Dependencies |
|---|-----------------------------|------------|---|
| Syncfusion.Blazor.Lists | Syncfusion.Blazor.Lists.dll | SfListView | <ul style="list-style-type: none"> Syncfusion.Blazor.Core Syncfusion.Blazor.Buttons Syncfusion.Blazor.Data |

[Syncfusion.Blazor.Maps](#)

The Blazor Maps component is used for rendering maps from GeoJSON data or other map providers like OpenStreetMap, Google Maps, and Bing Maps. Its rich feature set includes markers, labels, bubbles, navigation lines, legends, tooltips, zooming, panning, drill down, and much more.

| NuGet package name | Assembly name | Components | Dependencies |
|--|----------------------------|------------|---|
| Syncfusion.Blazor.Maps | Syncfusion.Blazor.Maps.dll | SfMaps | <ul style="list-style-type: none"> Syncfusion.Blazor.Core Syncfusion.Blazor.Data Syncfusion.PdfExport.Net.Core |

[Syncfusion.Blazor.Navigations](#)

A package of Blazor navigation components such as Accordion, ContextMenu, Tabs, Toolbar, TreeView, and Sidebar.

| NuGet package name | Assembly name | Components | Dependencies |
|--|----------------------------------|---|--|
| Syncfusion.Blazor.Navigation | Syncfusion.Blazor.Navigation.dll | <ul style="list-style-type: none"> SfAccordion SfBreadcrumb SfContextMenu SfMenu SfSidebar SfTab SfToolbar SfTreeView | <ul style="list-style-type: none"> Syncfusion.Blazor.Core Syncfusion.Blazor.Buttons Syncfusion.Blazor.Data Syncfusion.Blazor.Inputs Syncfusion.Blazor.Lists Syncfusion.Blazor.Popups |

[Syncfusion.Blazor.Notifications](#)

The notification component Toast is used to notify status or summary information to the end-users.

| NuGet package name | Assembly name | Components | Dependencies |
|---|-------------------------------------|------------|---|
| Syncfusion.Blazor.Notifications | Syncfusion.Blazor.Notifications.dll | SfToast | <ul style="list-style-type: none"> Syncfusion.Blazor.Core Syncfusion.Blazor.Buttons |

[Syncfusion.Blazor.PdfViewer](#)

The Blazor PDF Viewer supports viewing and reviewing PDF files in web applications and also printing them. The thumbnail, bookmark, hyperlink, and table of contents supports provide easy navigation within and outside the PDF files. The form-filling support provides a platform to fill and print with AcroForms. The PDF files can be reviewed with the available annotation tools.

[For Blazor WebAssembly application](#)

| NuGet package name | Assembly name | Components | Dependencies |
|---|---------------------------------|-------------|--|
| Syncfusion.Blazor.PdfViewer | Syncfusion.Blazor.PdfViewer.dll | SfPdfViewer | <ul style="list-style-type: none"> Syncfusion.Blazor.Core Syncfusion.Blazor.Buttons Syncfusion.Blazor.Data Syncfusion.Blazor.DropDowns Syncfusion.Blazor.Inputs Syncfusion.Blazor.InPlaceEditor Syncfusion.Blazor.Lists Syncfusion.Blazor.Navigation |

| | | | |
|--|--|--|---|
| | | | <ul style="list-style-type: none"> • Syncfusion.Blazor.Notifications • Syncfusion.Blazor.Popups • Syncfusion.Blazor.SplitButtons |
|--|--|--|---|

For Blazor Server application

| NuGet package name | Assembly name | Components | Dependencies |
|---|---------------------------------------|-------------------|---|
| Syncfusion.Blazor.PdfViewerServer.Windows | Syncfusion.Blazor.PdfViewerServer.dll | SfPdfViewerServer | <ul style="list-style-type: none"> • Syncfusion.Blazor.PdfViewer |

For developing Linux or Mac (OSX) operating system, use the following corresponding libraries:

* For Linux, use [Syncfusion.Blazor.PdfViewerServer.Linux](#)

* For Mac (OSX), use [Syncfusion.Blazor.PdfViewerServer.OSX](#)

Syncfusion.Blazor.PivotTable

The Blazor Pivot Table is a powerful control used to organize and summarize business data and display the result in a cross-table format. It includes major functionalities such as data binding, drilling up and down, Excel-like filtering and sorting, editing, Excel and PDF exporting, several built-in aggregations, pivot table field list, and calculated fields.

| NuGet package name | Assembly name | Components | Dependencies |
|--|----------------------------------|---|--|
| Syncfusion.Blazor.PivotTable | Syncfusion.Blazor.PivotTable.dll | <ul style="list-style-type: none"> • SfPivotFieldList • SfPivotView | <ul style="list-style-type: none"> • Syncfusion.Blazor.Core • Syncfusion.Blazor.Buttons • Syncfusion.Blazor.Charts • Syncfusion.Blazor.Data • Syncfusion.Blazor.DropDowns • Syncfusion.Blazor.Grid • Syncfusion.Blazor.Inputs • Syncfusion.Blazor.Navigations • Syncfusion.Blazor.Popups • Syncfusion.Blazor.Spinner • Syncfusion.Blazor.SplitButtons • Syncfusion.ExcelExport.Net.Core • Syncfusion.PdfExport.Net.Core |

Syncfusion.Blazor.Popups

A package of Blazor popup components Dialog and Tooltip are used to display information or to get input from the users in a popup.

| NuGet package name | Assembly name | Components | Dependencies |
|--|------------------------------|---|---|
| Syncfusion.Blazor.Popups | Syncfusion.Blazor.Popups.dll | <ul style="list-style-type: none"> SfDialog SfTooltip | <ul style="list-style-type: none"> Syncfusion.Blazor.Core Syncfusion.Blazor.Buttons |

Syncfusion.Blazor.ProgressBar

The Progress Bar control can be used to visualize the changing status of an extended operation such as a download, file transfer, or installation. All the progress bar elements are rendered using scalable vector graphics (SVG) to ensure the quality of the visual experience.

| NuGet package name | Assembly name | Components | Dependencies |
|---|-----------------------------------|---------------|--|
| Syncfusion.Blazor.ProgressBar | Syncfusion.Blazor.ProgressBar.dll | SfProgressBar | <ul style="list-style-type: none"> Syncfusion.Blazor.Core Syncfusion.Blazor.Data |

Syncfusion.Blazor.QueryBuilder

The Blazor QueryBuilder package contains the QueryBuilder component that allows the users to create and edit filters. It supports data binding, templates, validation, and horizontal and vertical orientation.

| NuGet package name | Assembly name | Components | Dependencies |
|--|------------------------------------|----------------|---|
| Syncfusion.Blazor.QueryBuilder | Syncfusion.Blazor.QueryBuilder.dll | SfQueryBuilder | <ul style="list-style-type: none"> Syncfusion.Blazor.Core Syncfusion.Blazor.Buttons Syncfusion.Blazor.Calendars Syncfusion.Blazor.Data Syncfusion.Blazor.DropDowns Syncfusion.Blazor.Inputs Syncfusion.Blazor.Popups Syncfusion.Blazor.SplitButtons |

Syncfusion.Blazor.RangeNavigator

The Blazor Range Navigator is an interface for selecting a small range from a larger collection. It is commonly used in financial dashboards to filter a date range for data that needs to be visualized.

| NuGet package name | Assembly name | Components | Dependencies |
|--|--------------------------------------|------------------|--|
| Syncfusion.Blazor.RangeNavigator | Syncfusion.Blazor.RangeNavigator.dll | SfRangeNavigator | <ul style="list-style-type: none"> • Syncfusion.Blazor.Core • Syncfusion.Blazor.Calendars • Syncfusion.Blazor.Charts • Syncfusion.Blazor.Data • Syncfusion.Blazor.DataVizCommon • Syncfusion.Blazor.Navigations • Syncfusion.PdfExport.Net.Core |

Syncfusion.Blazor.RichTextEditor

The Rich Text Editor component is the HTML and markdown editor that provides the best user experience for creating, updating, and formatting the content.

| NuGet package name | Assembly name | Components | Dependencies |
|--|--------------------------------------|------------------|--|
| Syncfusion.Blazor.RichTextEditor | Syncfusion.Blazor.RichTextEditor.dll | SfRichTextEditor | <ul style="list-style-type: none"> • Syncfusion.Blazor.Core • Syncfusion.Blazor.Buttons • Syncfusion.Blazor.Inputs • Syncfusion.Blazor.Navigations • Syncfusion.Blazor.Popups • Syncfusion.Blazor.SplitButtons |

Syncfusion.Blazor.Schedule

The Blazor Scheduler component is an event calendar that facilitates users with the common Outlook-calendar features, thus allowing them to plan and manage their events/appointments and their time in an efficient way.

| NuGet package name | Assembly name | Components | Dependencies |
|--|--------------------------------|--|---|
| Syncfusion.Blazor.Schedule | Syncfusion.Blazor.Schedule.dll | <ul style="list-style-type: none"> • SfRecurrenceEditor • SfSchedule | <ul style="list-style-type: none"> • Syncfusion.Blazor.Core • Syncfusion.Blazor.Buttons |

| | | | |
|--|--|--|---|
| | | | <ul style="list-style-type: none"> • Syncfusion.Blazor.Calendars • Syncfusion.Blazor.Data • Syncfusion.Blazor.DropDowns • Syncfusion.Blazor.Inputs • Syncfusion.Blazor.Navigations • Syncfusion.Blazor.Popups • Syncfusion.Blazor.Spinner • Syncfusion.ExcelExport.Net.Core |
|--|--|--|---|

[Syncfusion.Blazor.SmithChart](#)

The Blazor Smith Chart is a control for showing the parameters of transmission lines in high-frequency circuit applications. Its rich feature set includes features like legends, markers, tooltips, and data labels.

| NuGet package name | Assembly name | Components | Dependencies |
|--|----------------------------------|--------------|--|
| Syncfusion.Blazor.SmithChart | Syncfusion.Blazor.SmithChart.dll | SfSmithChart | <ul style="list-style-type: none"> • Syncfusion.Blazor.Core • Syncfusion.Blazor.Data |

[Syncfusion.Blazor.Sparkline](#)

The Blazor Sparkline Charts is a replacement for normal charts to display trends in a very small area. Customize sparklines completely by changing the series or axis type and by adding markers, data labels, range bands, and more.

| NuGet package name | Assembly name | Components | Dependencies |
|---|---------------------------------|-------------|--|
| Syncfusion.Blazor.Sparkline | Syncfusion.Blazor.Sparkline.dll | SfSparkline | <ul style="list-style-type: none"> • Syncfusion.Blazor.Core • Syncfusion.Blazor.Data |

[Syncfusion.Blazor.Spinner](#)

The Blazor Spinner is a loading indicator that denotes long-running tasks with no information about their progress. The component provides circular progress indicators without any interaction capabilities.

| NuGet package name | Assembly name | Components | Dependencies |
|---|-------------------------------|------------|--|
| Syncfusion.Blazor.Spinner | Syncfusion.Blazor.Spinner.dll | SfSpinner | Syncfusion.Blazor.Core |

Syncfusion.Blazor.SplitButtons

The Blazor SplitButtons package contains UI components such as DropDownButton, SplitButton, ProgressButton, and ButtonGroup components. DropDownButton and SplitButton component display a list of items when a button is clicked and the ButtonGroup can be used for easy navigation.

| NuGet package name | Assembly name | Components | Dependencies |
|--|------------------------------------|--|--|
| Syncfusion.Blazor.SplitButtons | Syncfusion.Blazor.SplitButtons.dll | <ul style="list-style-type: none"> SfButtonGroup SfDropDownButton SfProgressButton SfSplitButton | <ul style="list-style-type: none"> Syncfusion.Blazor.Core Syncfusion.Blazor.Buttons Syncfusion.Blazor.Popups Syncfusion.Blazor.Spinner |

Syncfusion.Blazor.StockChart

The Blazor Stock Chart is an easy-to-use financial charting package to track and visualize the stock price of any company over a specific period using charting and range tools. It also comes with a lot of features such as zooming, panning, tooltip, crosshair, trackball, period selector, range selector, and events to make the stock charts more interactive.

| NuGet package name | Assembly name | Components | Dependencies |
|--|----------------------------------|--------------|--|
| Syncfusion.Blazor.StockChart | Syncfusion.Blazor.StockChart.dll | SfStockChart | <ul style="list-style-type: none"> Syncfusion.Blazor.Core Syncfusion.Blazor.Calendars Syncfusion.Blazor.Charts Syncfusion.Blazor.Navigations Syncfusion.Blazor.SplitButtons |

Syncfusion.Blazor.Themes

This package contains the Syncfusion Blazor UI components theme files.

| NuGet package name | Assembly name | Themes | Dependencies |
|--|------------------------------|---|--------------|
| Syncfusion.Blazor.Themes | Syncfusion.Blazor.Themes.dll | <ul style="list-style-type: none"> Material Material Dark Fabric Fabric Dark Bootstrap Bootstrap Dark Bootstrap v4 | None |

| | | | |
|--|--|---|--|
| | | <ul style="list-style-type: none"> High-Contrast | |
|--|--|---|--|

[Syncfusion.Blazor.TreeGrid](#)

Blazor Tree Grid is a feature-rich control used to visualize self-referential and hierarchical data effectively in a tabular format. It can pull data from data sources such as an enumerable collection of records, RESTful services, OData services, WCF services, or DataManager. It also expands or collapses child data using the tree column.

| NuGet package name | Assembly name | Components | Dependencies |
|--|--------------------------------|------------|---|
| Syncfusion.Blazor.TreeGrid | Syncfusion.Blazor.TreeGrid.dll | SfTreeGrid | <ul style="list-style-type: none"> Syncfusion.Blazor.Core Syncfusion.Blazor.Data Syncfusion.Blazor.Grid Syncfusion.Blazor.Navigation Syncfusion.Blazor.Popups Syncfusion.Blazor.Spinner |

[Syncfusion.Blazor.TreeMap](#)

Blazor TreeMap is a feature-rich component used to visualize both hierarchical and flat data. The look and feel of the treemaps can be customized by using the built-in features like color mapping, legends, and label templates.

| NuGet package name | Assembly name | Components | Dependencies |
|---|-------------------------------|------------|---|
| Syncfusion.Blazor.TreeMap | Syncfusion.Blazor.TreeMap.dll | SfTreeMap | <ul style="list-style-type: none"> Syncfusion.Blazor.Core Syncfusion.Blazor.Data Syncfusion.PdfExport.Net.Core |

[Syncfusion.Blazor.WordProcessor](#)

The Blazor Word Processor (Document Editor) is a component with editing capabilities like Microsoft Word. It is used to create, edit, view, and print Word documents. It provides all the common Word processing features including editing text, formatting contents, resizing images and tables, finding and replacing text, bookmarks, tables of contents, printing, and importing and exporting Word documents.

| NuGet package name | Assembly name | Components | Dependencies |
|---|--------------------------------------|---|--|
| Syncfusion.Blazor.WordProcessor | Syncfusion.Blazor.DocumentEditor.dll | <ul style="list-style-type: none"> SfDocumentEditor SfDocumentEditorContainer | <ul style="list-style-type: none"> Newtonsoft.Json Syncfusion.Blazor.Core Syncfusion.Blazor.Calendars |

| | | | |
|--|--|--|---|
| | | | <ul style="list-style-type: none"> • Syncfusion.Blazor.Chart • Syncfusion.Blazor.Data • Syncfusion.Blazor.DropDowns • Syncfusion.Blazor.Navigations • Syncfusion.WordProcessor.AspNet.Core |
|--|--|--|---|

<!-- markdownlint-enable MD033 -->

Getting Started

Getting Started with Blazor Server Side App in Visual Studio

This article provides a step-by-step instructions for building Blazor Server App with **Blazor Calendar** component using [Visual Studio](#).

Prerequisites

- [System requirements for Blazor components](#)

Create a new Blazor Server App in Visual Studio

You can create **Blazor Server App** using Visual Studio in one of the following ways,

- [Create a Project using Microsoft Templates](#)
- [Create a Project using Syncfusion Blazor Extension](#)

Install Syncfusion Blazor Packages in the App

Syncfusion Blazor components are available in nuget.org. In order to use Syncfusion Blazor components in the application, add reference to the corresponding NuGet. Refer to [NuGet packages topic](#) for available NuGet packages list with component details.

To add Blazor Calendar component in the app, open the NuGet package manager in Visual Studio (*Tools* → *NuGet Package Manager* → *Manage NuGet Packages for Solution*), search for [Syncfusion.Blazor.Calendars](#) and then install it.

Add Style Sheet

Checkout the [Blazor Themes topic](#) to learn different ways to refer themes in Blazor application, and to have the expected appearance for Syncfusion Blazor components. Here, the theme is referred using [Static Web Assets](#).

To add theme to the app, open the NuGet package manager in Visual Studio (*Tools* → *NuGet Package Manager* → *Manage NuGet Packages for Solution*), search for [Syncfusion.Blazor.Themes](#) and then install it. Then, the theme style sheet from NuGet can be referred as follows,

- For **.NET 6** app, add the Syncfusion bootstrap5 theme in the `element` of the `~/Pages/_Layout.cshtml` page.

ASPX-CS

```
<head>
....
<link href="_content/Syncfusion.Blazor.Themes/bootstrap5.css"
rel="stylesheet" />
</head>
```

- For .NET 5 and .NET 3.X app, add the Syncfusion bootstrap5 theme in the element of the ~/Pages/_Host.cshtml page.

ASPX-CS

```
<head>
....
<link href="_content/Syncfusion.Blazor.Themes/bootstrap5.css"
rel="stylesheet" />
</head>
```

Add Script Reference

Checkout [Adding Script Reference topic](#) to learn different ways to add script reference in Blazor Application. In this getting started walk-through, the required scripts are referenced automatically via javascript script isolation approach.

Syncfusion recommends to reference scripts using [Static Web Assets](#), [CDN](#) and [CRG](#) by [disabling JavaScript isolation](#) for better loading performance of the Blazor application.

Register Syncfusion Blazor Service

- Open ~/_Imports.razor file and import the Syncfusion.Blazor namespace.

C#

```
@using Syncfusion.Blazor
```

- Now, register the Syncfusion Blazor Service in the Blazor Server App.
- For .NET 6 app, open the ~/Program.cs file and register the Syncfusion Blazor Service.

C#

```
using Microsoft.AspNetCore.Components;
using Microsoft.AspNetCore.Components.Web;
using Syncfusion.Blazor;
var builder = WebApplication.CreateBuilder(args);
// Add services to the container.
builder.Services.AddRazorPages();
builder.Services.AddServerSideBlazor();
builder.Services.AddSyncfusionBlazor();
var app = builder.Build();
....
```


- For .NET 5 and .NET 3.X app, open the ~/Startup.cs file and register the Syncfusion Blazor Service.

C#

```
using Syncfusion.Blazor;  
namespace BlazorApplication  
{  
    public class Startup  
    {  
        ...  
        public void ConfigureServices(IServiceCollection services)  
        {  
            services.AddRazorPages();  
            services.AddServerSideBlazor();  
            services.AddSyncfusionBlazor();  
        }  
        ...  
    }  
}
```

Add Syncfusion Blazor component

- Open ~/_Imports.razor file or any razor page under the ~/Pages folder where the component is to be added and import the Syncfusion.Blazor.Calendars namespace.

C#

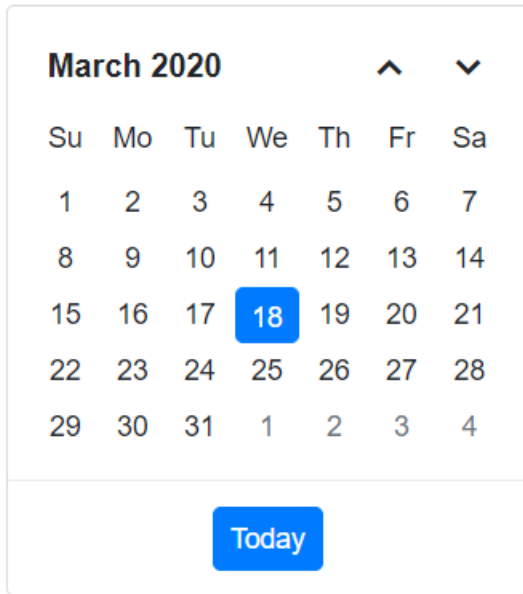
```
@using Syncfusion.Blazor  
@using Syncfusion.Blazor.Calendars
```

- Now, add the Syncfusion Calendar component in razor file. Here, the Calendar component is added in the ~/Pages/Index.razor page under the ~/Pages folder.

C#

```
<SfCalendar TValue="DateTime"/>
```

- Press Ctrl+F5 (Windows) or ⌘+F5 (macOS) to run the app. Then, the Syncfusion Blazor Calendar component will be rendered in the default web browser.



You need to include a valid license key (either paid or trial key) within your applications. Please refer to this [help topic](#) for more information.

Getting Started with Blazor WebAssembly App in Visual Studio

This article provides a step-by-step instructions for building Blazor WebAssembly App with **Blazor Calendar** component using [Visual Studio](#).

Prerequisites

- [System requirements for Blazor components](#)

Create a new Blazor WebAssembly App in Visual Studio

You can create **Blazor WebAssembly App** using Visual Studio in one of the following ways,

- [Create a Project using Microsoft Templates](#)
- [Create a Project using Syncfusion Blazor Extension](#)

Install Syncfusion Blazor Packages in the App

Syncfusion Blazor components are available in [nuget.org](#). In order to use Syncfusion Blazor components in the application, add reference to the corresponding NuGet. Refer to [NuGet packages topic](#) for available NuGet packages list with component details.

To add Blazor Calendar component in the app, open the NuGet package manager in Visual Studio (*Tools* → *NuGet Package Manager* → *Manage NuGet Packages for Solution*), search for [Syncfusion.Blazor.Calendars](#) and then install it.

Add Style Sheet

Checkout the [Blazor Themes topic](#) to learn different ways to refer themes in Blazor application, and to have the expected appearance for Syncfusion Blazor components. Here, the theme is referred using [Static Web Assets](#).

To add theme to the app, open the NuGet package manager in Visual Studio (*Tools* → *NuGet Package Manager* → *Manage NuGet Packages for Solution*), search for [Syncfusion.Blazor.Themes](#) and then install it. Then, the theme style sheet from NuGet can be referred inside the `<head>` element of `wwwroot/index.html` file of client web app.

HTML

```
<head>
...
<link href="_content/Syncfusion.Blazor.Themes/bootstrap5.css"
rel="stylesheet" />
</head>
```

Add Script Reference

Checkout [Adding Script Reference topic](#) to learn different ways to add script reference in Blazor Application. In this getting started walk-through, the required scripts are referenced automatically via javascript script isolation approach.

Syncfusion recommends to reference scripts using [Static Web Assets](#), [CDN](#) and [CRG](#) by [disabling JavaScript isolation](#) for better loading performance of the Blazor application.

Register Syncfusion Blazor Service

- Open `~/_Imports.razor` file and import the `Syncfusion.Blazor` namespace.

C#

```
@using Syncfusion.Blazor
```

- Now, Open `~/Program.cs` file and register the Syncfusion Blazor Service in the client web app.
- For .NET 6 app,

C# HL_LINES="12"

HTML

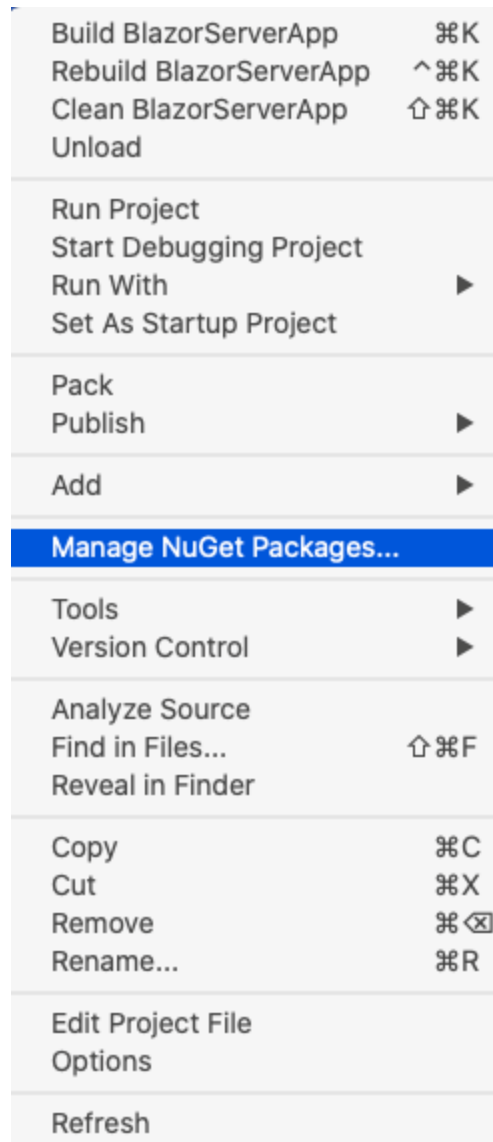
```
<head>
....
....
<link href="_content/Syncfusion.Blazor.Themes/bootstrap4.css"
rel="stylesheet" />
</head>
```

Warning: `Syncfusion.Blazor` package should not be installed along with [individual NuGet packages](#). Hence, you have to add the above `Syncfusion.Blazor.Themes` static web assets (styles) in the application.

Using Syncfusion.Blazor NuGet Package [Old standard]

Warning: If you prefer the above new standard (individual NuGet packages), then skip this section. Using both old and new standards in the same application will throw ambiguous compilation errors.

1. Now, install **Syncfusion.Blazor** NuGet package to the newly created application by using the NuGet Package Manager. Right-click the project, and then select Manage NuGet Packages.



2. Search **Syncfusion.Blazor** keyword in the Browse tab and install **Syncfusion.Blazor** NuGet package in the application.



3. The Syncfusion Blazor package will be installed in the project once the installation process is completed.
4. Add the Syncfusion bootstrap4 theme in the element of the `~/Pages/_Host.html` page.

HTML

```
<head>
...
<link href="_content/Syncfusion.Blazor.Themes/bootstrap4.css"
rel="stylesheet" />
</head>
```

The same theme file can be referred through the CDN version by using

<https://cdn.syncfusion.com/blazor/{site.blazorversion}/styles/bootstrap4.css>.

Adding Syncfusion Blazor component and running the application

1. Open `~/_Imports.razor` file and import the `Syncfusion.Blazor`.

ASPX-CS

```
@using Syncfusion.Blazor
@using Syncfusion.Blazor.Calendars
```

2. Open the `~/Startup.cs` file and register the Syncfusion Blazor Service.

C#

```
using Syncfusion.Blazor;
namespace WebApplication1
{
    public class Startup
    {
        public void ConfigureServices(IServiceCollection services)
        {
            ...
            services.AddSyncfusionBlazor();
        }
    }
}
```

```
}
}
}
```

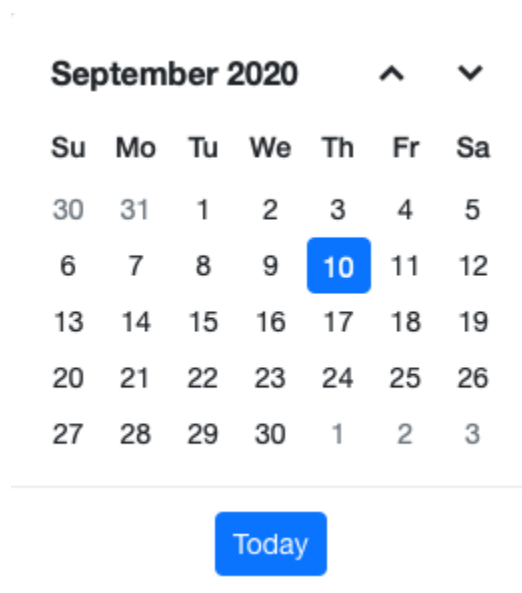
We can disable the dynamic script loading and refer to the scripts from the application end by using the `IgnoreScriptIsolation` parameter in `AddSyncfusionBlazor()` at the `program.cs`. For more details, please refer here for [how to refer custom/CDN resources](#).

- Now, add the Syncfusion Blazor components in any `.razor` file in the `~/Pages` folder. For example, the Calendar component is added in the `~/Pages/Index.razor` page.

ASPX-CS

```
<SfCalendar TValue="DateTime"></SfCalendar>
```

- Run the application, the Syncfusion Blazor Calendar component will be rendered in the default web browser.



<!-- markdownlint-disable MD024 -->

Getting started with Blazor WebAssembly App in VS for Mac

This article provides a step-by-step introduction to configure Syncfusion Blazor components setup, and also build and run a simple Blazor WebAssembly application using [Visual Studio for Mac](#).

Starting with version 17.4.0.39 (2019 Volume 4), you need to include a valid license key (either paid or trial key) within your applications. Please refer to this [help topic](#) for more information.

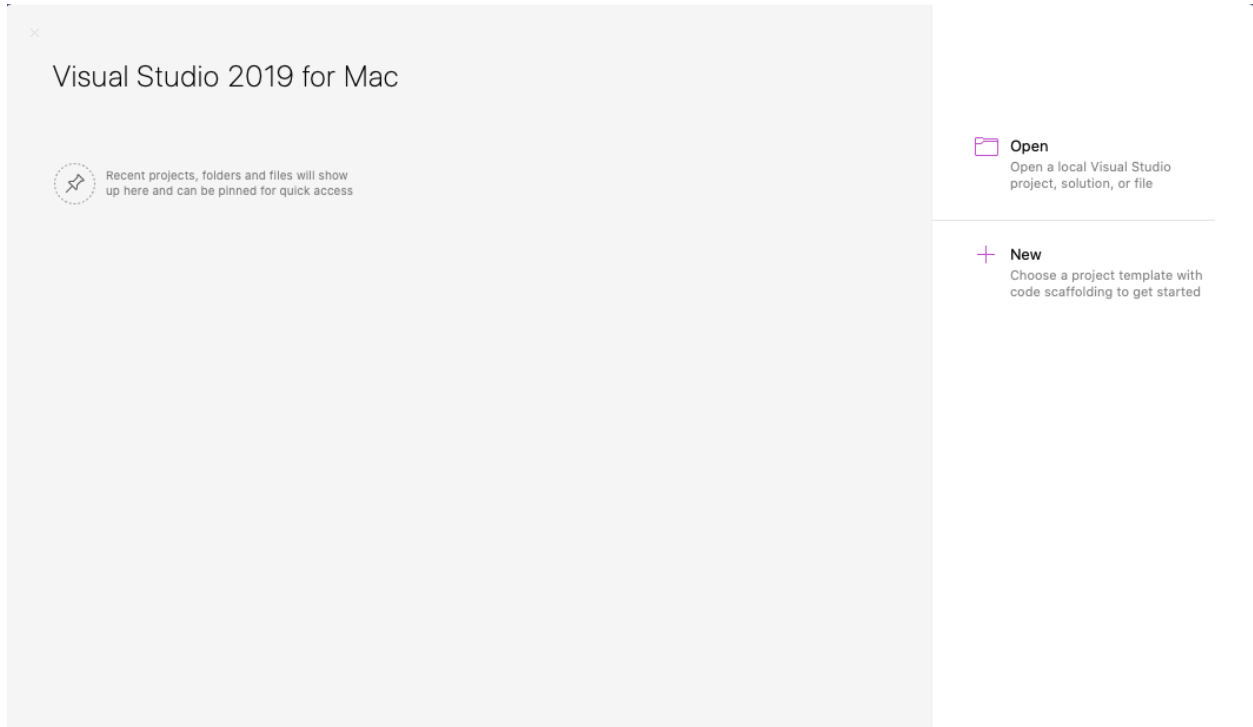
Prerequisites

- [Visual Studio for Mac](#)
- [.NET Core SDK 3.1.8](#) / [.NET 5.0 SDK](#)

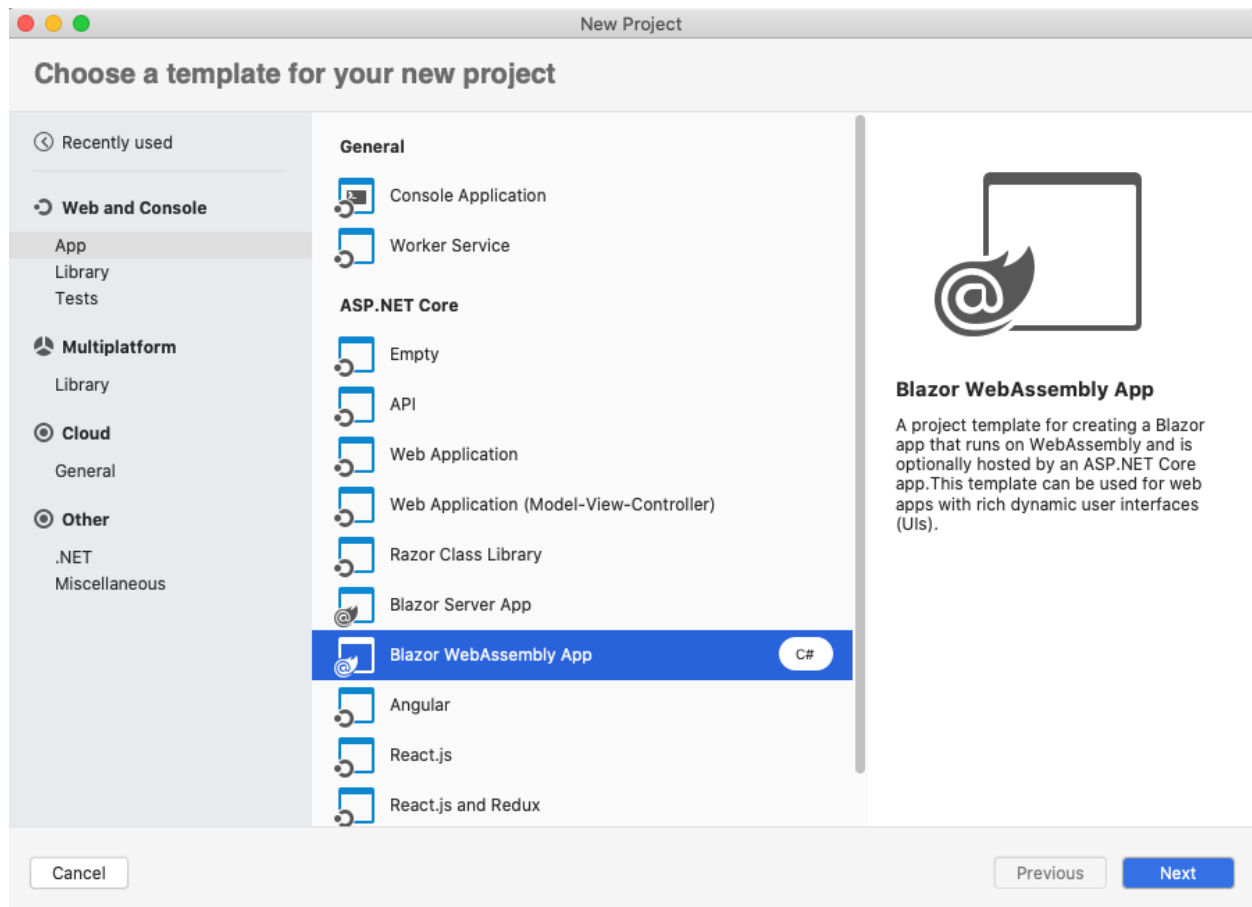
.NET Core SDK 3.1.8 requires Visual Studio for Mac 8.7.6 or later. **.NET 5.0** requires Visual Studio 2019 for Mac 8.8 or later.

Create a Blazor WebAssembly project in Visual Studio for Mac

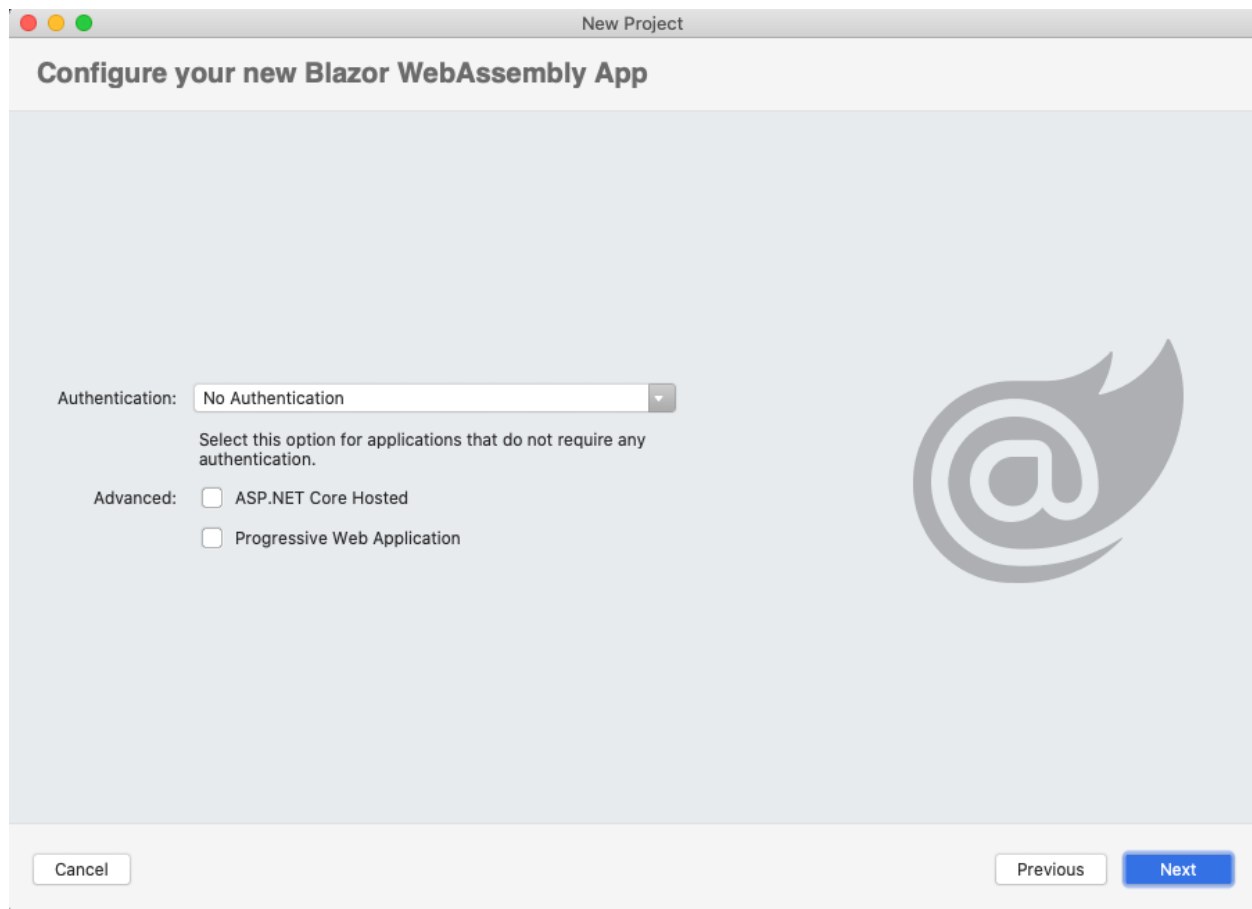
1. Choose **New** from the Visual Studio for Mac dashboard.



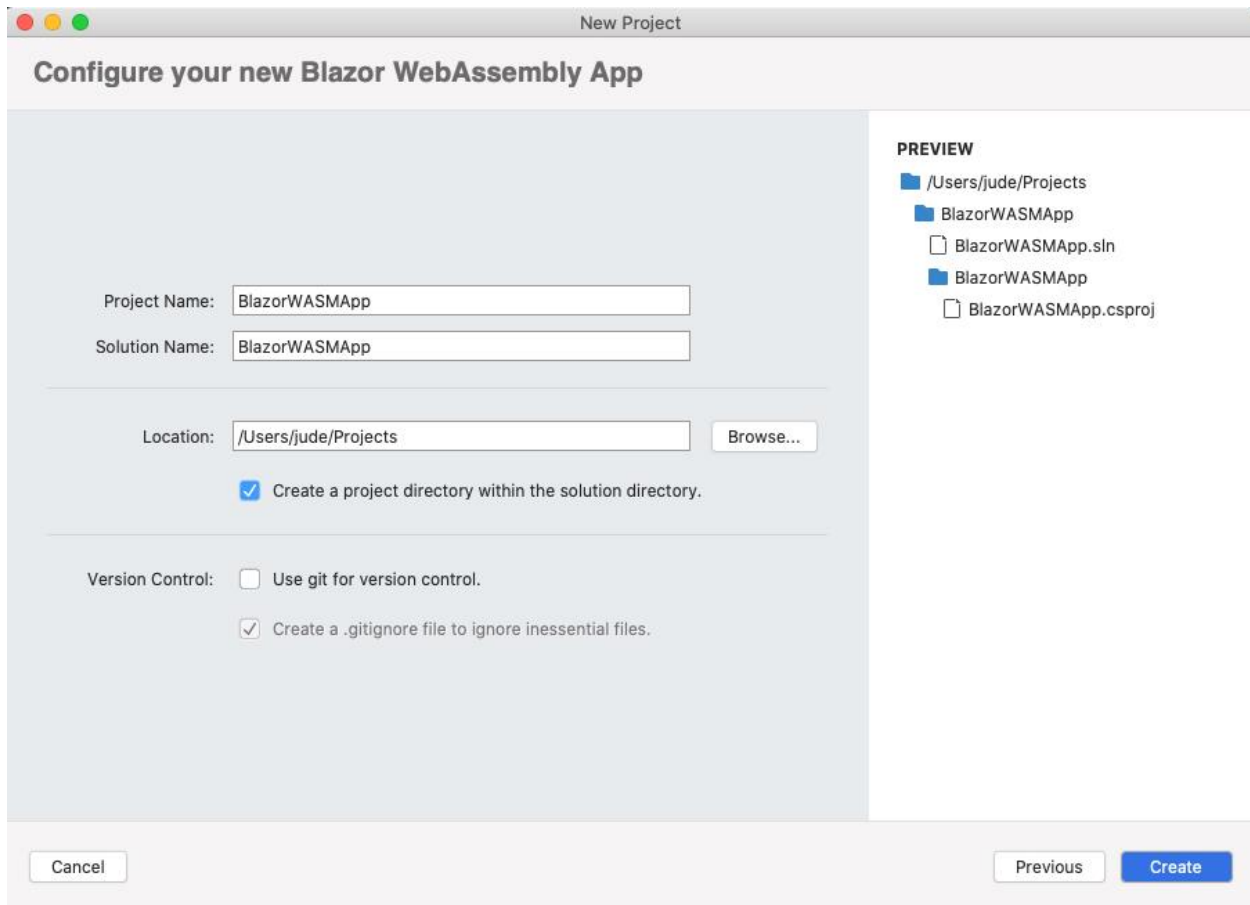
2. Select **Blazor WebAssembly App** from the template, and then click **Next** button.



3. Continue with **No Authentication** selection in Authentication, and then click **Next** button.



4. Now, the Blazor WebAssembly App project configuration window will popup. Click **Create** button to create a new project after filling the Project name.



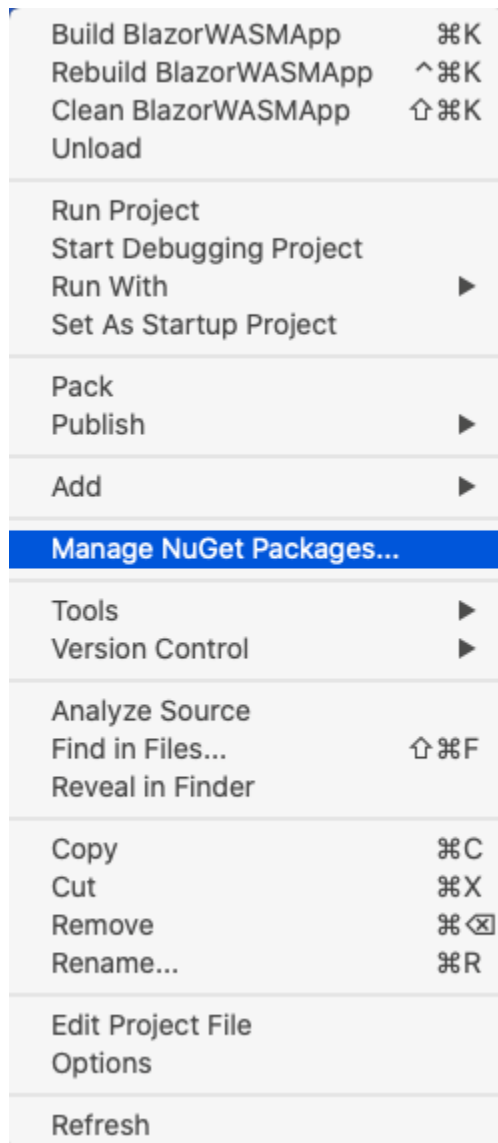
Installing Syncfusion Blazor packages in the application

You can use any one of the below standards to install the Syncfusion Blazor library in your application.

[Using Syncfusion Blazor individual NuGet Packages \[New standard\]](#)

Starting with Volume 4, 2020 (v18.4.0.30) release, Syncfusion provides [individual NuGet packages](#) for our Syncfusion Blazor components. We highly recommend this new standard for your Blazor production applications. Refer to [this section](#) to know the benefits of the individual NuGet packages.

1. Now, add **Syncfusion.Blazor.Calendars** NuGet package to the new application using the **NuGet Package Manager**. For more details about the available NuGet packages, refer to the [Individual NuGet Packages](#) documentation.
2. Right-click the project, and then select Manage NuGet Packages.



3. Search **Syncfusion.Blazor.Calendars** keyword in the Browse tab and install **Syncfusion.Blazor.Calendars** NuGet package in the application.
4. The Syncfusion Blazor Calendars package will be included in the newly created project once the installation process is completed.
5. Add the Syncfusion bootstrap4 theme in the `element` of the `~/wwwroot/index.html` page.

HTML

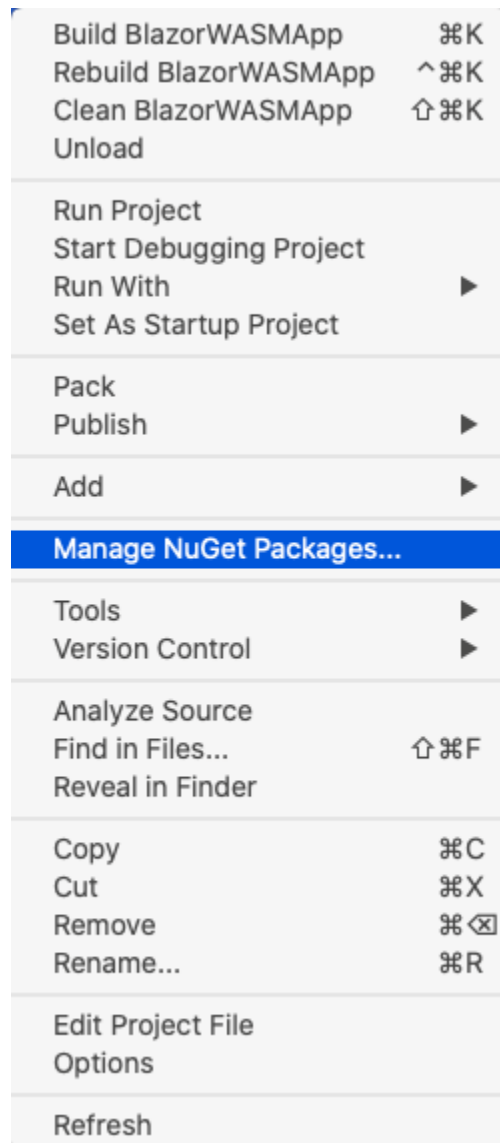
```
<head>
....
....
<link href="_content/Syncfusion.Blazor.Themes/bootstrap4.css"
rel="stylesheet" />
</head>
```

Warning: `Syncfusion.Blazor` package should not be installed along with [individual NuGet packages](#). Hence, you have to add the above `Syncfusion.Blazor.Themes` static web assets (styles) in the application.

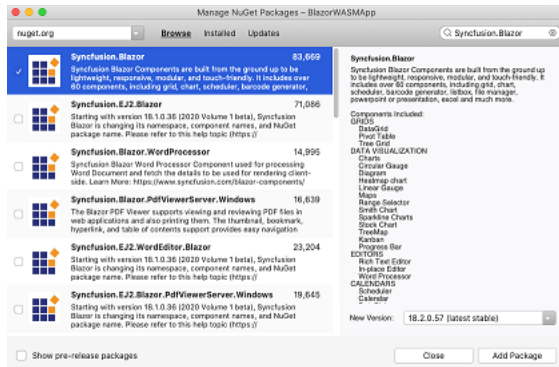
Using Syncfusion.Blazor NuGet Package [Old standard]

Warning: If you prefer the above new standard (individual NuGet packages), then skip this section. Using both old and new standards in the same application will throw ambiguous compilation errors.

1. Now, install **Syncfusion.Blazor** NuGet package to the newly created application by using the `NuGet Package Manager`. Right-click the project, and then select `Manage NuGet Packages`.



2. Search **Syncfusion.Blazor** keyword in the Browse tab and install **Syncfusion.Blazor** NuGet package in the application.



3. The Syncfusion Blazor package will be included in the newly created project once the installation process is completed.
4. Add the Syncfusion bootstrap4 theme in the element of the `~/wwwroot/index.html` page.

HTML

```
<head>
    . . . .
    . . . .
    <link href="_content/Syncfusion.Blazor/styles/bootstrap4.css"
    rel="stylesheet" />
</head>
```

The same theme file can be referred through the CDN version by using <https://cdn.syncfusion.com/blazor/{{ site.blazorversion }}/styles/bootstrap4.css>.

Adding Syncfusion Blazor component and running the application

1. Open `~/_Imports.razor` file and import the `Syncfusion.Blazor` namespace.

ASPX-CS

```
@using Syncfusion.Blazor
@using Syncfusion.Blazor.Calendars
```

2. Open the `~/Program.cs` file and register the Syncfusion Blazor Service.

C#

```
using Syncfusion.Blazor;
namespace WebApplication1
{
    public class Program
    {
        public static async Task Main(string[] args)
        {
            . . . .
            . . . .
            builder.Services.AddSyncfusionBlazor();
        }
    }
}
```

```
await builder.Build().RunAsync();
}
}
```

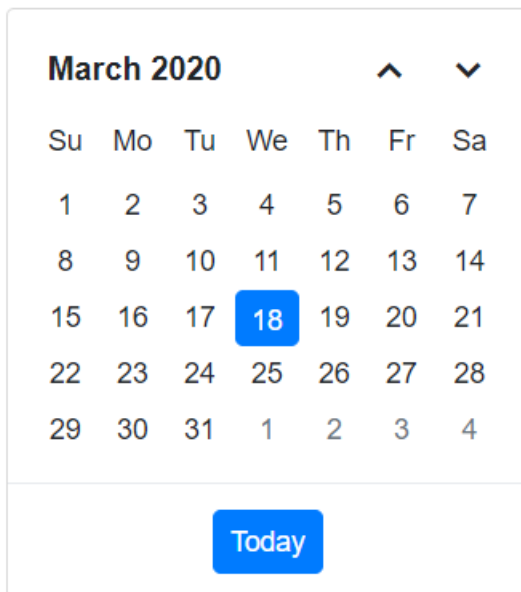
We can disable the dynamic script loading and refer to the scripts from the application end by using the `IgnoreScriptIsolation` parameter in `AddSyncfusionBlazor()` at the `program.cs`. For more details, please refer here for [how to refer custom/CDN resources](#).

- Now, add the Syncfusion Blazor components in any `.razor` file in the `~/Pages` folder. For example, the calendar component is added in the `~/Pages/Index.razor` page.

ASPX-CS

```
<SfCalendar TValue="DateTime"></SfCalendar>
```

- Run the application, the Syncfusion Blazor Calendar component will be rendered in the default web browser.



<!-- markdownlint-disable MD024 -->

Getting started with Blazor ASP.NET Core Hosted App in Visual Studio

This article provides step-by-step instructions about how to create Blazor ASP.NET Core Hosted application using [Visual Studio](#) with Syncfusion Blazor components setup pre-configured in it.

Starting with version 17.4.0.39 (2019 Volume 4), you need to include a valid license key (either paid or trial key) within your applications. Please refer to this [help topic](#) for more information.

Prerequisites

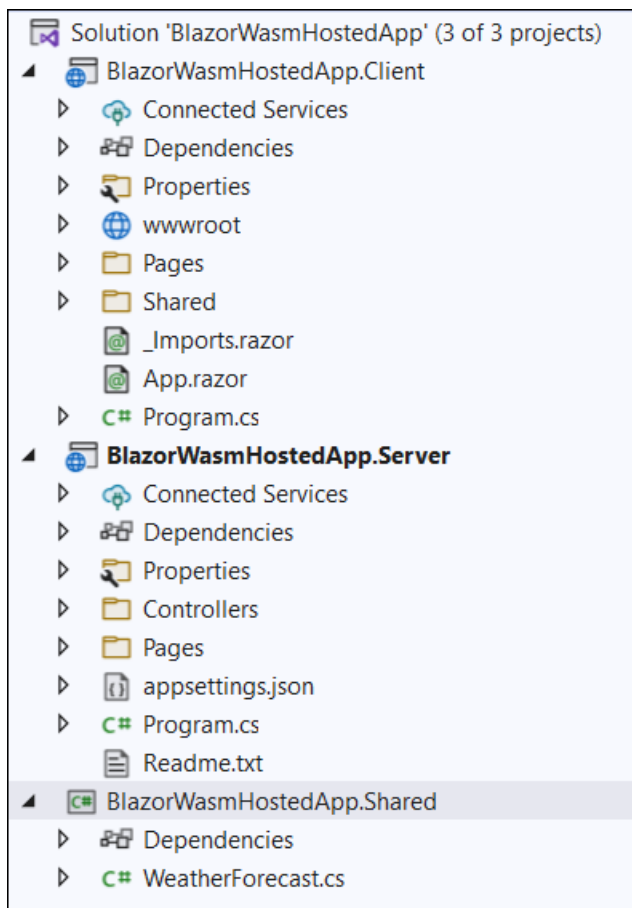
- [Visual Studio 2019](#) / [Visual Studio 2022 Preview](#)
- [.NET Core SDK 3.1.8](#) / [.NET 5.0 SDK](#) / [.NET 6.0 SDK](#)

.NET Core SDK 3.1.8 requires Visual Studio 2019 16.7 or later. **.NET 5** requires Visual Studio 2019 16.8 or later. **.NET 6** requires Visual Studio 2022 Preview 4.1 or later.

Create a Blazor ASP.NET Core Hosted project in Visual Studio

Refer to the [Blazor Tooling documentation](#) to create a new Blazor Server-Side Application using Visual Studio.

- The application will have the following structure once project is created.



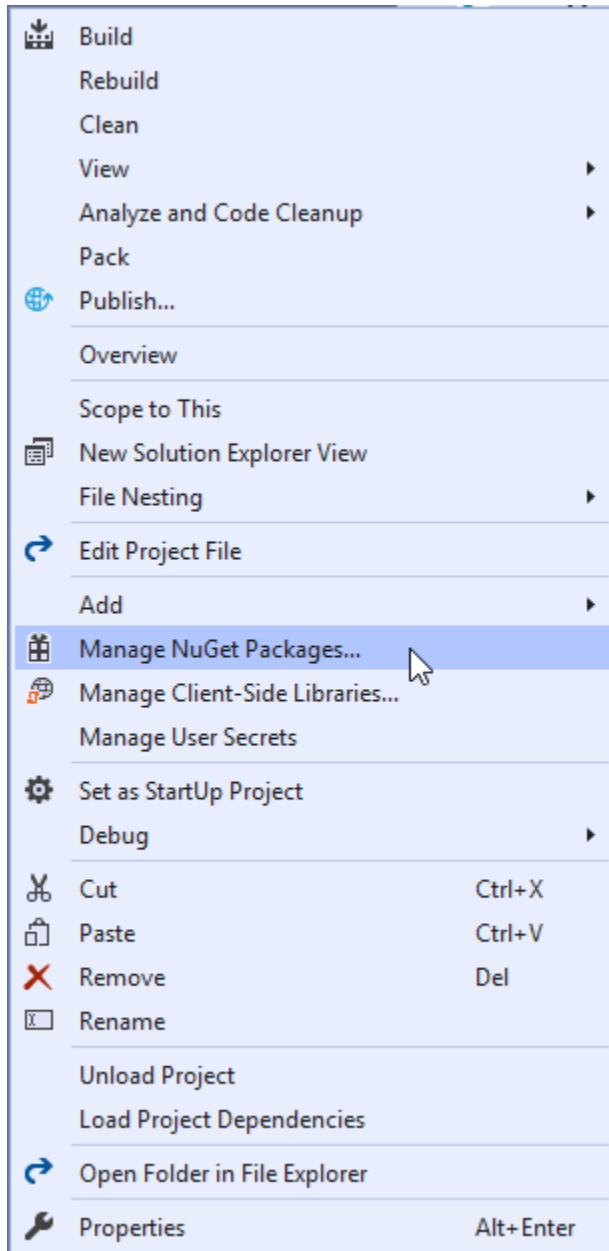
Installing Syncfusion Blazor packages in the application

You can use any one of the following standards to install the Syncfusion Blazor library in your application.

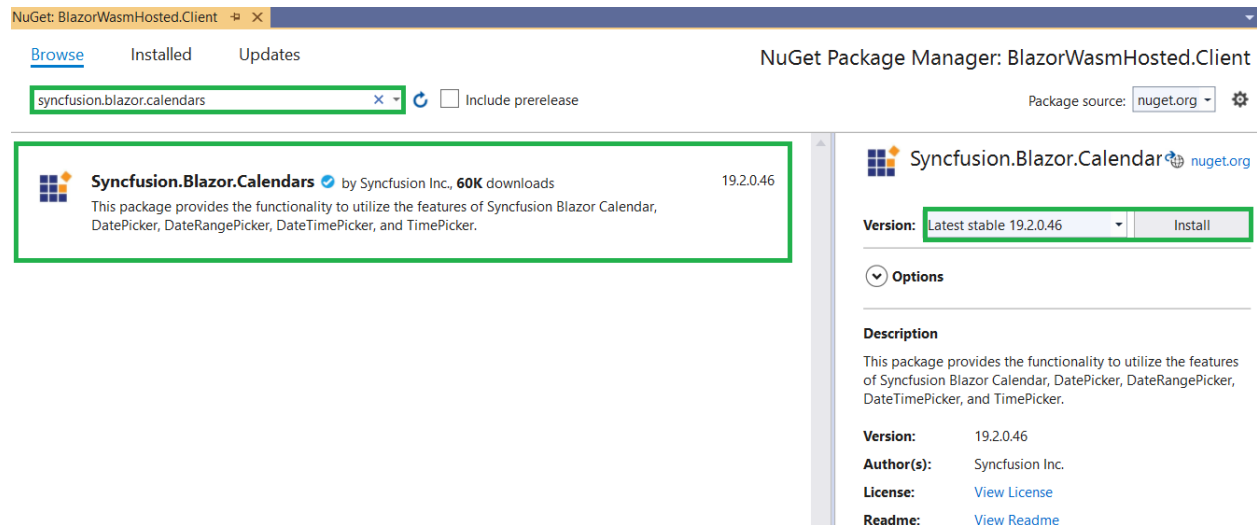
Using Syncfusion Blazor individual NuGet Packages [New standard]

Starting with Volume 4, 2020 (v18.4.0.30) release, Syncfusion provides [individual NuGet packages](#) for our Syncfusion Blazor components. We highly recommend this new standard for your Blazor production applications. Refer to [this section](#) to know the benefits of the individual NuGet packages.

1. Now, install the **Syncfusion.Blazor.Calendars** NuGet package to the new application using the **NuGet Package Manager**. For more details about available NuGet packages, refer to the [Individual NuGet Packages](#) documentation.
2. Right-click on the **Client[BlazorWasmHosted.Client]** project ,and then select Manage NuGet Packages.



3. Search **Syncfusion.Blazor.Calendars** keyword in the Browse tab and install **Syncfusion.Blazor.Calendars** NuGet package in the application.



4. The Syncfusion Blazor Calendars package will be included in the newly created project once the installation process is completed.
5. Add the Syncfusion bootstrap4 theme in the `element` of the `~/wwwroot/index.html` page in `Client[BlazorWasmHosted.Client]` project.

HTML

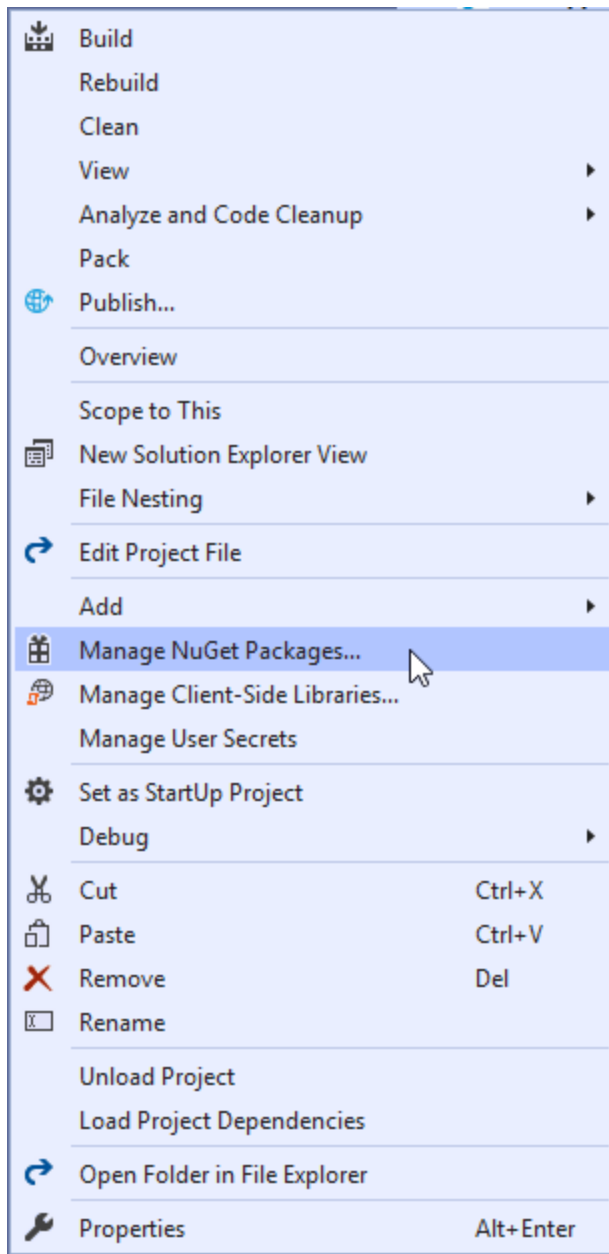
```
<head>
....
....
<link href="_content/Syncfusion.Blazor.Themes/bootstrap4.css"
rel="stylesheet" />
</head>
```

Warning: `Syncfusion.Blazor` package should not be installed along with [individual NuGet packages](#). Hence, you have to add the above `Syncfusion.Blazor.Themes` static web assets (styles) in the application.

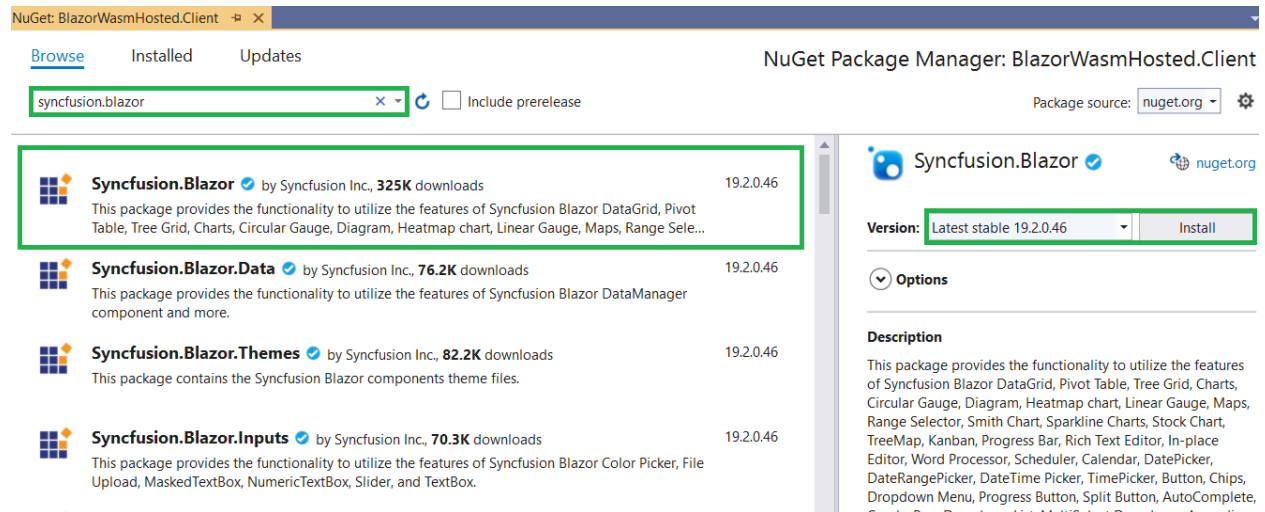
Using Syncfusion.Blazor NuGet Package [Old standard]

Warning: If you prefer the above new standard (individual NuGet packages), then skip this section. Using both old and new standards in the same application will throw ambiguous compilation errors.

1. Now, install the **Syncfusion.Blazor** NuGet package to the newly created application by using the **NuGet Package Manager**. Right-click on the `Client[BlazorWasmHosted.Client]` project and then select **Manage NuGet Packages**.



2. Search **Syncfusion.Blazor** keyword in the Browse tab and install **Syncfusion.Blazor** NuGet package in the application.



3. The Syncfusion Blazor package will be installed in the project once the installation process is completed.
4. Add the Syncfusion bootstrap4 theme in the `element` of the `~/wwwroot/index.html` page in `Client[BlazorWasmHosted.Client]` project.

HTML

```
<head>
...
...
<link href="_content/Syncfusion.Blazor/styles/bootstrap4.css"
rel="stylesheet" />
</head>
```

The same theme file can be referred through the CDN version by using <https://cdn.syncfusion.com/blazor/{ site.blazorversion }/styles/bootstrap4.css>.

Adding Syncfusion Blazor component and running the application

1. Open `~/_Imports.razor` file in `Client[BlazorWasmHosted.Client]` project and import the `Syncfusion.Blazor` namespace.

ASPX-CS

```
@using Syncfusion.Blazor
@using Syncfusion.Blazor.Calendar
```

2. Open the `~/Program.cs` file in the `Client[BlazorWasmHosted.Client]` project and register the Syncfusion Blazor Service.

C#

```
// For .NET 6 project.
using Microsoft.AspNetCore.Components;
```

```
using Microsoft.AspNetCore.Components.Web;
using Syncfusion.Blazor;
var builder = WebApplication.CreateBuilder(args);
builder.Services.AddRazorPages();
builder.Services.AddServerSideBlazor();
....
builder.Services.AddSyncfusionBlazor();
var app = builder.Build();
....
....
```

C#

```
// For .NET 5 or .NET Core SDK 3.1 project.
using Syncfusion.Blazor;
namespace BlazorWasmHosted.Client
{
    public class Program
    {
        public static async Task Main(string[] args)
        {
            ....
            ....
            builder.Services.AddSyncfusionBlazor();
            await builder.Build().RunAsync();
        }
    }
}
```

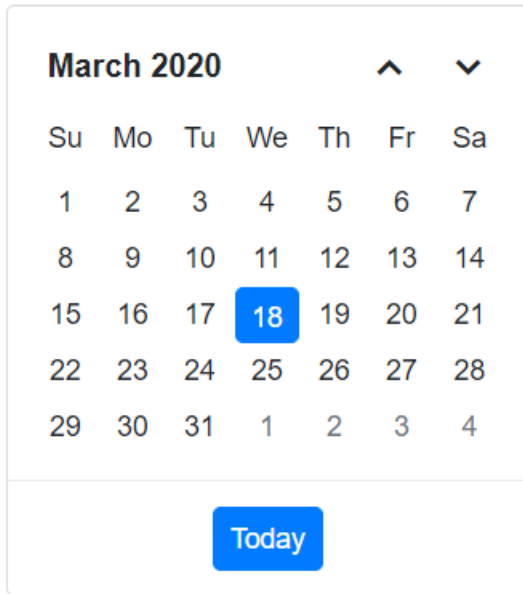
You can disable the dynamic script loading and refer to the scripts from the application end by using the `IgnoreScriptIsolation` parameter in `AddSyncfusionBlazor()` at the `Program.cs`. For more details, please refer here for [how to refer custom/CDN resources](#).

3. Now, add the Syncfusion Blazor component in any web page (razor) in the `Pages` folder of `Client[BlazorWasmHosted.Client]` project. For example, the calendar component is added to the `~/Pages/Index.razor` page.

ASPX-CS

```
<SfCalendar TValue="DateTime"></SfCalendar>
```

4. Run the application. Then, the Syncfusion Blazor Calendar component will be rendered in the default web browser.



For ASP.NET Core Hosted Blazor application, the **Server[BlazorWasmHosted.Server]** project should be the startup project.

See Also

- [ASP.NET Core Blazor hosting models](#)

<!-- markdownlint-disable MD024 -->

Getting started with Blazor ASP.NET Core Hosted App in .NET Core CLI

This article provides step-by-step instructions about how to create Blazor ASP.NET Core Hosted application using [.NET Core CLI](#) with Syncfusion Blazor components setup pre-configured in it.

Starting with version 17.4.0.39 (2019 Volume 4), you need to include a valid license key (either paid or trial key) within your applications. Please refer to this [help topic](#) for more information.

Prerequisites

- [.NET Core SDK 3.1.8](#) / [.NET 5.0 SDK](#) / [.NET 6.0 SDK](#)

Create a Blazor ASP.NET Core Hosted project using .NET Core CLI

Run the following command line to create a new Blazor WebAssembly application.

HTML

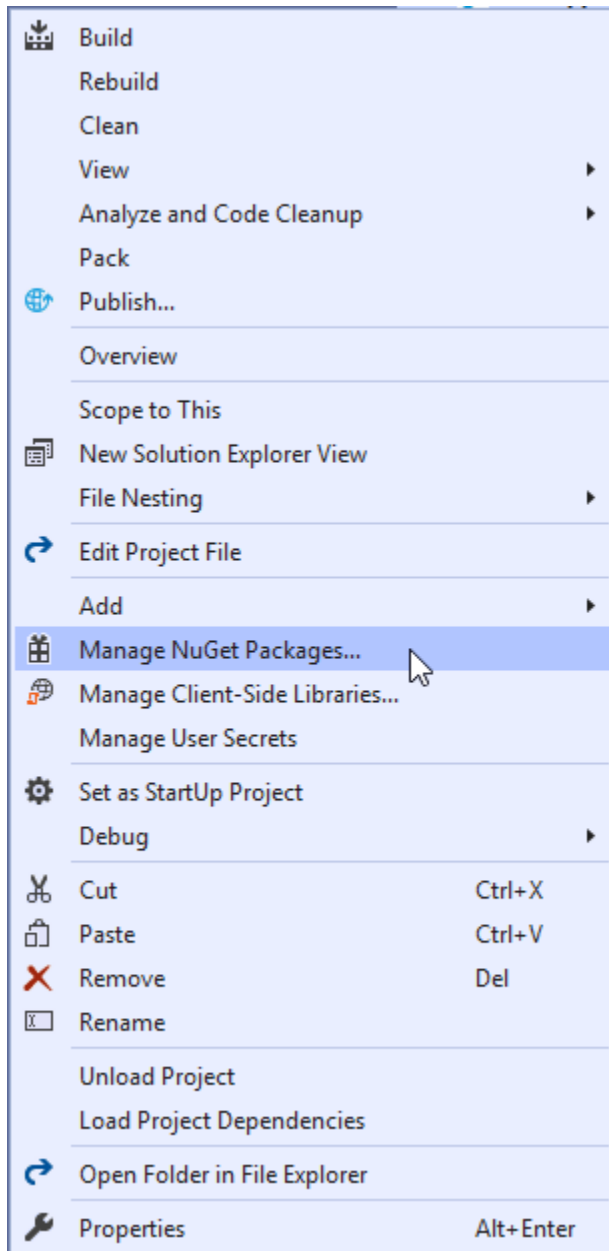
```
<head>
....
....
<link href="_content/Syncfusion.Blazor.Themes/bootstrap4.css"
rel="stylesheet" />
</head>
```

Warning: `Syncfusion.Blazor` package should not be installed along with [individual NuGet packages](#). Hence, you have to add the above `Syncfusion.Blazor.Themes` static web assets (styles) in the application.

Using Syncfusion.Blazor NuGet Package [Old standard]

Warning: If you prefer the above new standard (individual NuGet packages), then skip this section. Using both old and new standards in the same application will throw ambiguous compilation errors.

1. Now, install the **Syncfusion.Blazor** NuGet package to the newly created application by using the NuGet Package Manager. Right-click on the **Client[BlazorWasmHosted.Client]** project and then select Manage NuGet Packages.



2. Navigate to created **Client[BlazorWasmHosted.Client]** project to configure the Syncfusion Blazor Library.

BASH

```
cd Client
dotnet add package Syncfusion.Blazor -v ':{:nuget-version:}'
dotnet restore
```

2. The Syncfusion Blazor Calendars package will be included in the newly created project once the installation process is completed.
3. Add the Syncfusion bootstrap4 theme in the `element` of the `~/wwwroot/index.html` page in **Client[BlazorWasmHosted.Client]** project.

HTML

```
<head>
....
....
<link href="_content/Syncfusion.Blazor/styles/bootstrap4.css"
rel="stylesheet" />
</head>
```

The same theme file can be referred through the CDN version by using

<https://cdn.syncfusion.com/blazor/{:site.blazorversion:}/styles/bootstrap4.css>.

Adding Syncfusion Blazor component and running the application

1. Open `~/_Imports.razor` file in **Client[BlazorWasmHosted.Client]** project and import the `Syncfusion.Blazor` namespace.

ASPX-CS

```
@using Syncfusion.Blazor
@using Syncfusion.Blazor.Calendars
```

2. Open the `~/Program.cs` file in the **Client[BlazorWasmHosted.Client]** project and register the Syncfusion Blazor Service.

C#

```
// For .NET 6 project.
using Microsoft.AspNetCore.Components;
using Microsoft.AspNetCore.Components.Web;
using Syncfusion.Blazor;
var builder = WebApplication.CreateBuilder(args);
builder.Services.AddRazorPages();
builder.Services.AddServerSideBlazor();
....
builder.Services.AddSyncfusionBlazor();
```

```
var app = builder.Build();  
....  
....
```

C#

```
// For .NET 5 or .NET Core SDK 3.1 project.  
using Syncfusion.Blazor;  
namespace BlazorWasmHosted.Client  
{  
    public class Program  
    {  
        public static async Task Main(string[] args)  
        {  
            ....  
            ....  
            builder.Services.AddSyncfusionBlazor();  
            await builder.Build().RunAsync();  
        }  
    }  
}
```

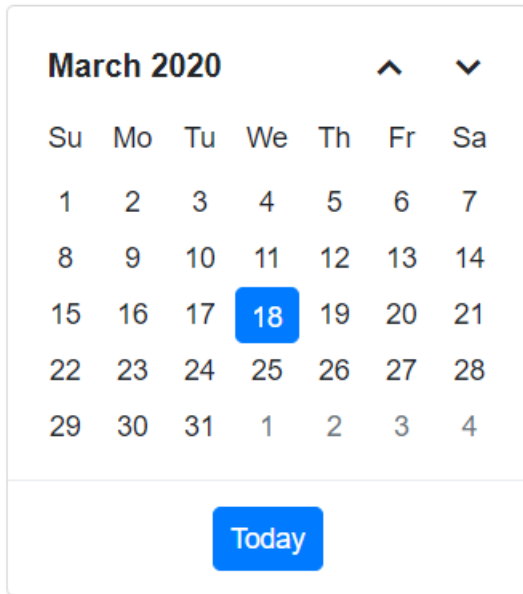
You can disable the dynamic script loading and refer to the scripts from the application end by using the `IgnoreScriptIsolation` parameter in `AddSyncfusionBlazor()` at the `Program.cs`. For more details, please refer here for [how to refer custom/CDN resources](#).

3. Now, add the Syncfusion Blazor component in any web page (razor) in the `Pages` folder of `Client[BlazorWasmHosted.Client]` project. For example, the calendar component is added to the `~/Pages/Index.razor` page.

ASPX-CS

```
<SfCalendar TValue="DateTime"></SfCalendar>
```

4. Then Navigate to the `Server[BlazorWasmHosted.Server]` project and Run the `dotnet run` command to run the application. The Syncfusion Blazor Calendar component will be rendered in the web browser.



For ASP.NET Core Hosted Blazor application, the **Server[BlazorWasmHosted.Server]** project should be the startup project.

See Also

- [ASP.NET Core Blazor hosting models](#)

Creating Razor Class Library (RCL) using Syncfusion Blazor components

This section provides information about creating Razor Class Library with the Syncfusion Blazor components using [Visual Studio](#).

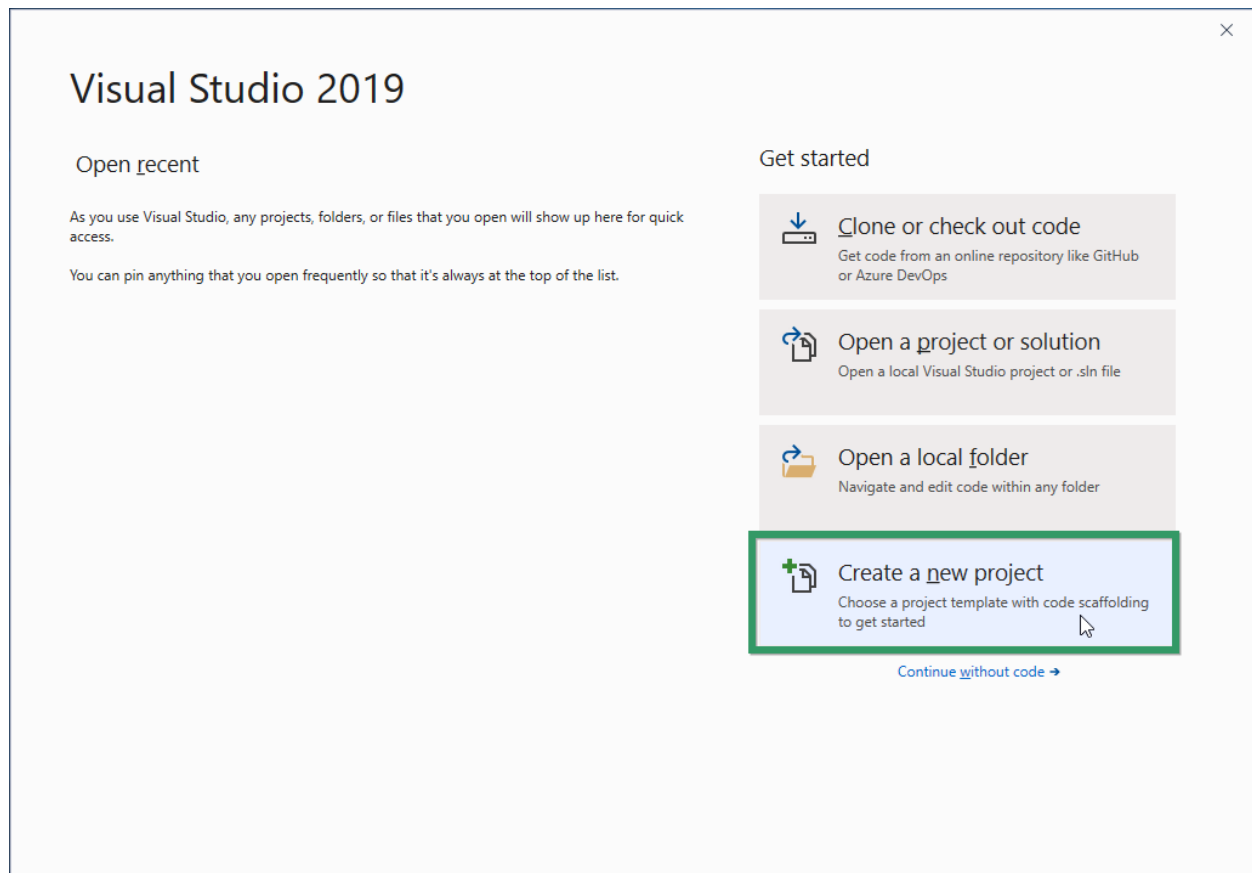
Prerequisites

- [Visual Studio 2019](#) / [Visual Studio 2022 Preview](#)
- [.NET Core SDK 3.1.8](#) / [.NET 5.0 SDK](#) / [.NET 6.0 SDK](#)

.NET Core SDK 3.1.8 requires Visual Studio 2019 16.7 or later.
 .NET 5 requires Visual Studio 2019 16.8 or later.
 .NET 6 requires Visual Studio 2022 Preview 4.1 or later.

Create a Razor Class Library using Syncfusion Blazor components in Visual Studio 2019

1. Choose **Create a new project** from the Visual Studio dashboard.



2. Select **Razor Class Library** from the template, and then click the **Next** button.

Create a new project


Recent project templates


	Blazor App	C#
	Razor Class Library	C#
	Console App (.NET Core)	C#
	WPF App (.NET Framework)	C#
	ASP.NET Web Application (.NET Framework)	C#
	ASP.NET Core Web Application	C#


All languages


All platforms


All project types

 **Razor Class Library**
A project template for creating a Razor class library.
C# Library Linux macOS Web Windows

 **Worker Service**
A project template for creating a worker service using .NET Core.
Cloud C# Linux macOS Service Windows

 **MSTest Test Project (.NET Core)**
A project that contains MSTest unit tests that can run on .NET Core on Windows, Linux and MacOS.
C# Linux macOS Test Windows

 **MSTest Test Project (.NET Core)**
A project that contains MSTest unit tests that can run on .NET Core on Windows, Linux and MacOS.
Linux macOS Test Visual Basic Windows

 **NUnit Test Project (.NET Core)**
A project that contains NUnit tests that can run on .NET Core on Windows, Linux and MacOS.

Back

Next

- Now, the project configuration window will popup. Click **Create** button to create a new project with the default project configuration.

Configure your new project

Razor Class Library

C#

Library

Linux

macOS

Web

Windows

Project name

RazorClassLibrary

Location

C:\Users\Documentation\source\repos



Solution name ⓘ

RazorClassLibrary

☐ Place solution and project in the same directory


Back

Create

4. Choose **Razor Class Library** from the dashboard and click **Create** button to create a new Razor Class Library Server application. Select the target framework **.NET Core 3.1** or **.NET5.0** at the top based on your required target.

Create a new Razor class library

.NET Core 3.1

**Razor Class Library**
A project template for creating a Razor class library.

Authentication
No Authentication
[Change](#)

Advanced
☐ Enable Docker Support
(Requires Docker Desktop)

Linux

☐ Support pages and views

Author: Microsoft
Source: Templates 3.1.6

[Get additional project templates](#)

Back

Create

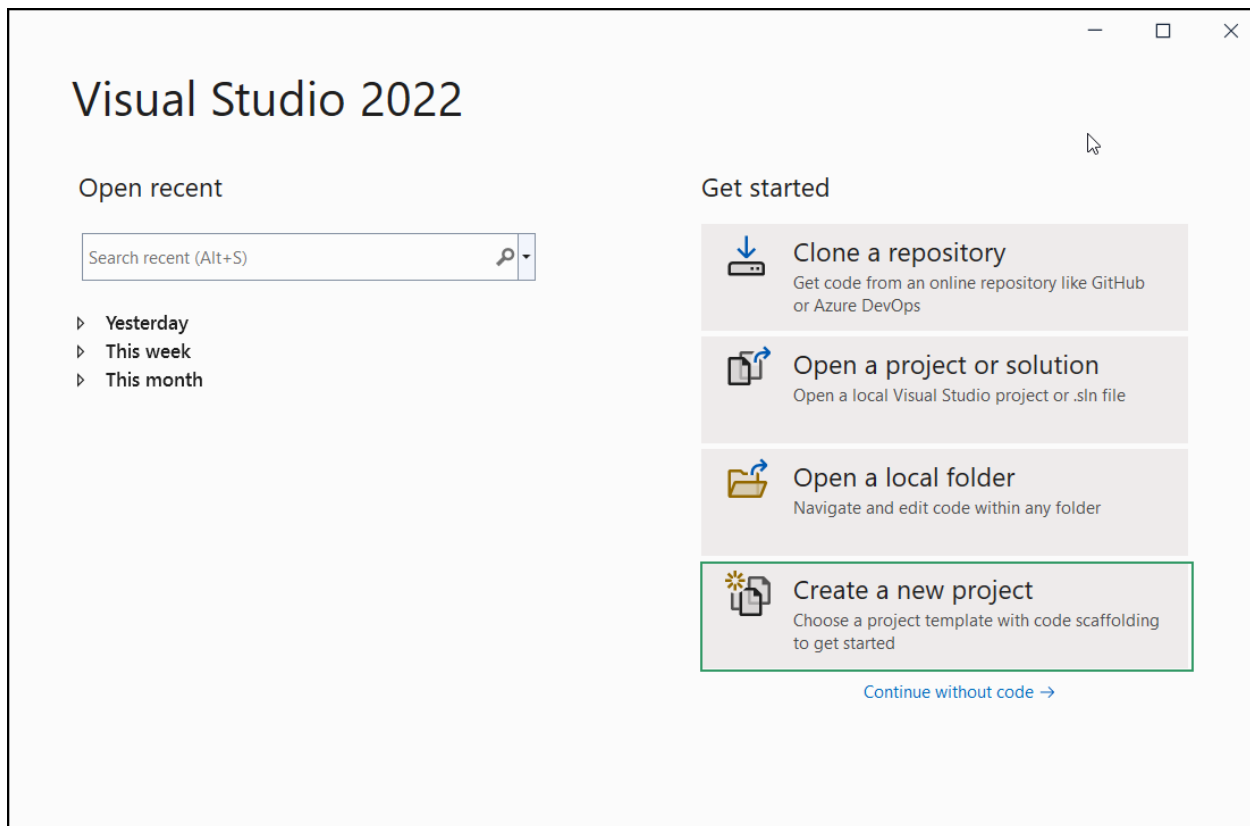
If existing .NET Standard version is 2.0 in `RazorClassLibrary.csproj`, then change it to **.NET Standard2.1** or above.

XML

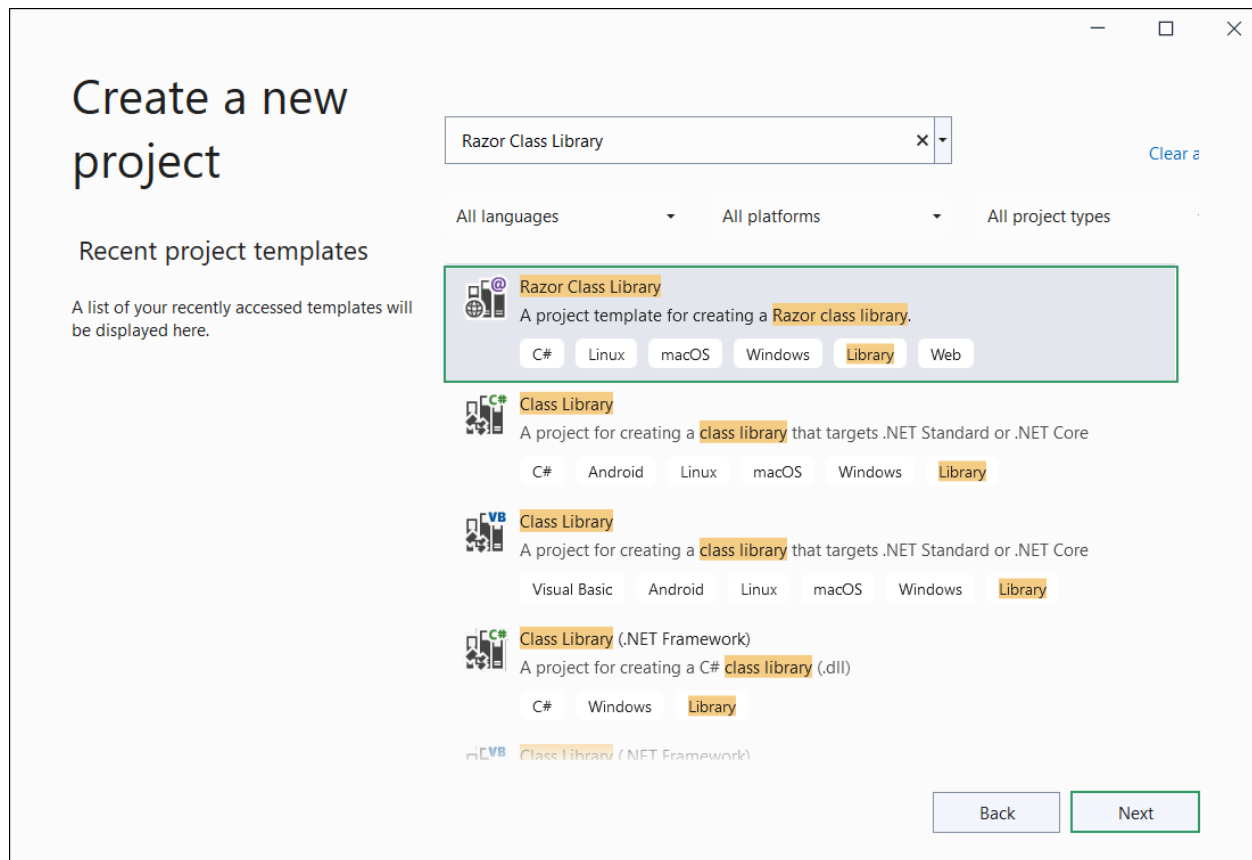
```
<PropertyGroup>
<TargetFramework>netstandard2.1</TargetFramework>
....
</PropertyGroup>
```

Create a Razor Class Library using Syncfusion Blazor components in Visual Studio 2022

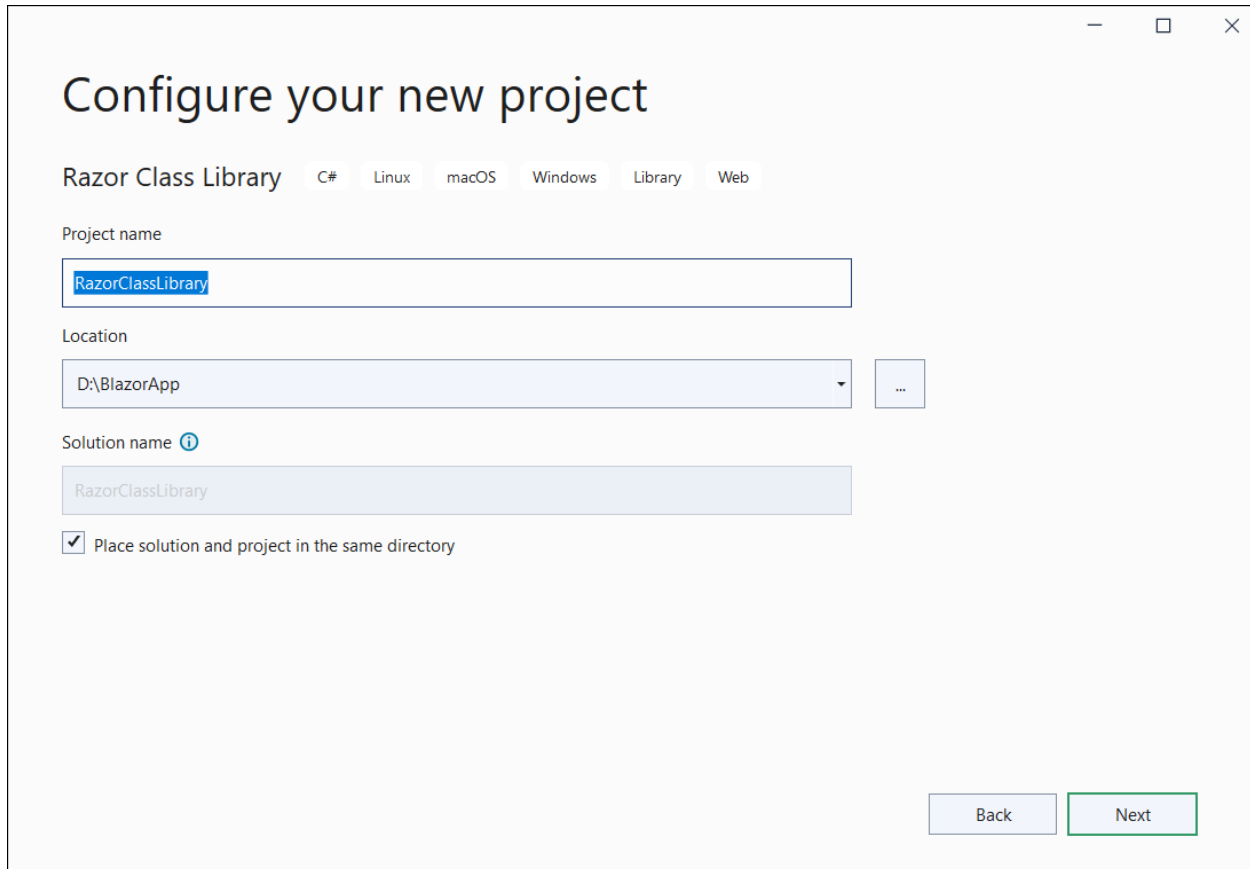
1. Choose **Create a new project** from the Visual Studio dashboard.



2. Select **Razor Class Library** from the template, and then click the **Next** button.



- Now, the project configuration window will popup. Click **Create** button to create a new project with the default project configuration.



Configure your new project

Razor Class Library C# Linux macOS Windows Library Web

Project name

RazorClassLibrary

Location

D:\BlazorApp

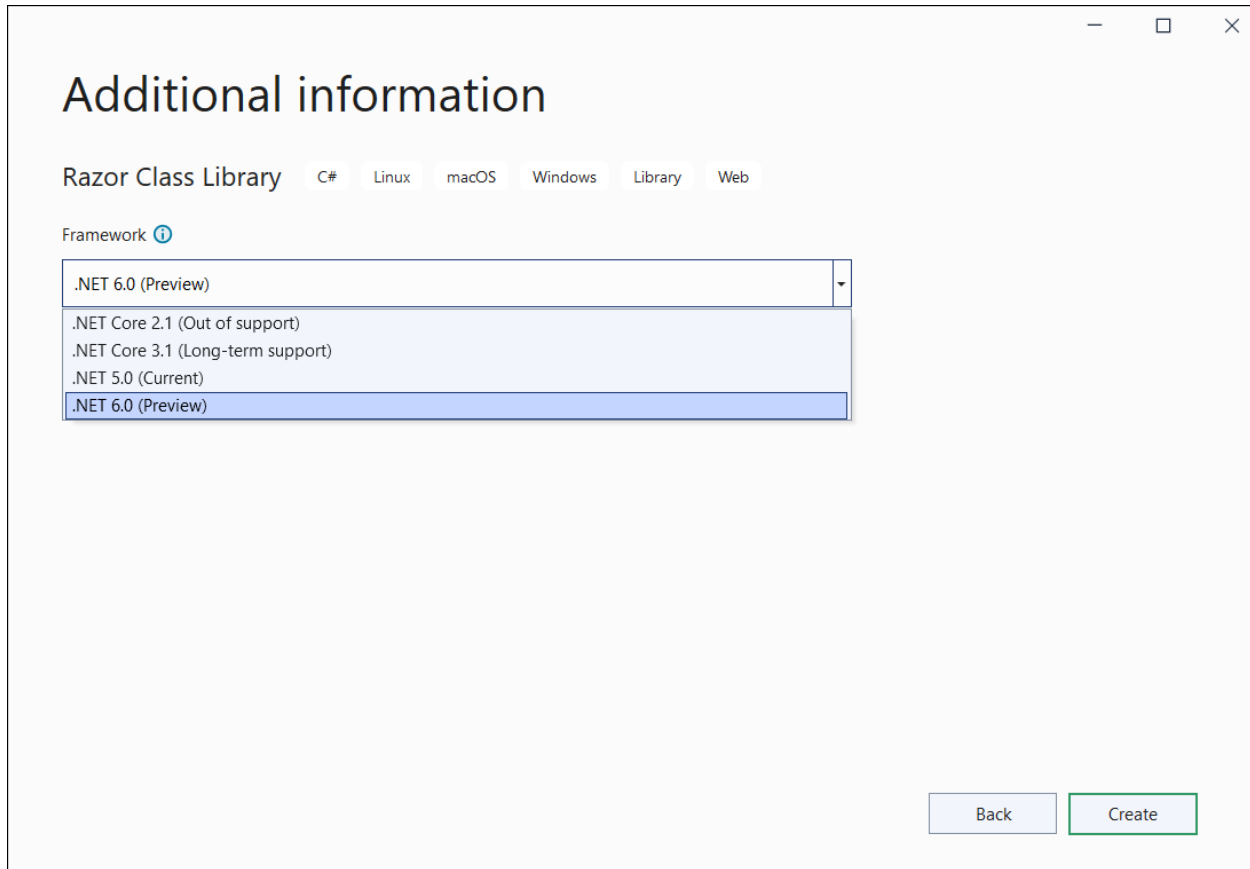
Solution name ⓘ

RazorClassLibrary

☒ Place solution and project in the same directory

Back Next

4. Select the target Framework **.NET 6** at the top of the Application based on your required target that you want and then click the **Create** button to create a new Razor Class Library application.



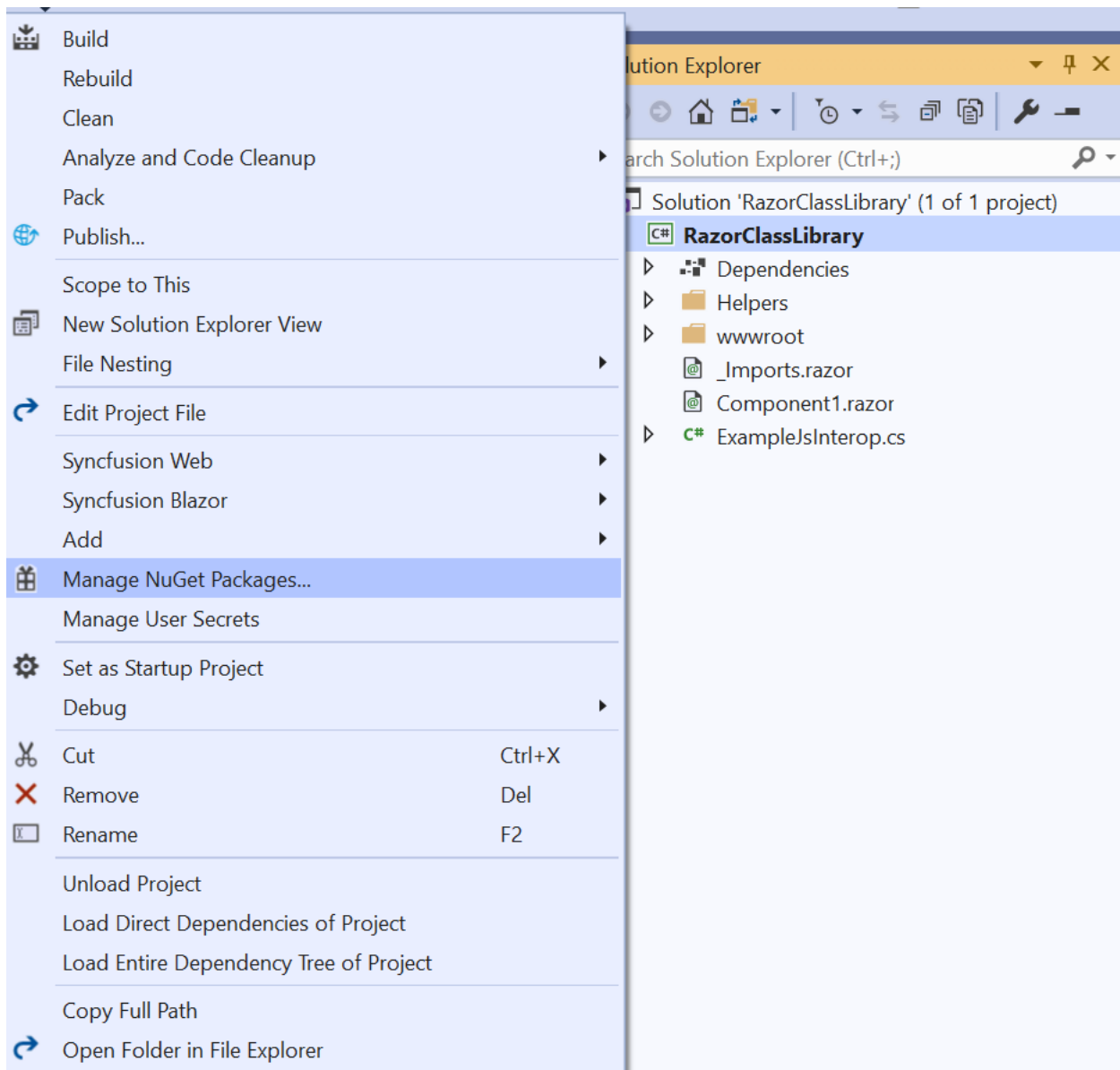
Importing Syncfusion Blazor component in Razor Class Library

You can use any one of the below standards to install the Syncfusion Blazor library in your Razor Class Library Server application.

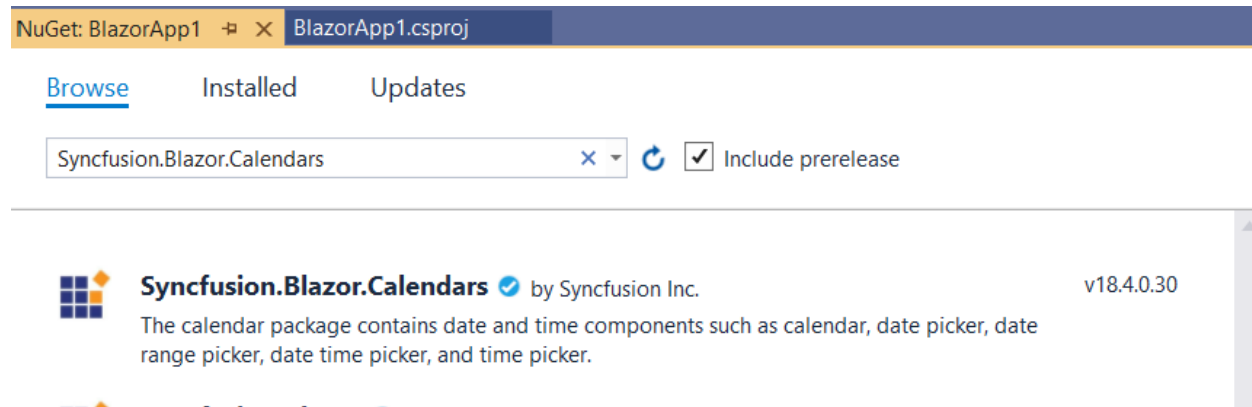
Using Syncfusion Blazor individual NuGet Packages [New standard]

Starting with Volume 4, 2020 (v18.4.0.30) release, Syncfusion provides [individual NuGet packages](#) for our Syncfusion Blazor components. We highly recommend this new standard for your Blazor production applications. Refer to [this section](#) to know the benefits of the individual NuGet packages.

1. Now, install **Syncfusion.Blazor.Calendars** NuGet package to the newly created RLC by using the **NuGet Package Manager**. For more details about the available NuGet packages, refer to the [Individual NuGet Packages](#) documentation.
2. Right-click the project, and then select Manage NuGet Packages.



3. Search **Syncfusion.Blazor.Calendars** keyword in the Browse tab and install **Syncfusion.Blazor.Calendars** NuGet package in RLC.



4. The Syncfusion Blazor Calendars package will be included in the newly created project once the installation process is completed.
5. Now, import and add the Syncfusion Blazor components in the `~/Component.razor` file. For example, the Calendar component is imported and added in the `~/Component.razor` page.

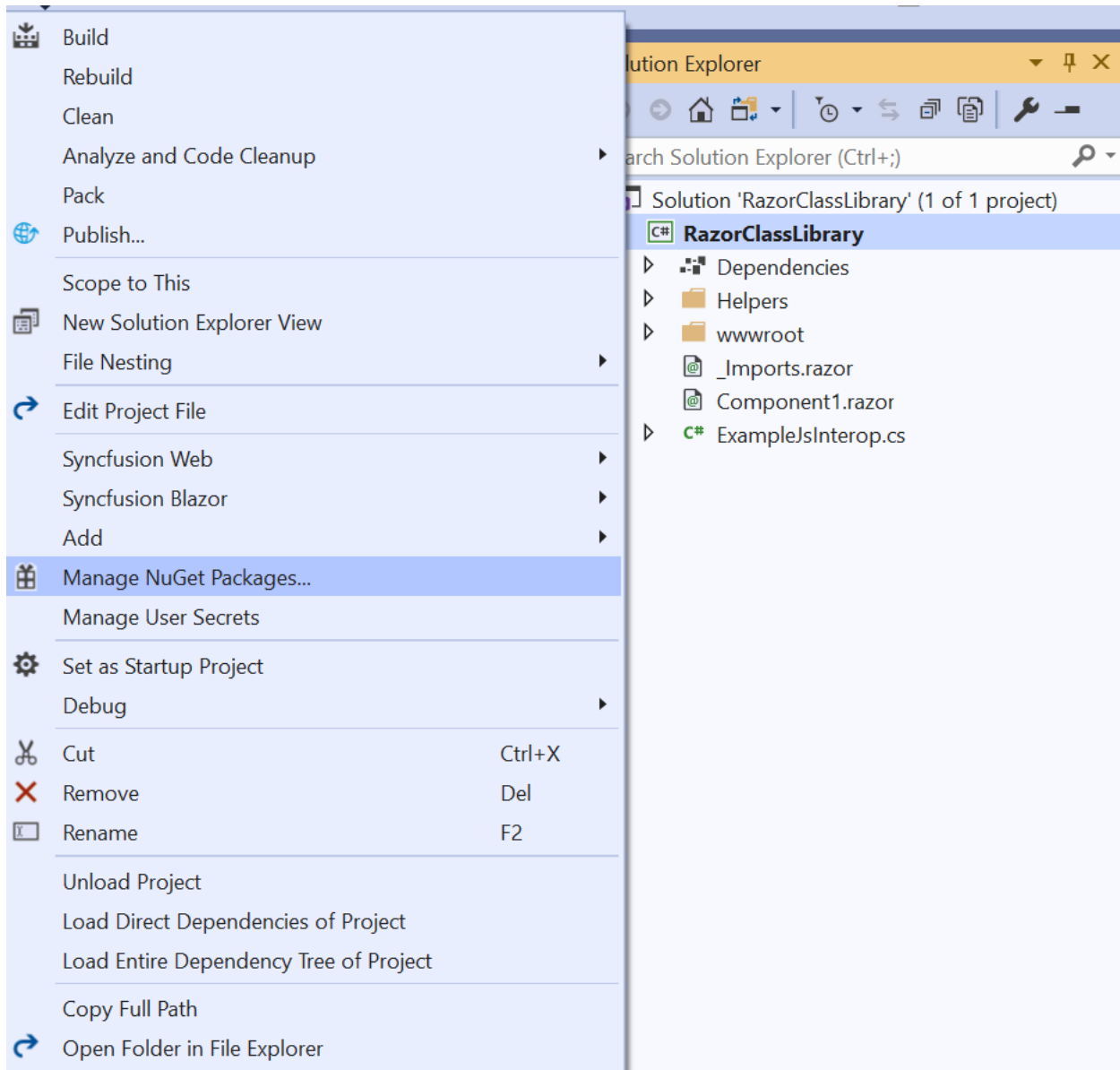
ASPX-CS

```
@using Syncfusion.Blazor.Calendars
<div class="my-component">
This Blazor component is defined in the <strong>RazorClassLibrary</strong>
package.
</div><br />
<SfCalendar TValue="DateTime"></SfCalendar>
```

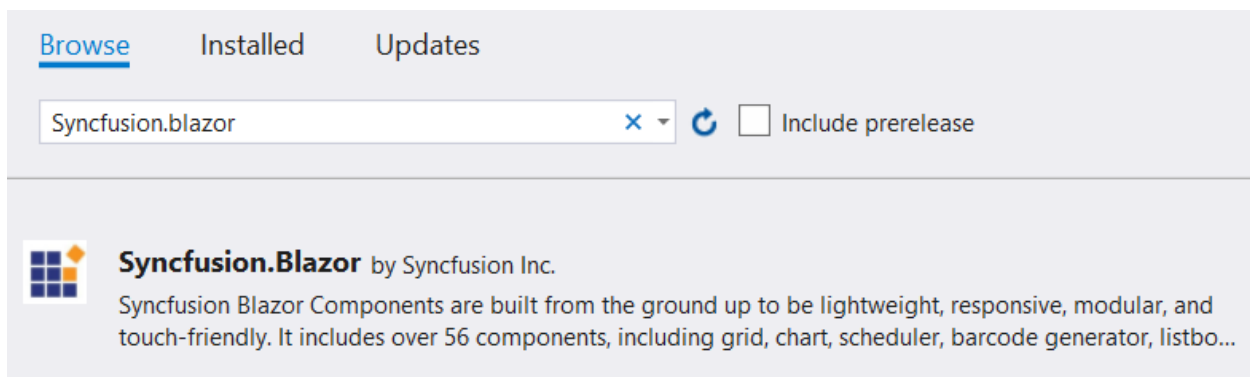
Using Syncfusion.Blazor NuGet Package [Old standard]

Warning: If you prefer the above new standard (individual NuGet packages), then skip this section. Using both old and new standards in the same application will throw ambiguous compilation errors.

1. Now, install **Syncfusion.Blazor** NuGet package to the newly created RLC by using the **NuGet Package Manager**. Right-click the project and select Manage NuGet Packages.



2. Search **Syncfusion.Blazor** keyword in the Browse tab and install **Syncfusion.Blazor** NuGet package in RLC.



3. The Syncfusion Blazor package will be installed in the project once the installation process is completed.
4. Open `~/_Imports.razor` file in RLC and import the `Syncfusion.Blazor`.

ASPX-CS

```
@using Syncfusion.Blazor
```

5. Now, import and add the Syncfusion Blazor components in the `~/Component.razor` file. For example, the Calendar component is imported and added in the `~/Component.razor` page.

ASPX-CS

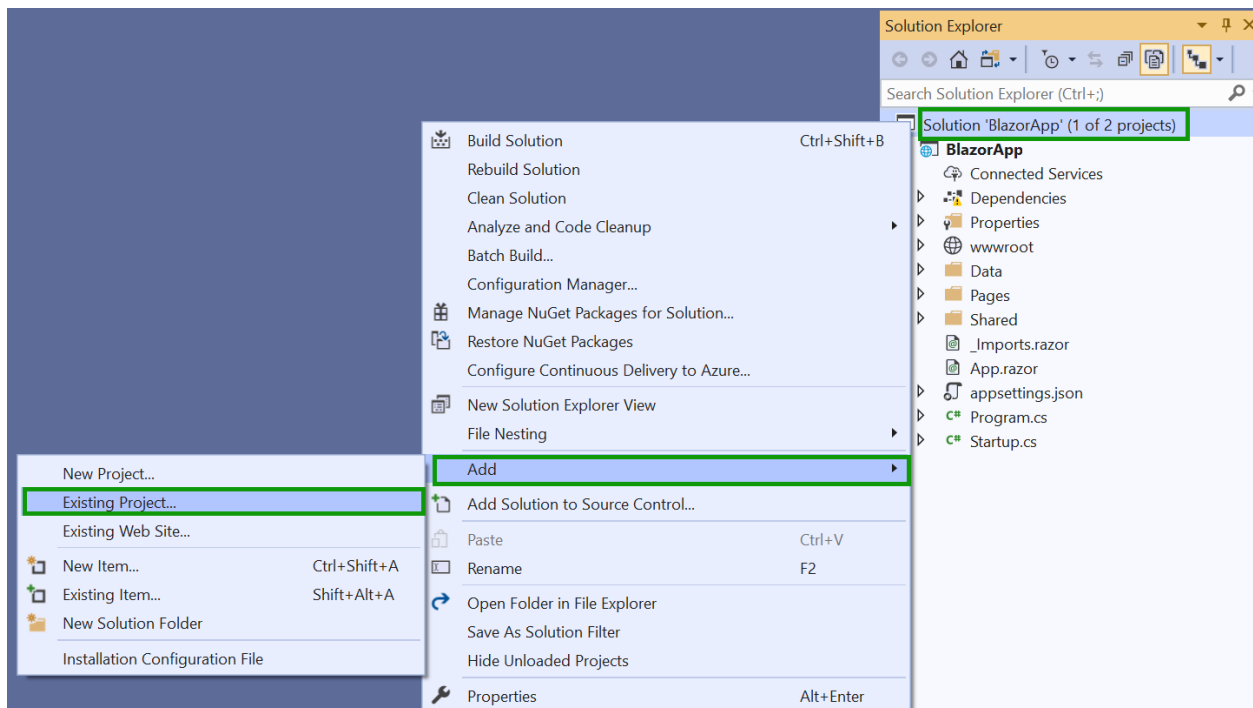
```
@using Syncfusion.Blazor.Calendars
<div class="my-component">
This Blazor component is defined in the <strong>RazorClassLibrary</strong>
package.
</div><br />
<SfCalendar TValue="DateTime"></SfCalendar>
```

Create a Blazor Server project in Visual Studio with Razor Class Library (RCL)

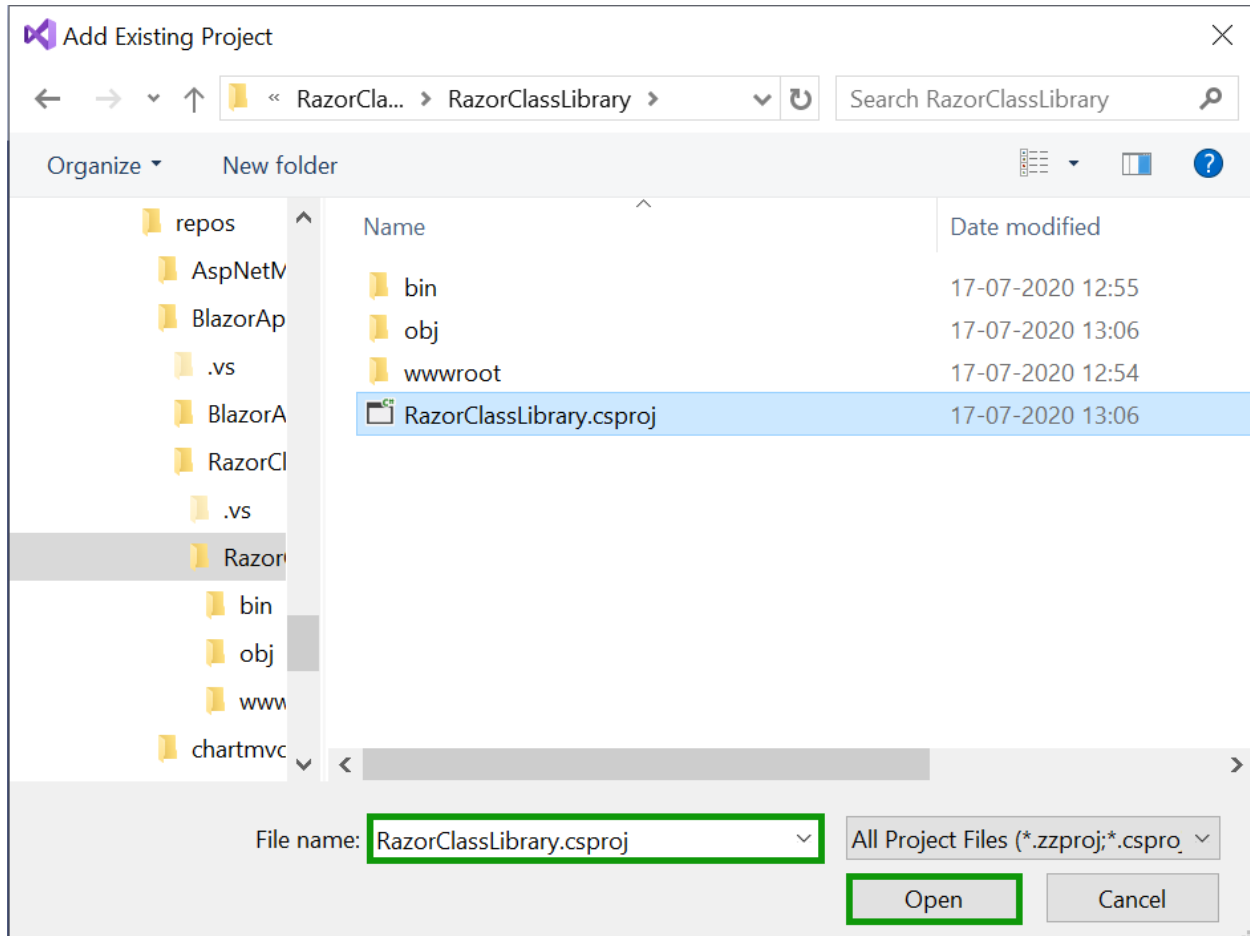
Refer to the [Blazor Tooling documentation](#) to create a new Blazor Server-Side Application using Visual Studio.

Configure the Razor Class Library and Blazor Server Application

1. Now, Right-click the solution, and then select Add/Existing Project.

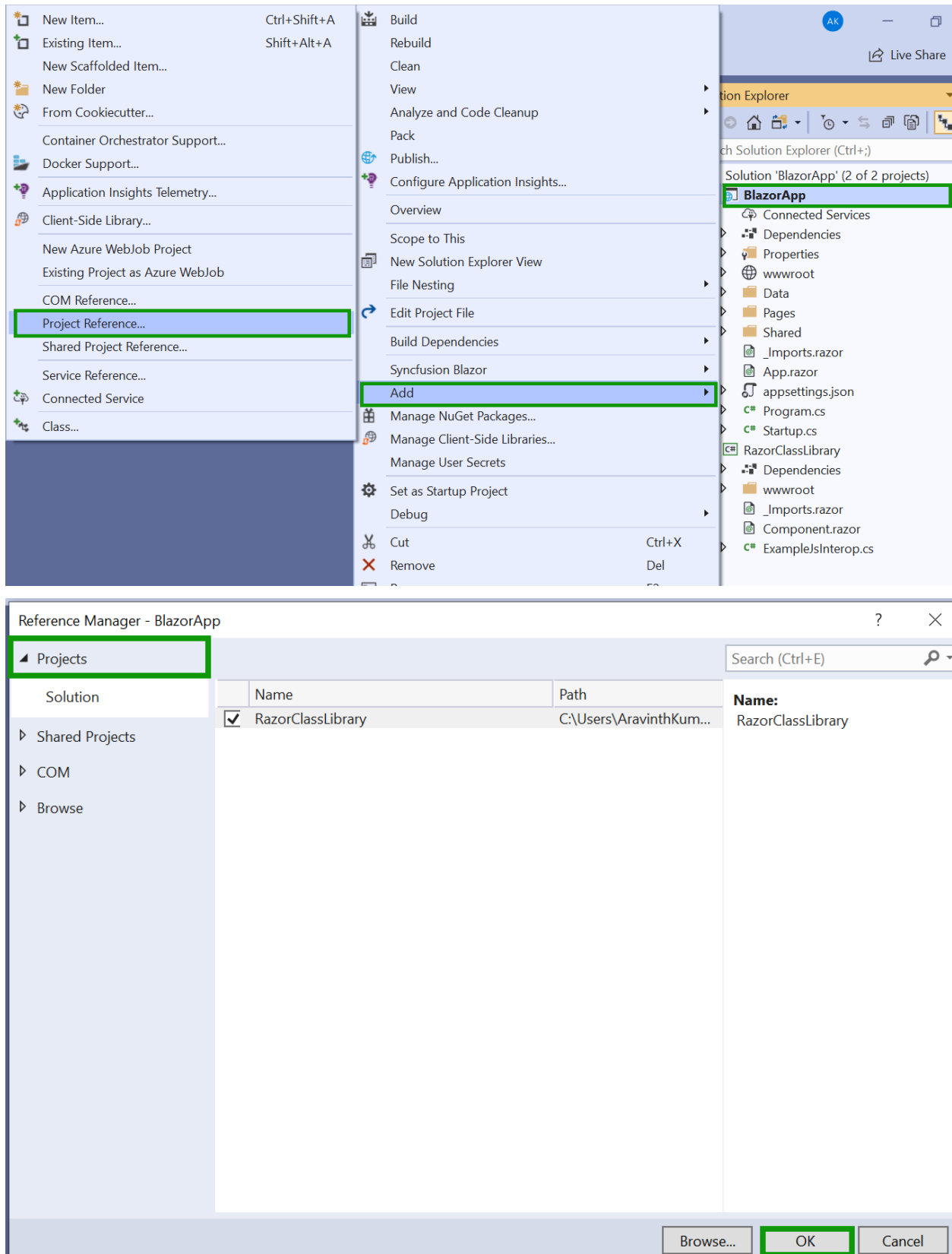


2. Add the **Razor Class Library** project by selecting the `RazorClassLibrary.csproj` file.



Razor Class Library project is added to the existing Blazor Server Application.

3. Right-click the Blazor App project, and then select Add/Project reference. Now click the checkbox and configure the **Razor Class Library** and **Blazor Server Application**.



Importing Razor Class Library in the Blazor Server Application

1. Open `~/_Imports.razor` file in Blazor App and import the `RazorClassLibrary`.

ASPX-CS

```
@using RazorClassLibrary
```

2. Now, register the Syncfusion Blazor Service to the Blazor Server App.

- a) For **.NET 6** project, open the `~/Program.cs` file and register the Syncfusion Blazor Service.

C#

```
// For .NET 6 project, add the Syncfusion Blazor Service in Program.cs file.  
using Microsoft.AspNetCore.Components;  
using Microsoft.AspNetCore.Components.Web;  
using Syncfusion.Blazor;  
var builder = WebApplication.CreateBuilder(args);  
builder.Services.AddRazorPages();  
builder.Services.AddServerSideBlazor();  
....  
builder.Services.AddSyncfusionBlazor();  
var app = builder.Build();  
....  
....
```

- b) For **.NET 5** or **.NET Core SDK 3.1** project, open the `~/Startup.cs` file and register the Syncfusion Blazor Service.

C#

```
// For .NET 5 or .NET Core SDK 3.1 project, add the Syncfusion Blazor  
Service in Startup.cs file.  
using Syncfusion.Blazor;  
namespace WebApplication1  
{  
    public class Startup  
    {  
        public void ConfigureServices(IServiceCollection services)  
        {  
            ....  
            ....  
            services.AddSyncfusionBlazor();  
        }  
    }  
}
```

3. Now, add the Syncfusion Blazor theme to the Blazor Server App.

- a) For **.NET 6** project, add the Syncfusion bootstrap4 theme in the `<head>` element of the `~/Pages/_Layout.cshtml` page.

HTML

```
<head>
....
....
// Using individual NuGet packages
<link href="_content/Syncfusion.Blazor.Themes/bootstrap4.css"
rel="stylesheet" />
// (or)
// Using overall NuGet package
<link href="_content/Syncfusion.Blazor/styles/bootstrap4.css"
rel="stylesheet" />
</head>
```

b) For **.NET 5** or **.NET Core SDK 3.1** project, add the Syncfusion bootstrap4 theme in the `<head>` element of the `~/Pages/_Host.cshtml` page.

HTML

```
<head>
....
....
// Using individual NuGet packages
<link href="_content/Syncfusion.Blazor.Themes/bootstrap4.css"
rel="stylesheet" />
// (or)
// Using overall NuGet package
<link href="_content/Syncfusion.Blazor/styles/bootstrap4.css"
rel="stylesheet" />
</head>
```

Warning: Syncfusion.Blazor package should not to be installed along with [individual NuGet packages](#). If you are using individual NuGet packages, you have to add the above Syncfusion.Blazor.Themes static web assets (styles) reference in the application. Or else, you have to add the above Syncfusion.Blazor styles reference for overall NuGet package.

Also, you can referred the themes through the CDN version by using below link instead of package theme reference.

```
[https://cdn.syncfusion.com/blazor/{{ site.blazorversion
}}/styles/bootstrap4.css](https://cdn.syncfusion.com/blazor/{{ site.blazorversion
}}/styles/bootstrap4.css).
```

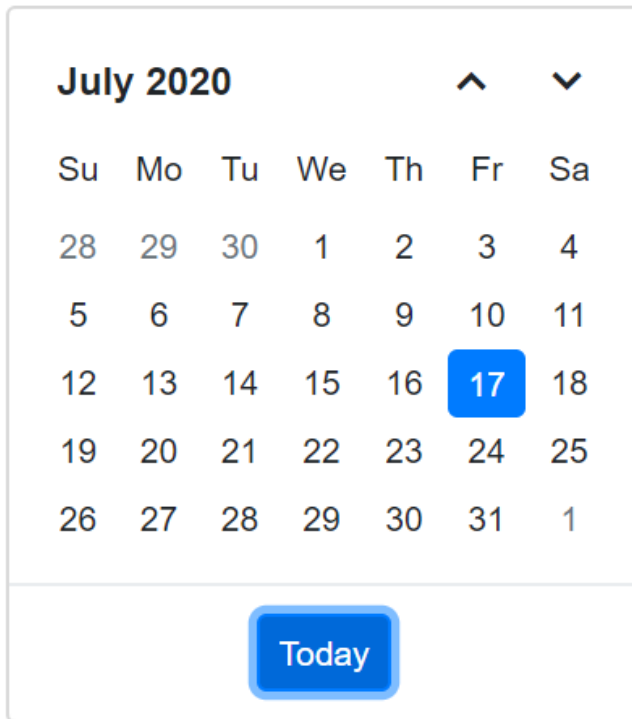
- Now, add the created custom component that is imported with Syncfusion Blazor component from Razor Class Library in any web page (razor) in the `~/Pages` folder. For example, the custom component with imported Syncfusion Blazor Calendar component from Razor Class Library is added to the `~/Pages/Index.razor` page as like below.

ASPX-CS

```
<Component></Component>
```

5. Run the application, The Syncfusion Blazor Calendar component will be rendered in the default web browser.

This Blazor component is defined in the **RazorClassLibrary** package.

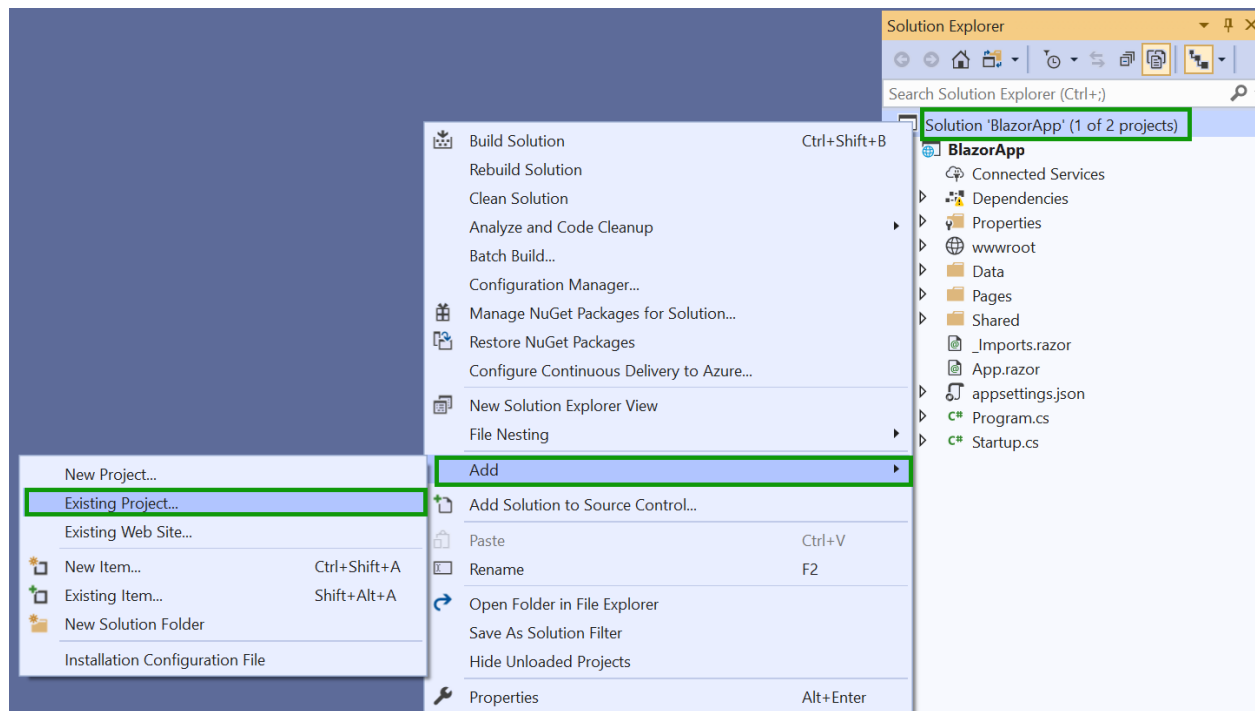


Create a Blazor WebAssembly project in Visual Studio with Razor Class Library (RCL)

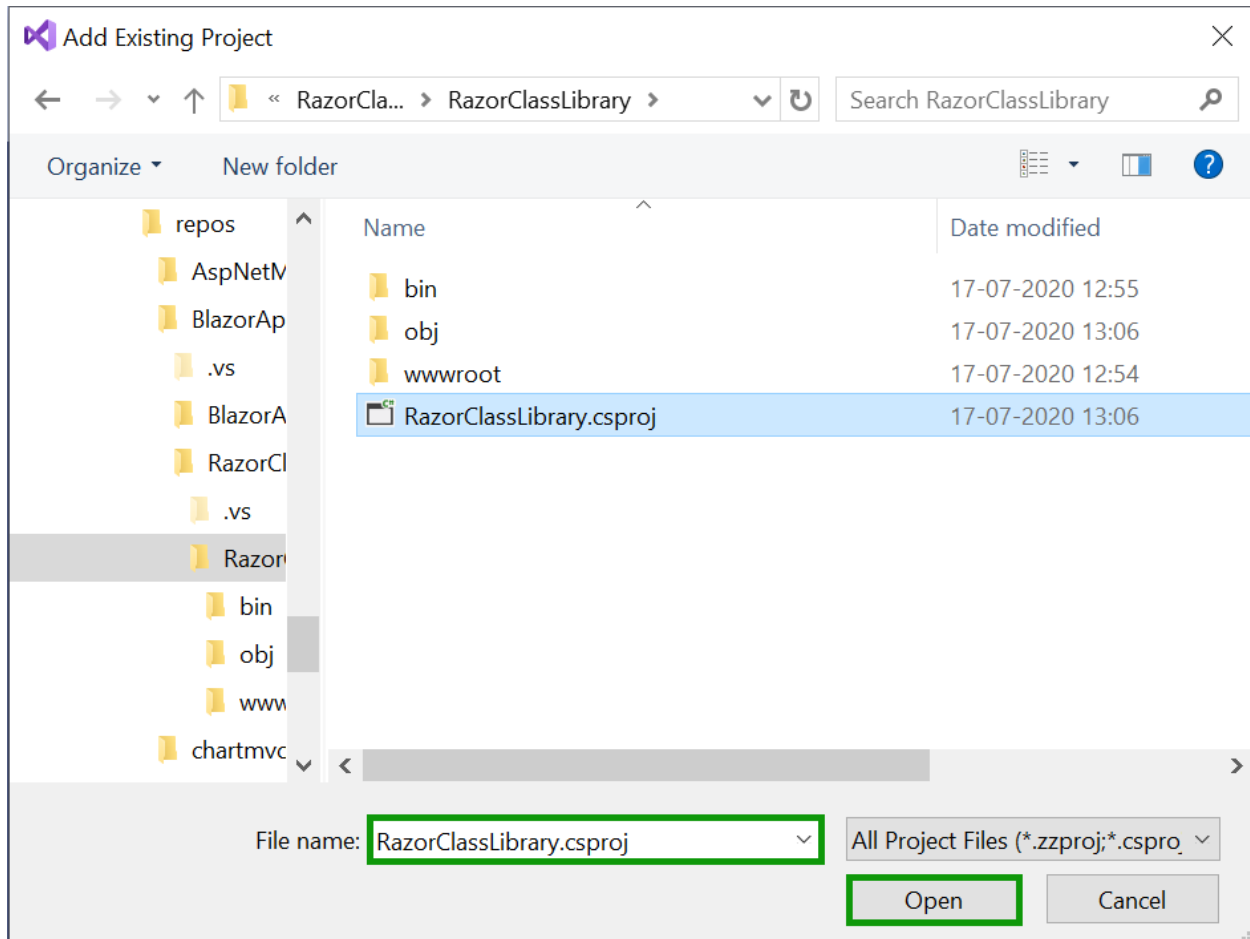
Refer to the [Blazor Tooling documentation](#) to create a new Blazor Server-Side Application using Visual Studio.

Configure the Razor Class Library and Blazor WebAssembly Application

1. Now, Right-click the solution, and then select Add/Existing Project.

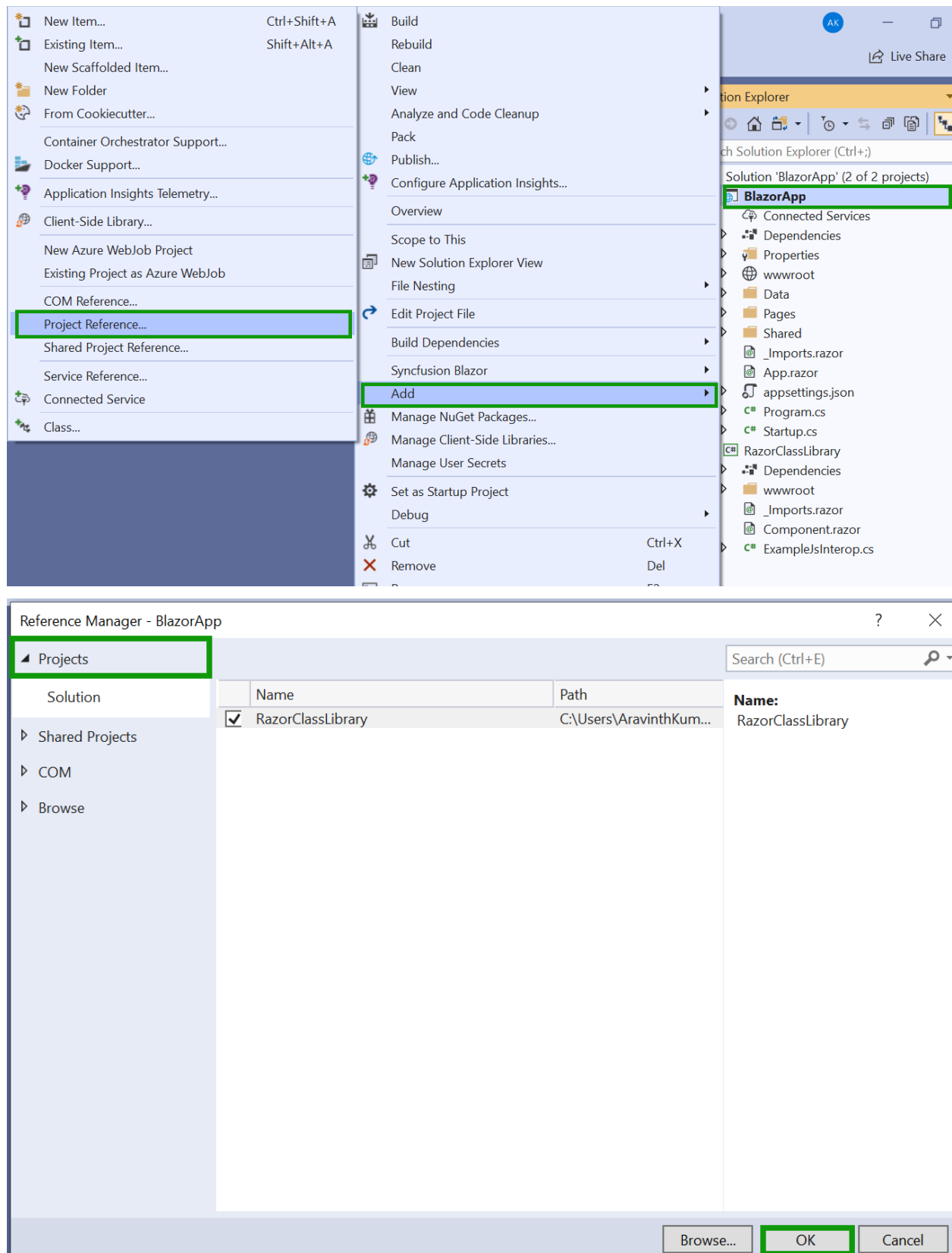


2. Add the **Razor Class Library** project by selecting `RazorClassLibrary.csproj` file.



Razor Class Library project is added to the existing Blazor WebAssembly Application.

3. Right-click the Blazor App project, and then select Add/Project reference. Now click the checkbox and configure the **Razor Class Library** and **Blazor WebAssembly Application**.



Importing Razor Class Library in the Blazor WebAssembly Application

1. Open `~/_Imports.razor` file in Blazor WebAssembly App and import the `RazorClassLibrary`.

ASPX-CS

```
@using RazorClassLibrary
```

2. Open the `~/Program.cs` file and register the Syncfusion Blazor Service from RCL.

C#

```
// For .NET 6 project.
using Microsoft.AspNetCore.Components;
using Microsoft.AspNetCore.Components.Web;
using Syncfusion.Blazor;
var builder = WebApplication.CreateBuilder(args);
builder.Services.AddRazorPages();
builder.Services.AddServerSideBlazor();
....
builder.Services.AddSyncfusionBlazor();
var app = builder.Build();
....
....
```

C#

```
// For .NET 5 or .NET Core SDK 3.1 project.
using Syncfusion.Blazor;
namespace BlazorApp
{
    public class Program
    {
        public static async Task Main(string[] args)
        {
            ....
            ....
            builder.Services.AddSyncfusionBlazor();
            await builder.Build.RunAsync();
        }
    }
}
```

3. Add the Syncfusion bootstrap4 theme in the `element` of the `~/wwwroot/index.html` page.

HTML

```
<head>
....
....
// Using individual NuGet packages
```

```
<link href="_content/Syncfusion.Blazor.Themes/bootstrap4.css"
rel="stylesheet" />
// (or)
// Using overall NuGet package
<link href="_content/Syncfusion.Blazor/styles/bootstrap4.css"
rel="stylesheet" />
</head>
```

Warning: Syncfusion.Blazor package should not to be installed along with [individual NuGet packages](#). If you are using individual NuGet packages, you have to add the above Syncfusion.Blazor.Themes static web assets (styles) reference in the application. Or else, you have to add the above Syncfusion.Blazor styles reference for overall NuGet package.

Also, you can referred the themes through the CDN version by using below link instead of package theme reference.

```
[https://cdn.syncfusion.com/blazor/{{ site.blazorversion
}}/styles/bootstrap4.css](https://cdn.syncfusion.com/blazor/{{ site.blazorversion
}}/styles/bootstrap4.css).
```

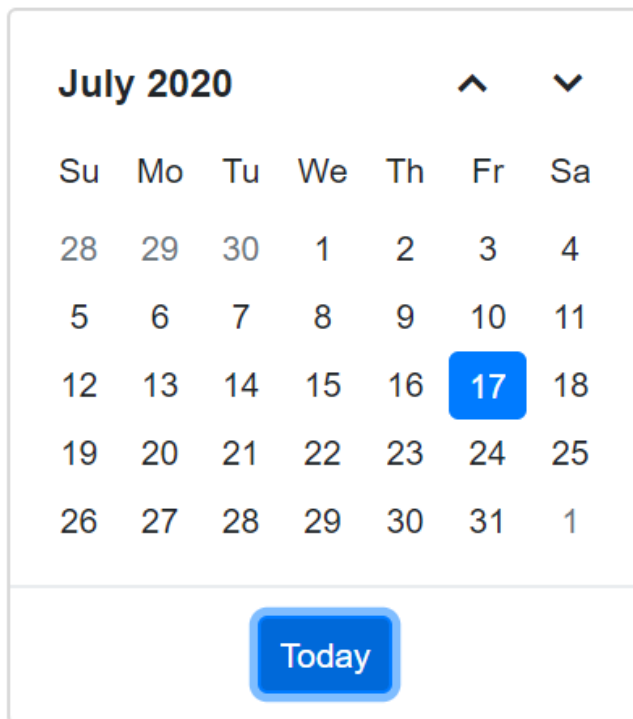
- Now, add the created custom component that is imported with Syncfusion Blazor component from Razor Class Library in any web page (razor) in the ~/Pages folder. For example, the custom component with imported Syncfusion Blazor Calendar component from Razor Class Library is added to the ~/Pages/Index.razor page as like below.

ASPX-CS

```
<Component></Component>
```

- Run the application, The Syncfusion Blazor Calendar component will be rendered in the default web browser.

This Blazor component is defined in the **RazorClassLibrary** package.



License Key

Syncfusion Licensing Overview

Syncfusion license key registration is applicable for all evaluators and only to paid customers who use NuGet packages from [NuGet.org](https://www.nuget.org). So, if you use the evaluation installer or the NuGet feed to reference Syncfusion assemblies, you must also include the corresponding platform and version license key in your projects

Following licensing error will be shown if the license key is not registered in your projects, while using assemblies from evaluation installer or from the [nuget.org](https://www.nuget.org).

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.

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 just a string that needs to be registered in your project to avoid licensing warning.

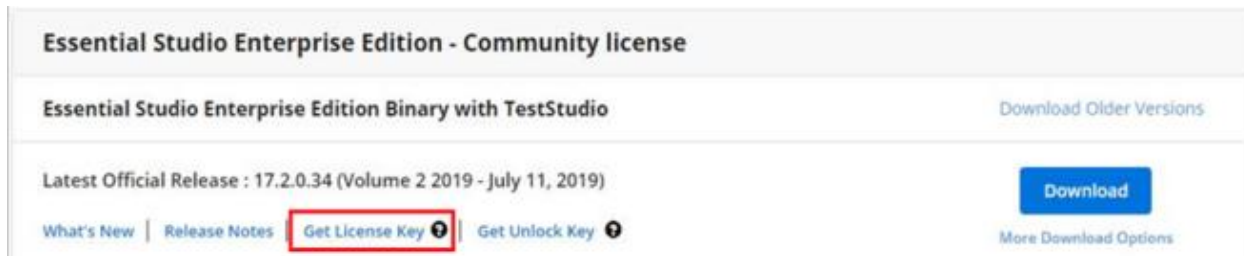
Refer to [this](#) KB article to know more about difference between the Syncfusion Unlock Key and the Syncfusion License Key.

See Also

- [How to Generate Syncfusion Blazor License Key?](#)
- [How to Register Syncfusion License Key in Blazor Application?](#)
- [How to Register Syncfusion License Key in Razor Class Library Application?](#)

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

Register Syncfusion License key in Blazor application

Syncfusion license key should be registered, if your project using Syncfusion Blazor packages reference from [nuget.org](#) or from trial installer. The generated license key is just a string that needs to be registered before any Syncfusion control is initiated. The following code is used to register the license.

CSHARP

```
Syncfusion.Licensing.SyncfusionLicenseProvider.RegisterLicense("YOUR LICENSE KEY");
```

Note:

Place the license key between double quotes. Also, ensure that Syncfusion.Licensing.dll is referenced in your project where the license key is being registered.

For Server side application

Register the license key in Configure method of Startup.cs

CSHARP

```
// This method gets called by the runtime. Use this method to configure the HTTP request pipeline.
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    // Register Syncfusion license
    Syncfusion.Licensing.SyncfusionLicenseProvider.RegisterLicense("YOUR LICENSE KEY");
}
```

```
if (env.IsDevelopment())
{
    app.UseDeveloperExceptionPage();
}
else
{
    app.UseExceptionHandler("/Error");
    // The default HSTS value is 30 days. You may want to change this for
    // production scenarios, see https://aka.ms/aspnetcore-hsts.
    app.UseHsts();
}
app.UseHttpsRedirection();
app.UseStaticFiles();
app.UseRouting();
app.UseEndpoints(endpoints =>
{
    endpoints.MapBlazorHub();
    endpoints.MapFallbackToPage("/_Host");
});
}
```

For Server side application using .NET 6.0

Register the license key in the Program.cs file if you created the Blazor server side application with Visual Studio 2022 and .NET 6.0.

CSHARP

```
var app = builder.Build();
//Register Syncfusion license
Syncfusion.Licensing.SyncfusionLicenseProvider.RegisterLicense("YOUR LICENSE KEY");
// Configure the HTTP request pipeline.
if (!app.Environment.IsDevelopment())
{
    app.UseExceptionHandler("/Error");
    // The default HSTS value is 30 days. You may want to change this for
    // production scenarios, see https://aka.ms/aspnetcore-hsts.
    app.UseHsts();
}
```

For Client side application

Register the license key in main method of Program.cs

CSHARP

```
using Syncfusion.Blazor;
public static async Task Main(string[] args)
{
    // Register Syncfusion license
    Syncfusion.Licensing.SyncfusionLicenseProvider.RegisterLicense("YOUR LICENSE KEY");
    var builder = WebAssemblyHostBuilder.CreateDefault(args);
    builder.RootComponents.Add<App>("app");
    builder.Services.AddSyncfusionBlazor();
    await builder.Build().RunAsync();
}
```

Register Syncfusion license key in a Razor Class Library application

The generated license key is just a string that might be registered before any Syncfusion control is initiated. The following code is used to register the license.

CSHARP

```
Syncfusion.Licensing.SyncfusionLicenseProvider.RegisterLicense("YOUR LICENSE KEY");
```

Note:

Place the license key between double quotes. Also, ensure that Syncfusion.Licensing.dll is referenced in your project where the license key is being registered.

If your Razor Class Library (RCL) project uses Syncfusion Blazor packages from nuget.org or the trial installer, you must register your license key. We need to register the license key in RCL project similar to how we do for the Blazor project based on your application type (Server application / Client WebAssembly application).

Refer to this [link](#) for more information on getting started with the Syncfusion Blazor components in the RCL project.

For Server side application

After configuring the RCL project with your Blazor Server application, register the license key in Configure method of Startup.cs

Refer to this [link](#) for more information on getting started with RCL in the Blazor Server application.

CSHARP

```
using Syncfusion.Blazor;  
  
// This method gets called by the runtime. Use this method to configure the  
// HTTP request pipeline.  
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)  
{  
    // Register Syncfusion license  
    Syncfusion.Licensing.SyncfusionLicenseProvider.RegisterLicense("YOUR LICENSE KEY");  
    if (env.IsDevelopment())  
    {  
        app.UseDeveloperExceptionPage();  
    }  
    else  
    {  
        app.UseExceptionHandler("/Error");  
        // The default HSTS value is 30 days. You may want to change this for  
        // production scenarios, see https://aka.ms/aspnetcore-hsts.  
        app.UseHsts();  
    }  
    app.UseHttpsRedirection();  
    app.UseStaticFiles();  
    app.UseRouting();  
    app.UseEndpoints(endpoints =>  
    {  
        endpoints.MapBlazorHub();  
    });  
}
```

```
endpoints.MapFallbackToPage("/_Host");
});
}
```

For Client side application

After configuring the RCL project with your Blazor Client WebAssembly application, register the license key in main method of Program.cs

Refer to this [link](#) for more information on getting started with RCL in the Blazor Client WebAssembly application.

CSHARP

```
using Syncfusion.Blazor;
public static async Task Main(string[] args)
{
    // Register Syncfusion license
    Syncfusion.Licensing.SyncfusionLicenseProvider.RegisterLicense("YOUR LICENSE KEY");
    var builder = WebAssemblyHostBuilder.CreateDefault(args);
    builder.RootComponents.Add<App>("app");
    builder.Services.AddSyncfusionBlazor();
    await builder.Build().RunAsync();
}
```

Syncfusion Licensing Errors

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

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 blazor components through trial installer or obtained our components via [NuGet.org](https://www.nuget.org), 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](#).

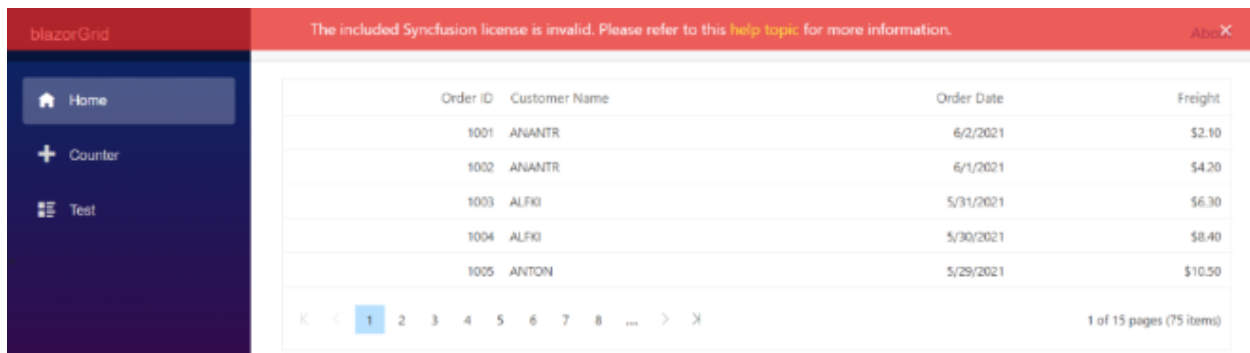


2. If you **have a Syncfusion account and an active trial**, you can **generate the trial license key for a specific version and platform** from [this page](#).
3. If you **have a Syncfusion account but no active trials**, [purchase a license](#) or [start your 30-day free trial](#). Then you can generate 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](#).

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.

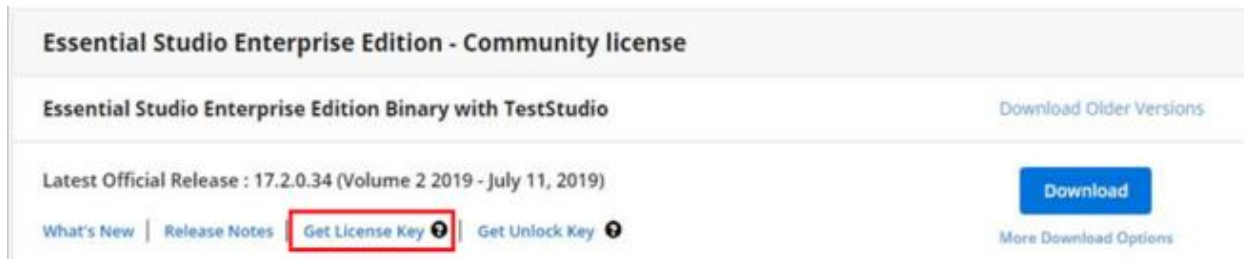
Error message:
 The included Syncfusion license is invalid. Please refer to this [help topic](#) for more information.



Solution:

If you use blazor components through trial installer or obtained our components via [NuGet.org](#), 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](#).

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.



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

Platform Mismatch

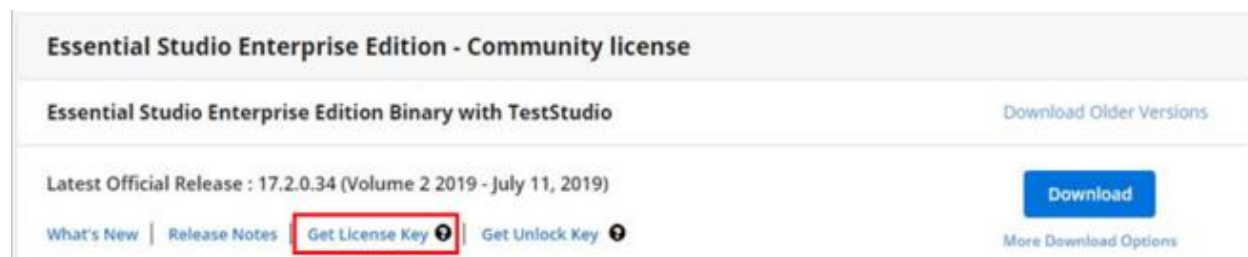
If the application is registered with another platform's license key, the following error message will pop up when launching the application.

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

**Solution:**

License keys are version and product specific. So, if you use blazor components through trial installer or obtained our components via [NuGet.org](https://nuget.org), 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](#).

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.



Solution:

License keys are version and product specific. So, if you use blazor components through trial installer or obtained our components via [NuGet.org](https://nuget.org), 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](#).

Facing licensing error message even after registering proper license keys

1. Ensure that respective version [Syncfusion.Licensing](#) nuget package or assembly is referred properly in the application.
2. Ensure that all the Syncfusion assemblies referenced are of the same version and that the corresponding version and product's license key is registered in the application.
3. The license key should be registered before initializing any Syncfusion controls in the application. Refer [this](#) page for more information on registering license keys.
4. Same version Syncfusion assemblies should be present in the application output folders or published folders.
5. If you've upgraded the Syncfusion version and license keys in the application, try cleaning and rebuilding the application to see if that resolves the license error message.

Syncfusion Licensing FAQ

How to upgrade from Trial version after purchasing a license

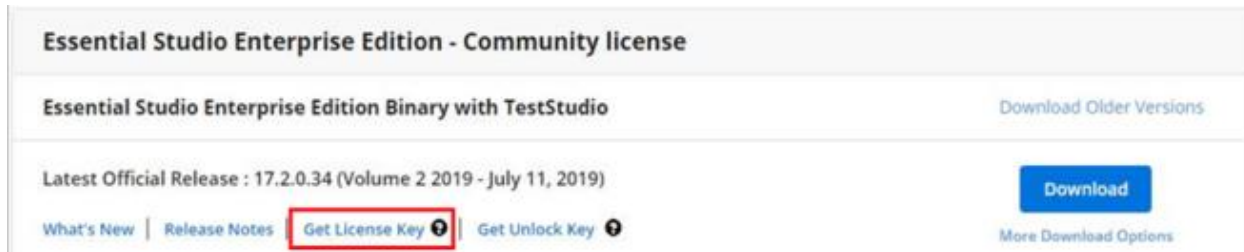
To upgrade from trial version, there are two possible solutions.

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

Note License registration is not required if you reference Syncfusion assemblies from Licensed installer. These licensing changes applicable to all evaluators who refers the Syncfusion assemblies from evaluation installer and those who use Syncfusion NuGet packages from nuget.org.

Where can I get a 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 this [KB](#) to know about which version of the Syncfusion license key should be used in the application.

Registering Syncfusion account for direct NuGet.org user

If you have directly obtained Syncfusion assemblies from [NuGet.org](https://nuget.org) and do not have a Syncfusion account, follow the steps to obtain a free 30-day trial license key:

- Register for a free Syncfusion account [here](#)
- Go to the start trials [page](#) and start a trial
- Finally proceed to the [Trial & Downloads section](#) to obtain the [license key](#)

Appearance

Blazor Themes in Syncfusion Components

The following list of themes are included in the Syncfusion Blazor components library.

Theme	Style Sheet Name
-----	-----
Bootstrap 5	bootstrap5.css
Bootstrap 5 Dark	bootstrap5-dark.css
Bootstrap 4	bootstrap4.css
Bootstrap 3	bootstrap.css
Bootstrap 3 Dark	bootstrap-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 |

The Syncfusion Blazor Bootstrap Theme is designed based on Bootstrap v3, whereas the Bootstrap4 theme is designed based on Bootstrap v4.

Reference themes in Blazor application

Syncfusion Blazor themes can be used in your Blazor application by referencing the style sheet.

- For **Blazor WebAssembly application**, refer style sheet inside the `wwwroot` element of

`wwwroot/index.html` file.

- For **Blazor Server application**, refer style sheet inside the `wwwroot` element of
- `~/Pages/_Host.cshtml` file for .NET 3 and .NET 5.
- `~/Pages/_Layout.cshtml` for .NET 6.

Using the below approaches the themes can be referenced in the Blazor application,

1. [Static Web assets](#) - Used to reference complete css via static web assets.
2. [CDN](#) - Used to reference complete css via static web assets.
3. [CRG](#) - Used to generate resources only for the selected (used) components.
4. [Theme Studio](#) - Used to customize and generate themes only for the selected (used) components.
5. [NPM packages](#) - Used to customize the existing themes and bundle stylesheet's in an application.
6. [LibMan](#) - Used to customize the existing themes and bundle stylesheet's in an application.

Instead of using [Static Web assets](#) or a [CDN reference](#), you can reference the style sheet into your projects to customize the theme or bundle it with the other style sheets using [NPM packages](#) or [LibMan](#).

Static Web Assets

Syncfusion Blazor themes are available as Static web Assets in the [Syncfusion.Blazor](#) and [Syncfusion.Blazor.Themes](#) NuGet Packages.

- For **Blazor WebAssembly application**, refer style sheet inside the `wwwroot` element of

`wwwroot/index.html` file.

- For **Blazor Server application**, refer style sheet inside the `wwwroot` element of
- `~/Pages/_Host.cshtml` file for .NET 3 and .NET 5.
- `~/Pages/_Layout.cshtml` for .NET 6.

When using individual NuGet packages in your application, add [Syncfusion.Blazor.Themes](#) NuGet Package and reference style sheet as below,

HTML

```
<head>
<link href="_content/Syncfusion.Blazor.Themes/bootstrap5.css"
rel="stylesheet" />
</head>
```

When using [Syncfusion.Blazor](#) NuGet package,

HTML

```
<head>
<link href="_content/Syncfusion.Blazor/styles/bootstrap5.css"
rel="stylesheet" />
</head>
```

CDN Reference

Instead of using a local resource on your server, you can use a cloud CDN to reference the theme style sheets. CDN Stands for "Content Delivery Network". A CDN is a group of servers distributed in different locations. While CDNs are often used to host websites, they are commonly used to provide other types of downloadable data as well. Examples include software programs, images, videos, and streaming media.

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

HTML

```
<head>
<link href="https://cdn.syncfusion.com/blazor/{{ site.blazorversion
}}/styles/bootstrap5.css" rel="stylesheet"/>
</head>
```

Theme Name	CDN Reference
Bootstrap 5	https://cdn.syncfusion.com/blazor/{{ site.blazorversion }}/styles/bootstrap5.css
Bootstrap 5 Dark	https://cdn.syncfusion.com/blazor/{{ site.blazorversion }}/styles/bootstrap5-dark.css
Bootstrap 4	https://cdn.syncfusion.com/blazor/{{ site.blazorversion }}/styles/bootstrap4.css
Bootstrap 3	https://cdn.syncfusion.com/blazor/{{ site.blazorversion }}/styles/bootstrap.css
Bootstrap 3 Dark	https://cdn.syncfusion.com/blazor/{{ site.blazorversion }}/styles/bootstrap-dark.css
Google's Material	https://cdn.syncfusion.com/blazor/{{ site.blazorversion }}/styles/material.css
Google's Material Dark	https://cdn.syncfusion.com/blazor/{{ site.blazorversion }}/styles/material-dark.css
Tailwind CSS	https://cdn.syncfusion.com/blazor/{{ site.blazorversion }}/styles/tailwind.css
Tailwind Dark CSS	https://cdn.syncfusion.com/blazor/{{ site.blazorversion }}/styles/tailwind-dark.css

| Microsoft Office Fabric | <https://cdn.syncfusion.com/blazor/{{ site.blazorversion }}/styles/fabric.css> |

| Microsoft Office Fabric Dark | <https://cdn.syncfusion.com/blazor/{{ site.blazorversion }}/styles/fabric-dark.css> |

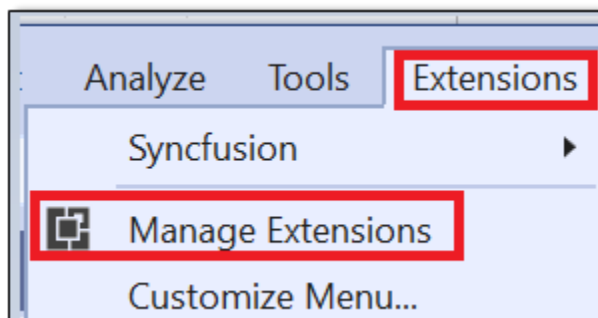
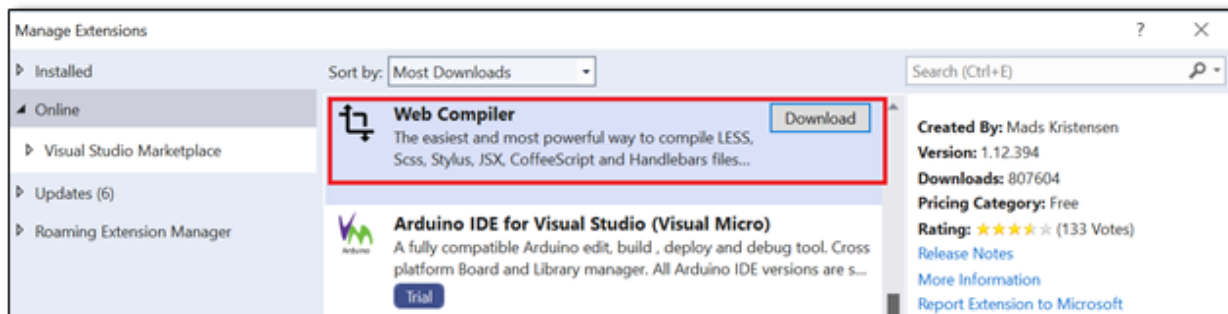
| High Contrast | <https://cdn.syncfusion.com/blazor/{{ site.blazorversion }}/styles/highcontrast.css> |

NPM Packages

NPM is a node package manager. It is basically used for managing dependencies of various server-side dependencies. You can manage server-side dependencies manually. It is a command-line program for dealing with said repository that aids in package installation, version management, and dependency management. It is an online repository for the publishing of open-source Node.js projects.

You can add the theme for the Blazor applications through **npm packages** using the **SCSS** files by following the below process.

- Install Web Compiler to use **SCSS** files in Blazor applications.
- To install Web Compiler, open Visual Studio and click the **Extensions** in the toolbar.

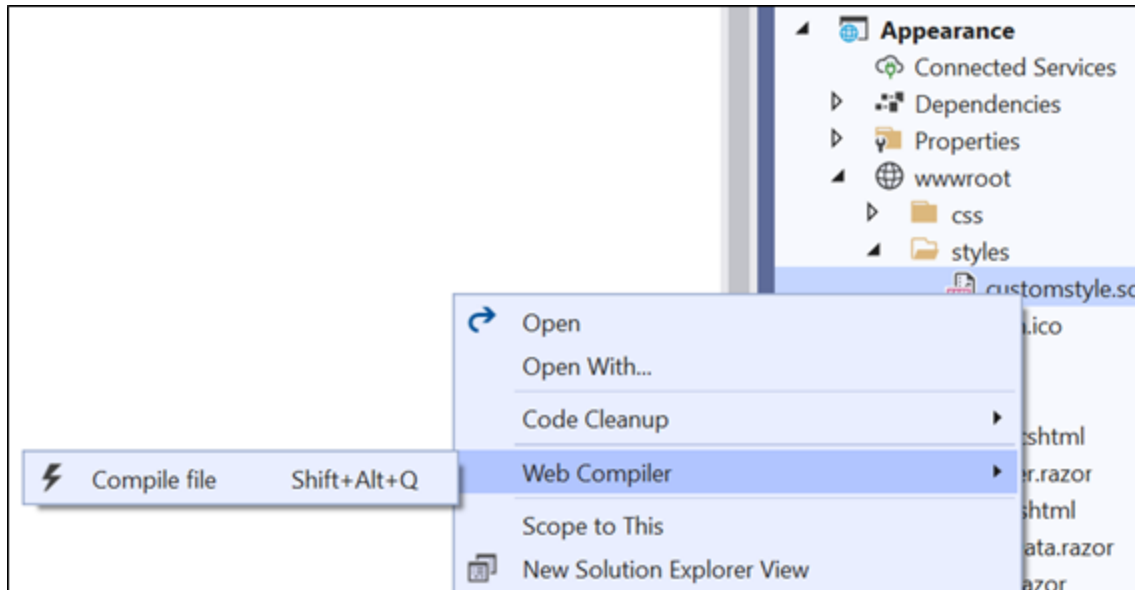


- Install the Syncfusion **node_modules** in this application using this command.

SCSS

```
$primary: blue !default;  
@import 'ej2/fabric.scss';
```

- Right-click the **SCSS** file and click the Web Compiler to compile the file.



- The `compiler config.json` file is created. Then, provide the location of the compiled CSS file and include a path as shown in the following code snippet.

JSON

```
[
{
  "outputFile": "wwwroot/styles/customstyle.css",
  "inputFile": "wwwroot/styles/customstyle.scss",
  "options": {
    "includePath": "node_modules/@syncfusion"
  }
}
]
```

- The SCSS file has been compiled to the CSS file. Then, add this CSS file to the `element` of the `~/Pages/_Host.cshtml` page.
- Run the application and see the fabric themes from installed npm packages was applied.

LibMan

Library Manager (LibMan) is a client-side library acquisition tool that is simple to use. LibMan is a program that downloads popular libraries and frameworks from a file system or a content delivery network (CDN).

LibMan offers the following advantages,

1. Only the library files you need are downloaded.
2. Additional tooling, such as Node.js, npm, and WebPack, isn't necessary to acquire a subset of files in a library.
3. Files can be placed in a specific location without resorting to build tasks or manual file copying.

In the server application root, add the **lib man.json** file with the following content:

JSON

```
{
  "version": "1.0",
  "defaultProvider": "cdnjs",
  "libraries": [
    {
      "library": "@progress/Syncfusion-Blazor-bootstrap@latest",
      "destination": "wwwroot/css/Syncfusion-Blazor/bootstrap",
      "files": [
        "dist/all.css"
      ]
    },
    {
      "library": "@progress/Syncfusion-Blazor-fabric@latest",
      "destination": "wwwroot/css/Syncfusion-Blazor/fabric",
      "files": [
        "dist/all.css"
      ]
    },
    {
      "library": "@progress/Syncfusion-Blazor-material@latest",
      "destination": "wwwroot/css/Syncfusion-Blazor/material",
      "files": [
        "dist/all.css"
      ]
    }
  ]
}
```

In the client Blazor application, go to the **wwwroot/index.html** file and replace the CDN link with the following one. For a server-side Blazor project, do that in the **~/Pages/_Host.cshtml** file.

HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
<link href="/css/Syncfusion-Blazor/fabric/dist/all.css" rel="stylesheet" />
</head>
</html>
```

Change theme dynamically

In the Blazor application, the application theme can be changed dynamically by changing its style sheet reference in code.

Change theme dynamically in blazor server app

The following example demonstrates how to change a theme dynamically in Blazor Server application using Syncfusion Blazor themes using Syncfusion Dropdown component.

1. In **_Host.cshtml**, refer syncfusion style sheet where the style sheet name is defined based on query string.

ASPX-CS

```

@page "/"
@namespace BlazorThemeSwitcher.Pages
@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers
@{
    Layout = null;
    QueryHelpers.ParseQuery(Request.QueryString.Value).TryGetValue("theme", out
    var themeName);
    themeName = themeName.Count > 0 ? themeName.First() : "bootstrap";
}
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8" />
<meta name="viewport" content="width=device-width, initial-scale=1.0" />
<title>BlazorThemeSwitcher</title>
<base href="~/>
<link rel="stylesheet" href="css/bootstrap/bootstrap.min.css" />
<link href="css/site.css" rel="stylesheet" />
<link href="BlazorThemeSwitcher.styles.css" rel="stylesheet" />
<link href="@("_content/Syncfusion.Blazor.Themes/" + themeName +
".css")rel="stylesheet" />
</head>
<body>
<script src="_framework/blazor.server.js"></script>
</ >
</body>
</html>

```

2. In **MainLayout.razor** page add dropdown list with themes and in **ValueChange** event handler, the page is refreshed by changing query string to change the theme in application.

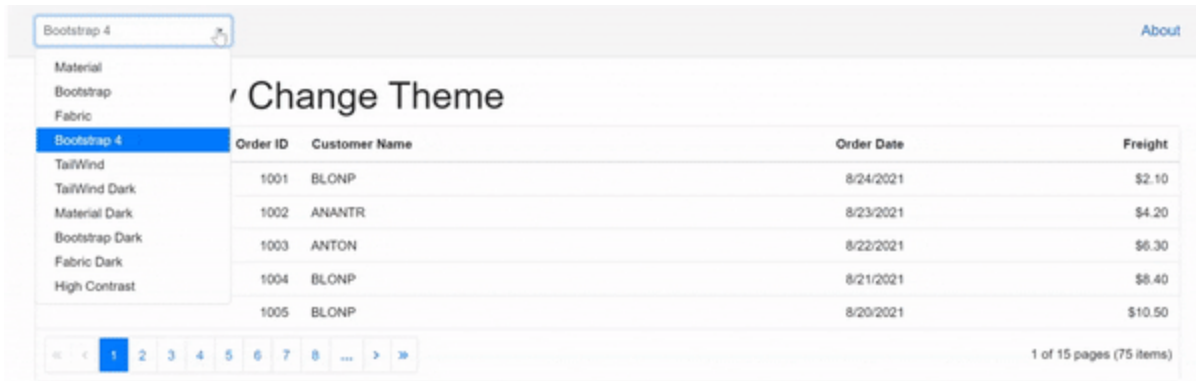
ASPX-CS

```

@inherits LayoutComponentBase
@inject NavigationManager UrlHelper;
@using Syncfusion.Blazor.DropDowns;
@using Syncfusion.Blazor.Buttons;
@using Microsoft.AspNetCore.WebUtilities
<div class="page">
<div class="main">
<div class="top-row px-4">
<div class="theme-switcher">
@*Theme switcher*@
<SfDropDownList TItem="ThemeDetails" TValue="string" @bind-Value="themeName"
DataSource="@Themes">
<DropDownListFieldSettings Text="Text"
Value="ID"></DropDownListFieldSettings>
<DropDownListEvents TItem="ThemeDetails" TValue="string"
ValueChange="OnThemeChange"></DropDownListEvents>
</SfDropDownList>
</div>
<a href="http://blazor.net" target="_blank" class="ml-md-auto">About</a>
</div>
<div class="content px-4">

```

```
@Body
</div>
</div>
</div>
@code {
private string themeName;
public class ThemeDetails
{
public string ID { get; set; }
public string Text { get; set; }
}
private List<ThemeDetails> Themes = new List<ThemeDetails>() {
new ThemeDetails(){ ID = "material", Text = "Material" },
new ThemeDetails(){ ID = "bootstrap", Text = "Bootstrap" },
new ThemeDetails(){ ID = "fabric", Text = "Fabric" },
new ThemeDetails(){ ID = "bootstrap4", Text = "Bootstrap 4" },
new ThemeDetails(){ ID = "tailwind", Text = "TailWind"},
new ThemeDetails(){ ID = "tailwind-dark", Text = "TailWind Dark" },
new ThemeDetails(){ ID = "material-dark", Text = "Material Dark" },
new ThemeDetails(){ ID = "bootstrap-dark", Text = "Bootstrap Dark" },
new ThemeDetails(){ ID = "fabric-dark", Text = "Fabric Dark" },
new ThemeDetails(){ ID = "highcontrast", Text = "High Contrast" }
};
public void OnThemeChange(ChangeEventArgs<string, ThemeDetails> args)
{
var theme = GetThemeName();
if (theme != args.ItemData.ID)
{
UrlHelper.NavigateTo(GetUri(args.ItemData.ID), true);
}
}
private string GetThemeName()
{
var uri = UrlHelper.ToAbsoluteUri(UrlHelper.Uri);
QueryHelpers.ParseQuery(uri.Query).TryGetValue("theme", out var theme);
theme = theme.Count > 0 ? theme.First() : "bootstrap";
return theme;
}
private string GetUri(string themeName)
{
var uri = UrlHelper.ToAbsoluteUri(UrlHelper.Uri);
return uri.AbsolutePath + "?theme=" + themeName;
}
protected override void OnInitialized()
{
var theme = GetThemeName();
themeName = theme.Contains("bootstrap4") ? "bootstrap4" : theme;
}
}
```

[View sample in GitHub](#)

Change theme dynamically in blazor WebAssembly (WASM) app

The following example demonstrates how to change a theme dynamically in Blazor WebAssembly using the application with the Syncfusion Blazor themes using Syncfusion Dropdown component.

1. Add the below function code in the **index.html** file to set the theme as selected in dropdown by using its **id** value.

HTML

```
<head>
.....
<link id="theme" href="_content/Syncfusion.Blazor.Themes/bootstrap4.css"
rel="stylesheet" />
</head>
.....
<script>
function setTheme(theme, isThemeDark) {
document.getElementsByTagName('body')[0].style.display = 'none';
let synclink = document.getElementById('theme');
synclink.href = '_content/Syncfusion.Blazor.Themes/' + theme + '.css';
setTimeout(function () {
document.getElementsByTagName('body')[0].style.display = 'block'; }, 200);
}
</script>
.....
```

2. Modify the **MainLayout.razor** page with the below code to implement a theme change dynamically using the dropdown by its id value in javascript function in the application.

ASPX-CS

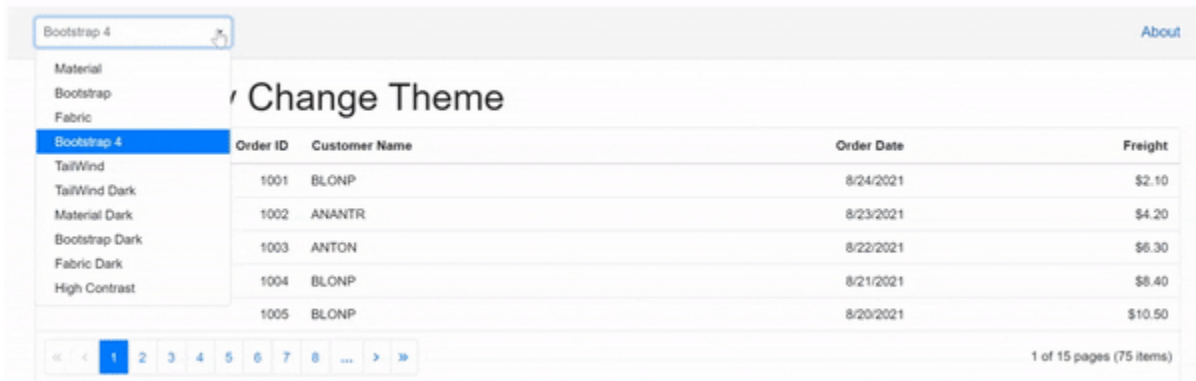
```
@inherits LayoutComponentBase
```

```

@inject NavigationManager UrlHelper;
@inject IJSRuntime JSRuntime;
@using Syncfusion.Blazor.DropDowns;
@using Syncfusion.Blazor.Buttons;
@using Microsoft.AspNetCore.WebUtilities;
<div class="page">
<div class="main">
<div class="top-row px-4">
<div class="theme-switcher">
@*Theme switcher*@
<SfDropDownList TItem="ThemeDetails" TValue="string" @bind-Value="themeName"
DataSource="@Themes">
<DropDownListFieldSettings Text="Text"
Value="ID"></DropDownListFieldSettings>
<DropDownListEvents TItem="ThemeDetails" TValue="string"
ValueChange="OnThemeChange"></DropDownListEvents>
</SfDropDownList>
</div>
<a href="http://blazor.net" target="_blank" class="ml-md-auto">About</a>
</div>
<div class="content px-4">
@Body
</div>
</div>
</div>
@code {
private string themeName;
public class ThemeDetails
{
public string ID { get; set; }
public string Text { get; set; }
}
private List<ThemeDetails> Themes = new List<ThemeDetails>() {
new ThemeDetails(){ ID = "material", Text = "Material" },
new ThemeDetails(){ ID = "bootstrap", Text = "Bootstrap" },
new ThemeDetails(){ ID = "fabric", Text = "Fabric" },
new ThemeDetails(){ ID = "bootstrap4", Text = "Bootstrap 4" },
new ThemeDetails(){ ID = "tailwind", Text = "TailWind"},
new ThemeDetails(){ ID = "tailwind-dark", Text = "TailWind Dark" },
new ThemeDetails(){ ID = "material-dark", Text = "Material Dark" },
new ThemeDetails(){ ID = "bootstrap-dark", Text = "Bootstrap Dark" },
new ThemeDetails(){ ID = "fabric-dark", Text = "Fabric Dark" },
new ThemeDetails(){ ID = "highcontrast", Text = "High Contrast" }
};
public void
OnThemeChange(Syncfusion.Blazor.DropDowns.ChangeEventArgs<string,
ThemeDetails> args)
{
JSRuntime.InvokeAsync<object>("setTheme", args.ItemData.ID);
}
private string GetThemeName()
{
var uri = UrlHelper.ToAbsoluteUri(UrlHelper.Uri);
QueryHelpers.ParseQuery(uri.Query).TryGetValue("theme", out var theme);
theme = theme.Count > 0 ? theme.First() : "bootstrap4";
return theme;
}
}

```

```
private string GetUri(string themeName)
{
    var uri = UrlHelper.ToAbsoluteUri(UrlHelper.Uri);
    return uri.AbsolutePath + "?theme=" + themeName;
}
protected override void OnInitialized()
{
    var theme = GetThemeName();
    themeName = theme.Contains("bootstrap4") ? "bootstrap4" : theme;
}
}
```



[View sample in GitHub](#)

Size Mode for Blazor Components

Syncfusion blazor components supports touch (bigger theme) and normal size modes. Below topics explains how to enable the same in your application.

Size mode for application

You can enable touch mode (bigger theme) for an application by adding `.e-bigger` class in `~/wwwroot/css/site.css` file and assign to the `body` element in `Pages/_Host.cshtml` (Blazor Server App) or `wwwroot/index.html` (Blazor WebAssembly App).

CSS

```
.e-bigger {
    font-size: x-large;
}
```

ASPX-CS

```
<body class="e-bigger">...</body>
```

Size mode for a control

You can enable touch mode (bigger theme) for a control by adding `.e-bigger` class and assign to the `div` which contains the control.

ASPX-CS

```
@page "/"
```

```
@using Syncfusion.Blazor.Calendars;
@using Syncfusion.Blazor.Buttons;
@using Syncfusion.Blazor.Popups;
<div class="e-bigger">
<SfCalendar TValue="DateTime?" Value="@DateValue"></SfCalendar>
</div>
<div class="e-bigger">
<SfButton> Button </SfButton>
</div>
<div class="e-bigger">
<SfCheckBox Label="checked" @bind-Checked="isChecked"></SfCheckBox>
</div>
<style>
.e-bigger {
font-size: x-large;
}
</style>
@code {
private bool isChecked = true;
public DateTime? DateValue { get; set; } = new DateTime(DateTime.Now.Year,
DateTime.Now.Month, 28);
}
```

Change size mode for application at runtime

You can change the size mode of an application between touch and normal (mouse) mode at runtime by adding and removing `.e-bigger` using `JavaScriptInterop`.

Follow below steps to change the size mode for an application at runtime.

1. Add the `e-bigger` CSS class in the `~/wwwroot/css/site.css` file.

CSS

```
.e-bigger {
font-size: x-large;
}
```

2. Add the following JavaScript methods inside the script tag of `wwwroot/index.html` (Blazor WebAssembly App) or `Pages/_Host.cshtml` (Blazor Server App) to switch between touch and mouse mode using `e-bigger` class.

ASPX-CS

```
<script>
function onTouch() {
document.body.classList.add('e-bigger');
}
function onMouse() {
document.body.classList.remove('e-bigger');
}
</script>
```

2. To call JavaScript method from .NET, inject the `IJSRuntime` abstraction and call `InvokeAsync` method as given in the below code,

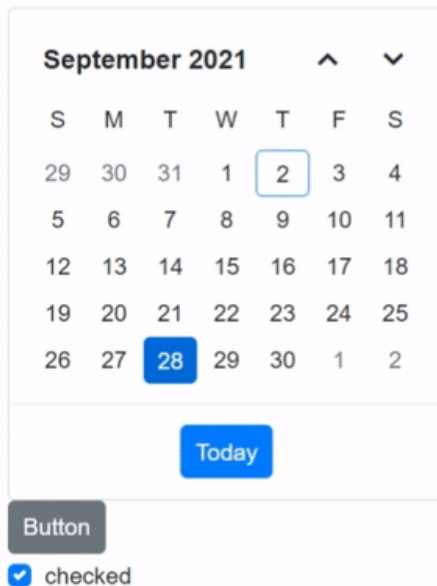
ASPX-CS

```
@page "/"
@using Syncfusion.Blazor.Calendars;
@using Syncfusion.Blazor.Buttons;
@using Syncfusion.Blazor.Popups
@inject IJSRuntime jsRuntime;
<p> Size-modes for application </p>
<p> This demo shows the Size-Modes applied for an entire application </p>
<button @onclick="callOnTouch">Touch Mode</button>
<button @onclick="callOnMouse">Mouse Mode</button>
<div>
<SfCalendar TValue="DateTime?" Value="@DateValue"></SfCalendar>
</div>
<div>
<SfButton> Button </SfButton>
</div>
<div>
<SfCheckBox Label="checked" @bind-Checked="isChecked"></SfCheckBox>
</div>
@code {
private bool isChecked = true;
private async void callOnTouch(MouseEventArgs args)
{
await jsRuntime.InvokeAsync<string>("onTouch");
}
private async void callOnMouse(MouseEventArgs args)
{
await jsRuntime.InvokeAsync<string>("onMouse");
}
}
```

Size-modes for application

This demo shows the Size-Modes applied for an entire application

Touch Mode Mouse Mode



[View sample in GitHub](#)

Change size mode for a control at runtime

You can change the size mode of a control between touch and normal (mouse) mode at runtime by setting `.e-bigger` CSS class.

Refer to the following code, in which the `e-bigger` class is added for enabling touch mode using the `check` variable.

ASPX-CS

```
@page "/"
@using Syncfusion.Blazor.Calendars;
@using Syncfusion.Blazor.Buttons;
@using Syncfusion.Blazor.Popups;
<h2>Syncfusion Component Size-Modes</h2>
<button @onclick="OnTouch">Touch Mode</button>
<button @onclick="OnMouse">Mouse Mode</button>
<div class="@touchCSS">
  <SfCalendar TValue="DateTime?" Value="@DateValue"></SfCalendar>
</div>
```

```
<div class="@touchCSS">
  <SfButton> Button </SfButton>
</div>
<div class="@touchCSS">
  <SfCheckBox Label="checked" @bind-Checked="isChecked"></SfCheckBox>
</div>
<style>
.e-bigger {
font-size: x-large;
}
</style>
@code {
private bool isChecked = true;
public DateTime? DateValue { get; set; } = new DateTime(DateTime.Now.Year,
DateTime.Now.Month, 28);
public string touchCSS { get; set; }
public void OnTouch()
{
touchCSS = "e-bigger";
}
public void OnMouse()
{
touchCSS = "";
}
}
```

Size-modes for application

This demo shows the Size-Modes applied for Syncfusion Control

Touch Mode

Mouse Mode

September 2021

^

v

S	M	T	W	T	F	S
29	30	31	1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	1	2

Today

Button

☒ checked

[View sample in GitHub](#)

See Also

Refer below topics to learn about responsiveness components based on available size in Syncfusion Blazor Components.

- [Sidebar Responsiveness](#)
- [DataGrid Responsiveness](#)

- [TreeGrid Responsiveness](#)
- [Dashboard Layout Responsiveness](#)
- [Kanban Responsiveness](#)
- [Toolbar Responsiveness](#)
- [Tab Responsiveness](#)

Theme Studio in Blazor Components

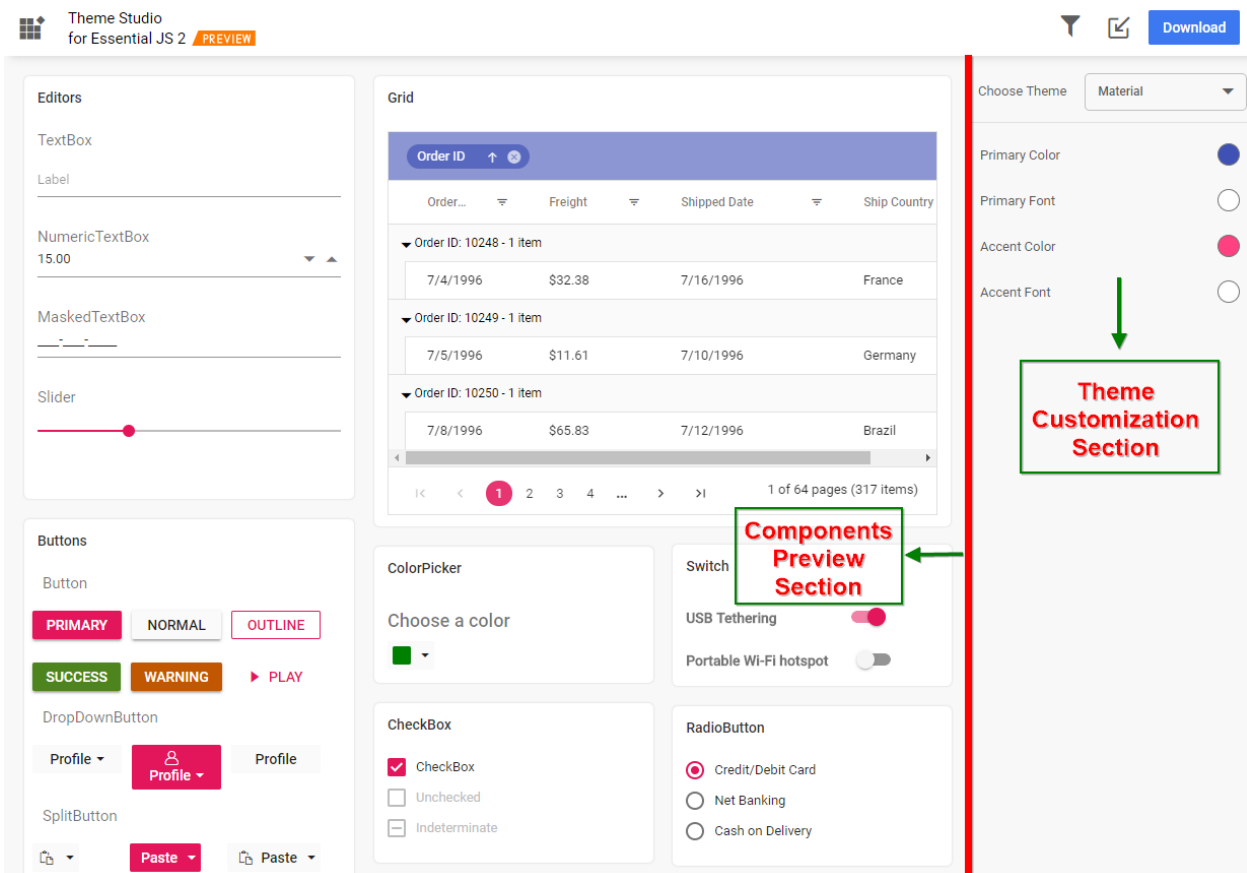
Theme Studio for Syncfusion Blazor can be used to customize a new theme from an existing theme. It does not support data visualization controls such as Chart, Diagram, Gauge, Range Navigator, and Maps.

Customizing theme color from theme studio

The Syncfusion Blazor 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 Blazor components. All Syncfusion Blazor 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.

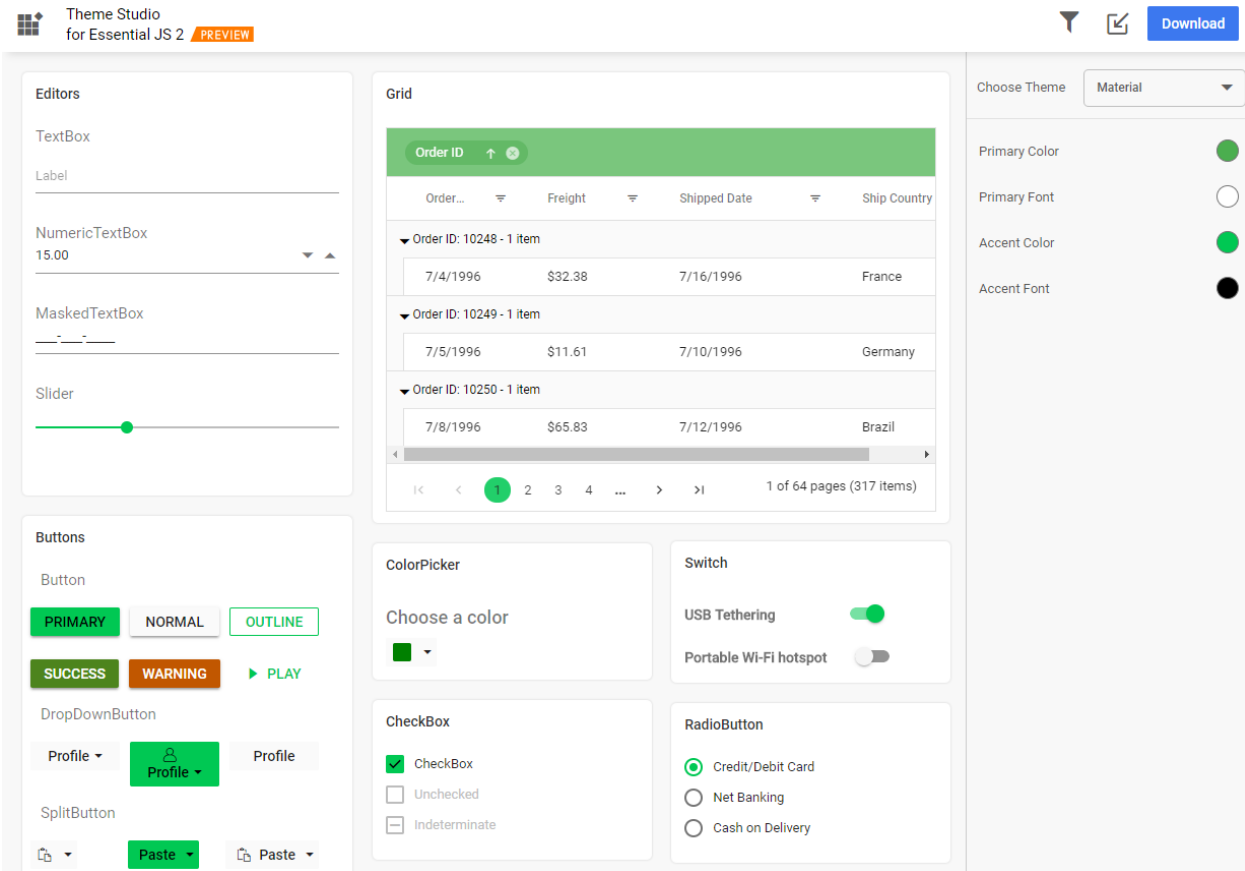
Step 1: Navigate to the Theme Studio application from this link: [Theme Studio](#).

Step 2: The Theme Studio application page can be divided into two sections: the controls 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.

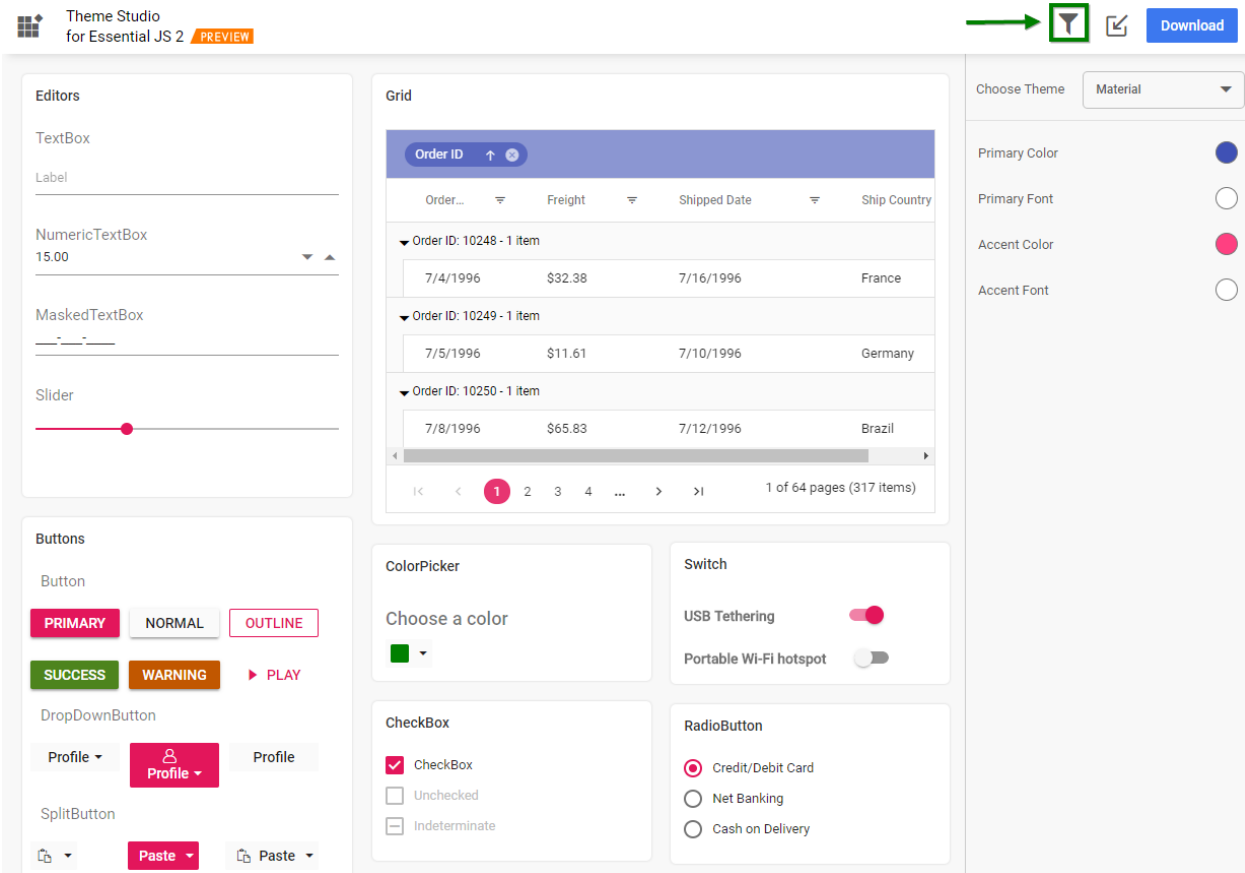
Step 4: The Syncfusion Blazor components will be rendered with the newly selected colors in the preview section after selecting a custom color from picker.



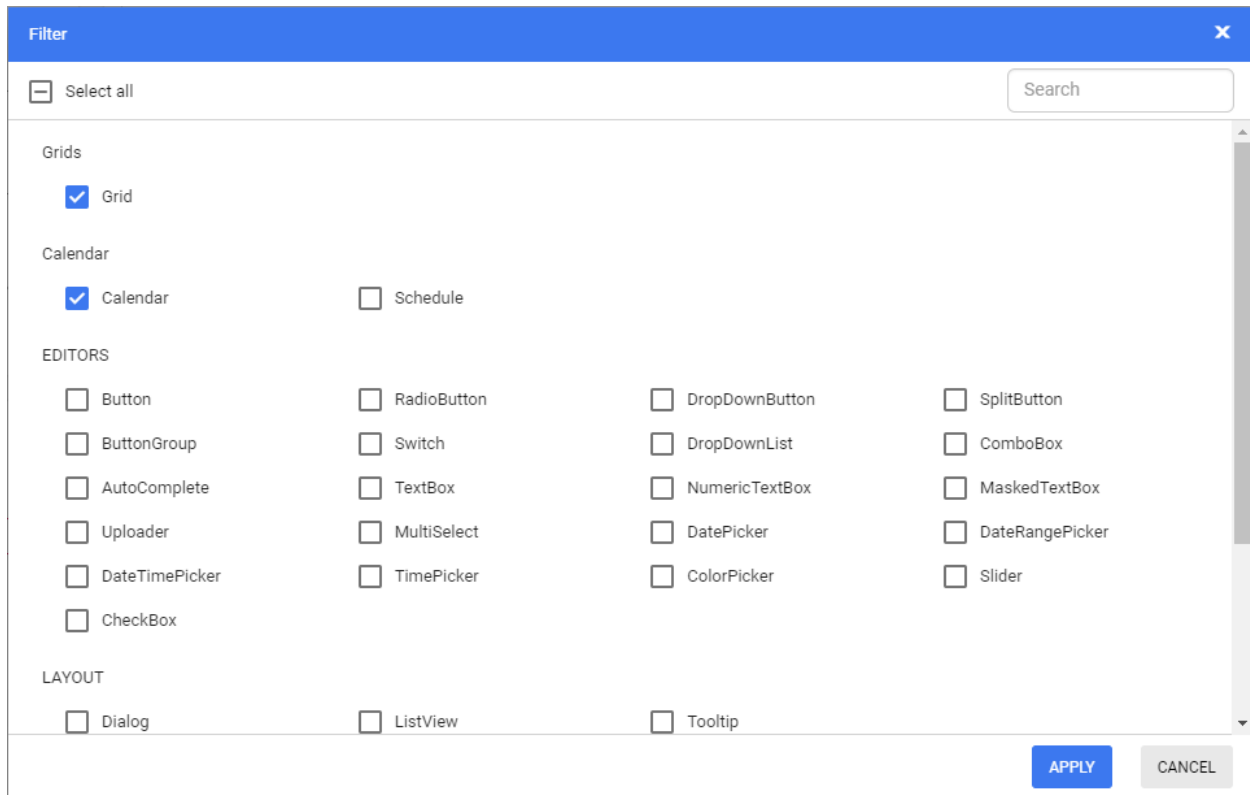
Filtering a specific list of controls

Using the theme studio, you can apply custom themes to a list of specific controls. This option is used when you integrate a selective list of Syncfusion Blazor components in your application. The theme studio will filter the selected controls and customize the final output for the controls' styles alone by reducing the final output file size.

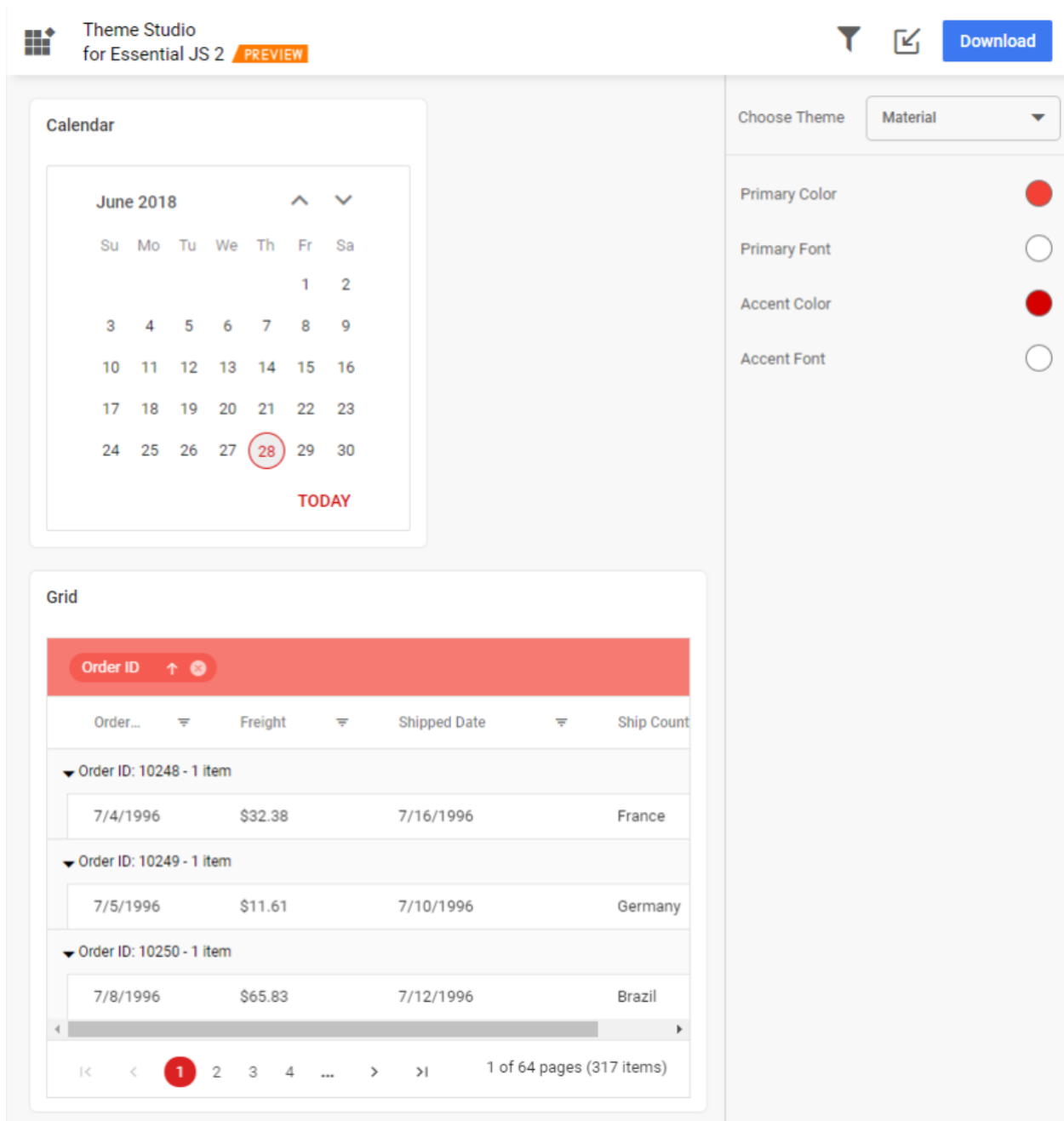
Step 1: Click the Filter icon at the top-right corner, and select the controls whose theme you want to customize.



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



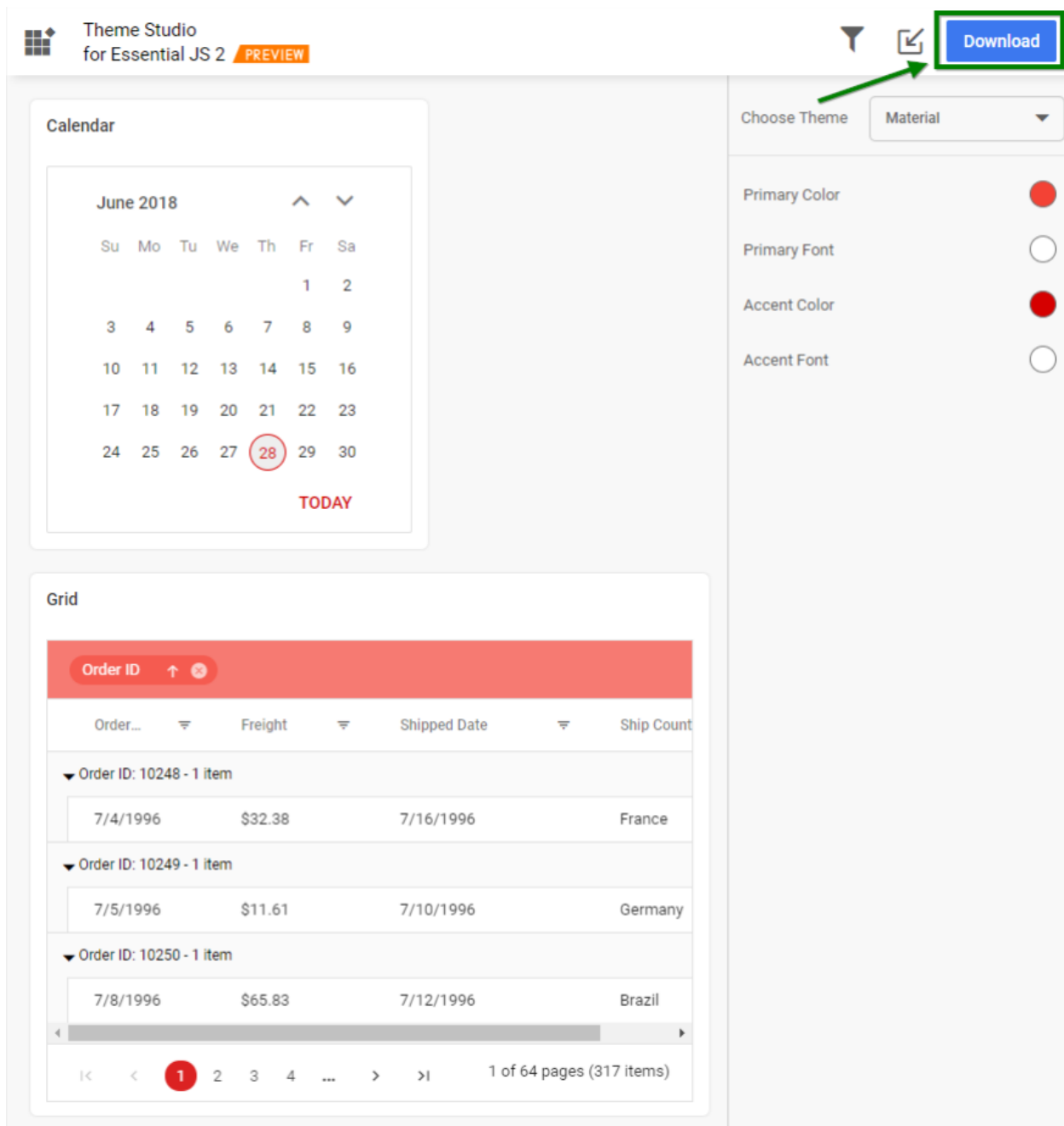
Step 3: Now, you can customize the colors in the theme customization section for the controls selected.



[Download the customized theme](#)

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

Step 1: Click the Download button at the top-right corner, and the Download dialog will appear.



Step 2: Assign a theme name in the File Name field, and click the Download button.

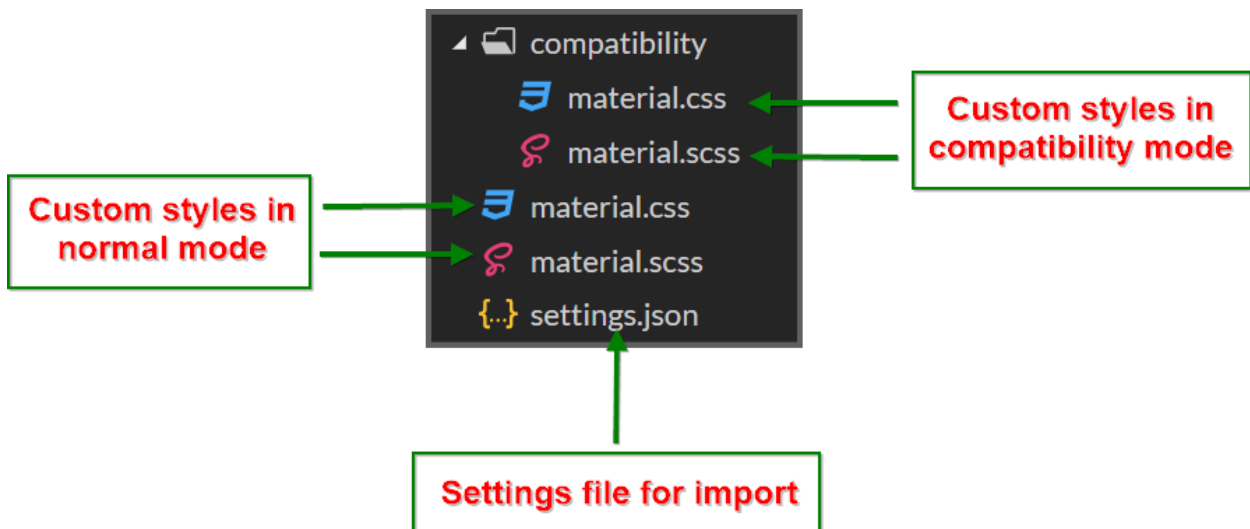
Download [X]

File Name
material

☐ Include compatibility css ⓘ

DOWNLOAD CANCEL

Step 3: The download styles will come as a zip file that contains SCSS and CSS files for the selected Syncfusion Blazor 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 and paste the customized CSS file from the download folder into any folder, e.g., `~/wwwroot/styles/{file-name}.css`.

Step 2: Refer the customized CSS file reference in the `~/wwwroot/index.html` or `~/Pages/_Host.cshtml` main page head section.

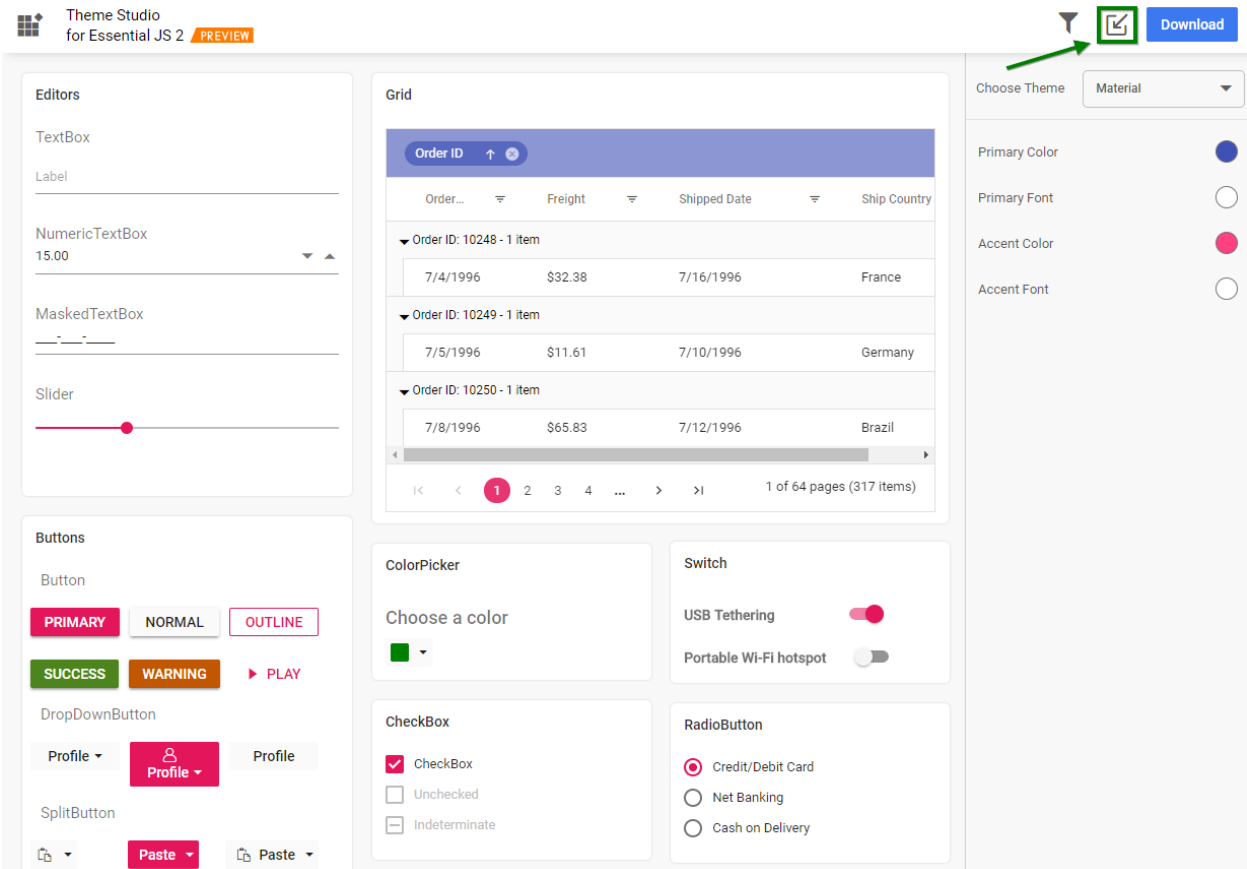
HTML

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

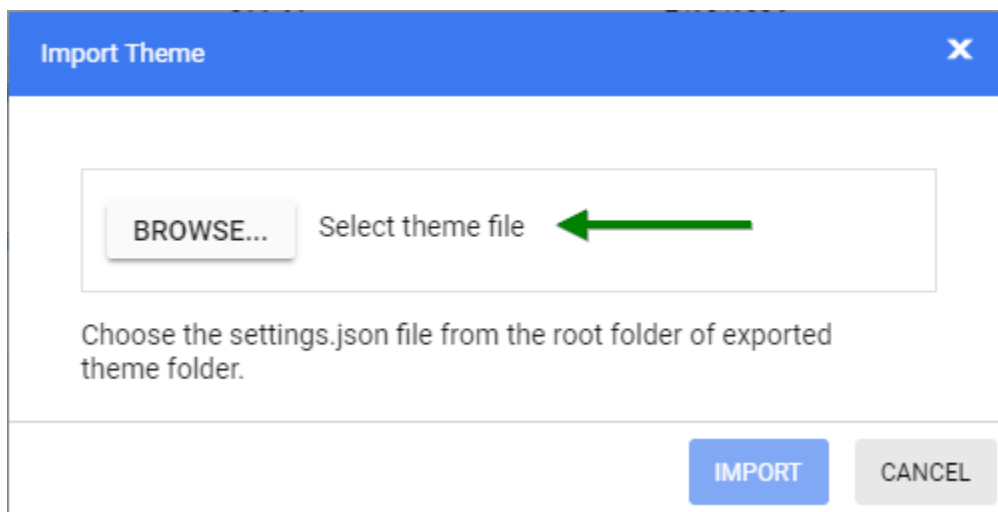

Import previously changed settings into theme studio

If you need to change your application theme and UI design in the future, do not customize the Syncfusion Blazor 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.

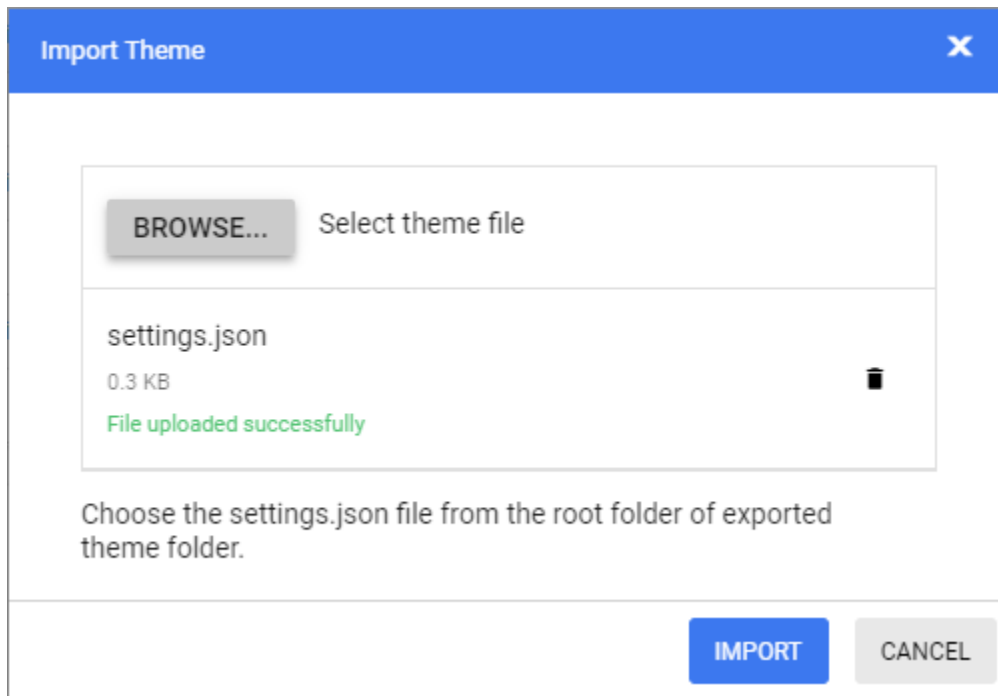
Step 1: Click the Import icon at the top-right corner.



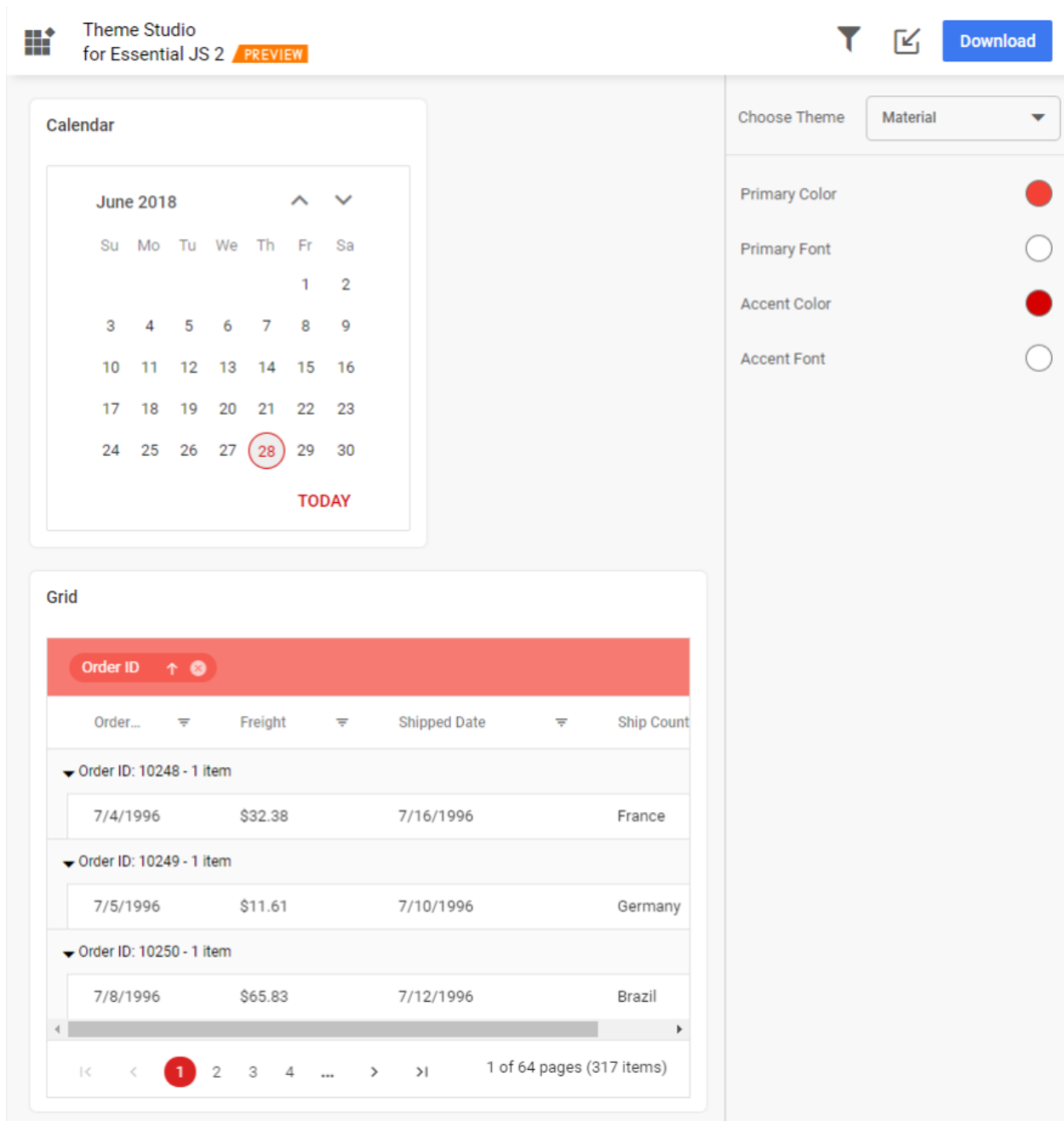
Step 2: The Import Theme dialog will open. Click the Browse button and select the `settings.json` file you exported previously.



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 contains your latest changes. You can just replace the older custom style with a new style to refresh your application.

Common Variables

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

Syncfusion Blazor 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
\$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

Syncfusion Blazor Bootstrap 4 Theme

Name	Value
\$white	#fff
\$gray-100	#f8f9fa
\$gray-200	#e9ecef
\$gray-300	#dee2e6
\$gray-400	#ced4da
\$gray-500	#adb5bd
\$gray-600	#6c757d
\$gray-700	#495057
\$gray-800	#343a40
\$gray-900	#212529
\$black	#000
\$blue	#007bff
\$indigo	#6610f2
\$purple	#6f42c1
\$pink	#e83e8c
\$red	#dc3545
\$orange	#fd7e14
\$yellow	#ffc107
\$green	#28a745
\$teal	#20c997
\$cyan	#17a2b8

Syncfusion Blazor Bootstrap Theme

Name	Value (Default Theme)	Value (Dark Theme)
\$brand-primary	#317ab9	#0070f0
\$brand-primary-darken-10	#3071a9	darken(\$brand-primary, 10%)
\$brand-primary-darken-15	#2a6496	darken(\$brand-primary, 20%)
\$brand-primary-darken-25	#1f496e	darken(\$brand-primary, 30%)
\$brand-primary-darken-35	#142f46	darken(\$brand-primary, 40%)
\$brand-primary-font	#fff	#fff
\$gray-base	#000	#1a1a1a
\$gray-darker	#222	#131313
\$gray-dark	#333	#2a2a2a
\$gray	#555	#313131
\$gray-light	#777	#393939
\$grey-44	#444	#414141
\$grey-88	#888	#484848
\$grey-99	#999	#505050
\$grey-8c	#8c8c8c	#585858
\$grey-ad	#adadad	#676767
\$grey-dark-font	#fff	#f0f0f0
\$grey-white	#fff	#6e6e6e
\$grey-lighter	#eee	#767676
\$grey-f9	#f9f9f9	#7e7e7e
\$grey-f8	#f8f8f8	#858585

Name	Value (Default Theme)	Value (Dark Theme)
\$grey-f5	#f5f5f5	#8d8d8d
\$grey-e6	#e6e6e6	#959595
\$grey-dd	#ddd	#9c9c9c
\$grey-d4	#d4d4d4	#a4a4a4
\$grey-cc	#ccc	#acacac
\$grey-light-font	#333	#fff
\$brand-success	#5cb85c	#48b14c
\$brand-success-dark	#3c763d	#358238
\$brand-info	#5bc0de	#2aaac0
\$brand-info-dark	#31708f	#208090
\$brand-warning	#f0ad4e	#fac168
\$brand-warning-dark	#8a6d3b	#f9ad37
\$brand-danger	#d9534f	#d44f4f
\$brand-danger-dark	#a94442	#c12f2f
\$brand-success-light	#dff0d8	#dff0d8
\$brand-info-light	#d9edf7	#d9edf7
\$brand-warning-light	#fcf8e3	#fcf8e3
\$brand-danger-light	#f2dede	#f2dede
\$input-border-focus	#66afe9	#104888
\$brand-success-font	#3c763d	#2f7432
\$brand-info-font	#31708f	#1a6c7a
\$brand-warning-font	#8a6d3b	#9d6106

Name	Value (Default Theme)	Value (Dark Theme)
\$brand-danger-font	#a94442	#ac2a2a
\$base-font	#000	#000
\$brand-primary-lighten-10		lighten(\$brand-primary, 10%)
\$brand-primary-lighten-15		lighten(\$brand-primary, 15%)
\$brand-primary-lighten-20		lighten(\$brand-primary, 20%)
\$brand-primary-lighten-30		lighten(\$brand-primary, 30%)
\$brand-primary-lighten-40		lighten(\$brand-primary, 40%)

Syncfusion Blazor 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

Name	Value (Default Theme)	Value (Dark Theme)
\$grey-black	#000	#000
\$grey-50	#fafafa	#fafafa
\$grey-100	#f5f5f5	#f5f5f5
\$grey-200	#eee	#eee
\$grey-300	#e0e0e0	#e0e0e0
\$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

Name	Value (Default Theme)	Value (Dark Theme)
\$warning-font		#ff9800
\$info-font		#03a9f4

Syncfusion Blazor 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
\$neutral-tertiary-alt	#c8c8c8	#6f6c6c
\$neutral-tertiary	#a6a6a6	#9a9a9a
\$neutral-secondary-alt	#767676	#c8c8c8

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

Syncfusion Blazor 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
\$content-font	#fff
\$content-font-alt	#969696

Name	Value
\$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

Syncfusion Blazor Tailwind CSS Theme

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

Name	Value (Default Theme)	Value (Dark Theme)
\$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
\$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

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

Blazor Icons Library

The Syncfusion Blazor library provides the set of **base64** formatted font icons which are being used in the Syncfusion Blazor components. These icons can be utilized in the web applications using **SfIcon** component or **e-icons** class.

Icon component

Syncfusion Icon component provides support to render predefined Syncfusion icons or custom font icons.

You can refer [Getting Started with Syncfusion Blazor for Server-side in Visual Studio page](#) for the introduction and configuring the common specifications.

The following code example shows the rendering of built-in Syncfusion icons from predefined **IconName** options using **Name** property by defining them in **SfIcon** tag.

ASPX-CS

```
@using Syncfusion.Blazor.Buttons
<SfIcon Name="IconName.Cut"></SfIcon>
<SfIcon Name="IconName.Copy"></SfIcon>
<SfIcon Name="IconName.Paste"></SfIcon>
<SfIcon Name="IconName.Bold"></SfIcon>
<SfIcon Name="IconName.Underline"></SfIcon>
<SfIcon Name="IconName.Italic"></SfIcon>
```



Icon size

The font size of the icon can be changed using the **Size** property. The icon displays **Medium** size by default.

- When IconSize set to Small, the font size will be **8px**.
- When IconSize set to Medium, the font size will be **16px**.
- When IconSize set to Large, the font size will be **24px**.

ASPX-CS

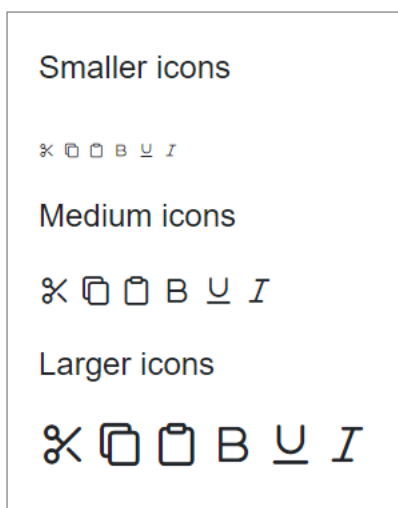
```
@using Syncfusion.Blazor.Buttons
<p>Smaller icons</p>
<SfIcon Name="IconName.Cut" Size="IconSize.Small"></SfIcon>
<SfIcon Name="IconName.Copy" Size="IconSize.Small"></SfIcon>
<SfIcon Name="IconName.Paste" Size="IconSize.Small"></SfIcon>
<SfIcon Name="IconName.Bold" Size="IconSize.Small"></SfIcon>
<SfIcon Name="IconName.Underline" Size="IconSize.Small"></SfIcon>
<SfIcon Name="IconName.Italic" Size="IconSize.Small"></SfIcon>
<p>Medium icons</p>
<SfIcon Name="IconName.Cut" Size="IconSize.Medium"></SfIcon>
<SfIcon Name="IconName.Copy" Size="IconSize.Medium"></SfIcon>
<SfIcon Name="IconName.Paste" Size="IconSize.Medium"></SfIcon>
```



```

<SfIcon Name="IconName.Bold" Size="IconSize.Medium"></SfIcon>
<SfIcon Name="IconName.Underline" Size="IconSize.Medium"></SfIcon>
<SfIcon Name="IconName.Italic" Size="IconSize.Medium"></SfIcon>
<p>Larger icons</p>
<SfIcon Name="IconName.Cut" Size="IconSize.Large"></SfIcon>
<SfIcon Name="IconName.Copy" Size="IconSize.Large"></SfIcon>
<SfIcon Name="IconName.Paste" Size="IconSize.Large"></SfIcon>
<SfIcon Name="IconName.Bold" Size="IconSize.Large"></SfIcon>
<SfIcon Name="IconName.Underline" Size="IconSize.Large"></SfIcon>
<SfIcon Name="IconName.Italic" Size="IconSize.Large"></SfIcon>

```



The `Size` property is applicable only when defining the icon using `Name` property. Otherwise, use [IconCss](#) property to customize the icon.

Tooltip for icons

`Title` property used to set title attribute for the icon to improve accessibility with screen readers and shows a tooltip on mouseover. The following example code displays tooltip text for appropriate icons.

ASPX-CS

```

@using Syncfusion.Blazor.Buttons
<SfIcon Name="IconName.Upload" Title="Upload"></SfIcon>
<SfIcon Name="IconName.Download" Title="Download"></SfIcon>
<SfIcon Name="IconName.Undo" Title="Undo"></SfIcon>
<SfIcon Name="IconName.Redo" Title="Redo"></SfIcon>
<SfIcon Name="IconName.AlignTop" Title="AlignTop"></SfIcon>
<SfIcon Name="IconName.AlignBottom" Title="AlignBottom"></SfIcon>
<SfIcon Name="IconName.AlignMiddle" Title="AlignMiddle"></SfIcon>

```



Icon appearance customization

The [SfIcon](#) supports to customize color and size by overriding the `e-icons` class.

The following example code demonstrates the custom font-size and color for icons.

ASPX-CS

```
@using Syncfusion.Blazor.Buttons
<SfIcon Name="IconName.AlignLeft"></SfIcon>
<SfIcon Name="IconName.AlignRight"></SfIcon>
<SfIcon Name="IconName.AlignCenter"></SfIcon>
<SfIcon Name="IconName.Justify"></SfIcon>
<SfIcon Name="IconName.DecreaseIndent"></SfIcon>
<SfIcon Name="IconName.IncreaseIndent"></SfIcon>
<style>
.e-icons{
color: #ff0000;
font-size: 26px !important;
}
</style>
```



Third party icons integration

The [SfIcon](#) supports to render custom font icons using the [IconCss](#) property. To render custom font icons define the required font CSS that provides the required font name, font size, and content for the icon.

The following code explains how to render `open-iconic` icons using `IconCss` property.

ASPX-CS

```
@using Syncfusion.Blazor.Buttons
<SfIcon IconCss="oi oi-list-rich"></SfIcon>
<SfIcon IconCss="oi oi-account-login"></SfIcon>
<SfIcon IconCss="oi oi-account-logout"></SfIcon>
<SfIcon IconCss="oi oi-action-redo"></SfIcon>
<SfIcon IconCss="oi oi-action-undo"></SfIcon>
<SfIcon IconCss="oi oi-clock"></SfIcon>
<SfIcon IconCss="oi oi-audio"></SfIcon>
<SfIcon IconCss="oi oi-bluetooth"></SfIcon>
```



HTML attribute support

You can add the additional HTML attributes to the icon element using [HtmlAttributes](#) property. HTML attributes can be added by specifying as inline attributes or by specifying `@attributes` razor directive.

The following example shows the icon font size customization using `@attributes` directive.

ASPX-CS

```
@using Syncfusion.Blazor.Buttons
<SfIcon Name="IconName.Copy" @attributes="customAttribute"></SfIcon>
@code{
Dictionary<string, object> customAttribute = new Dictionary<string,
object>()
{
{ "style", "font-size: 20px" }
};
}
```

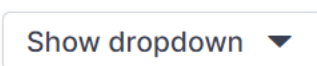
Icon integration with Button component

The built-in Syncfusion icons can be integrated with other Blazor components without defining the `<SfIcon>` tag. To use Syncfusion icons, add `e-icons` class that contains the font-family and common property of the font icons. Add the icon class with the corresponding icon name from the [available icons](#) with `e-` prefix.

The following example shows how to integrate the icons with Syncfusion button component by defining the icon class in the `IconCss` property of button.

ASPX-CS

```
@using Syncfusion.Blazor.Buttons
<SfButton IconCss="e-icons e-chevron-down-fill" Content="Show dropdown"
IconPosition="IconPosition.Right"></SfButton>
```



Using icons directly in HTML element

The built-in Syncfusion icons can be rendered directly in the HTML element by defining `e-icons` class that contains the font-family and common property of font icons, and defining the [available icon's](#) class with `e-` prefix.

The following code example explains the direct rendering of Syncfusion `search` icon in the span element.

ASPX-CS

Icons list

The complete pack of Syncfusion Blazor icons is listed in the following table. The corresponding icon content can be referred to 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>
```

Common

Version Compatibility of Syncfusion Blazor Components

The following table represents the supported .NET and .NET Core versions by different Syncfusion Blazor components releases.

Version	Syncfusion Blazor Components Version
-----	-----
.NET 6.0	19.3.0.43 and above

- | [.NET 5.0](#) | 18.4.0.30 and above |
- | [.NET Core 3.1.3](#) | 18.1.0.52 and above |
- | [.NET Core 3.1.2](#) | 18.1.0.42 and above |
- | [.NET Core 3.1.1](#) | 17.4.0.46 and above |
- | [.NET Core 3.1](#) | 17.4.0.39 and above |

Version Information

In a year, Syncfusion releases new volumes once every three months. Syncfusion Blazor components follows sequence-based identifiers to process software releases based on the **Major.Minor.Build.Revision** format to track all the release changes. It helps developers to keep track of the changes in every release.

For example, if the release a package version is **19.3.0.43**, the version number indicates the details as follows,

- **19** denotes **Major release** version which changes every year.
- **3** denotes **Volume release**. Syncfusion releases a new volume once every three months. Here **3** represents the third release of the year.
- **0** denotes **Build Number** and it will always zero for blazor.
- **43** denotes **Patch Number** also known as revision number which increases for each service pack release and weekly patch release.

See also

- [Syncfusion Product release lifecycle](#)
- [.NET and .NET Core release lifecycle](#)

Reference scripts in Blazor Application

This section provides information about the script isolation process and how to reference scripts from CDN, Static Web Assets and Custom resource generator (CRG) for Syncfusion Blazor Components.

The javascript interop files needs to be added to support the features that can't be implemented in native blazor.

JavaScript isolation

Syncfusion Blazor components supports JavaScript isolation where the needed scripts are loaded by the component itself when its rendered. So, you don't have to reference scripts externally in application.

Syncfusion recommends to reference scripts using [Static Web Assets](#), [CDN](#) and [CRG](#) by disabling JavaScript isolation for better loading performance of the blazor application.

Disable JavaScript isolation

The Syncfusion Blazor components supports to refer scripts externally at the application-end by disabling default JavaScript isolation approach for better initial loading performance which explained in the previous section. You can disable JS isolation by setting [IgnoreScriptIsolation](#) as **true** while adding Syncfusion blazor service using `AddSyncfusionBlazor()`.

Blazor Server App

- If you're using .NET 6 Blazor Server App, set [IgnoreScriptIsolation](#) property as `true` using `AddSyncfusionBlazor` service method in `~/Program.cs` file.

C# `TABTITLE="PROGRAM" HL_LINES="10 11"`

C#

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddRazorPages();
    services.AddServerSideBlazor();
    // Set IgnoreScriptIsolation as true to load custom scripts
    services.AddSyncfusionBlazor(options => {
        options.IgnoreScriptIsolation = true;
    });
    services.AddSingleton<WeatherForecastService>();
}
```

If you are using Blazor Server App with net6.0 SDK, the **Startup.cs** file is not available in the Application. Set `IgnoreScriptIsolation` as `true` in `AddSyncfusionBlazor` service in `~/Program.cs` file for Blazor Server app.

C#

```
// For .NET 6 project, add the Syncfusion Blazor Service in Program.cs file.
using Microsoft.AspNetCore.Components;
using Microsoft.AspNetCore.Components.Web;
using Syncfusion.Blazor;
var builder = WebApplication.CreateBuilder(args);
builder.Services.AddRazorPages();
builder.Services.AddServerSideBlazor();
....
builder.Services.AddSyncfusionBlazor(options => {
    options.IgnoreScriptIsolation = true; });
var app = builder.Build();
....
....
```

Blazor WebAssembly App (~/Program.cs)

C#

```
// For .NET 6 project.
using Microsoft.AspNetCore.Components;
using Microsoft.AspNetCore.Components.Web;
using Syncfusion.Blazor;
var builder = WebApplication.CreateBuilder(args);
builder.Services.AddRazorPages();
builder.Services.AddServerSideBlazor(options => {
    options.IgnoreScriptIsolation = true; });
....
```

```
builder.Services.AddSyncfusionBlazor();  
var app = builder.Build();  
....  
....
```

C#

```
// For .NET 5 or .NET Core SDK 3.1 project.  
using Syncfusion.Blazor;  
....  
....  
public static async Task Main(string[] args)  
{  
    var builder = WebAssemblyHostBuilder.CreateDefault(args) ;  
    ....  
    ....  
    // Set IgnoreScriptIsolation as true to load custom scripts  
    builder.Services.AddSyncfusionBlazor(options => {  
        options.IgnoreScriptIsolation = true;  
    });  
    await builder.Build().RunAsync();  
}
```

3. Now, manually add the custom interop script and styles in the Blazor App.

- a) For .NET 6 Blazor Server app, add the custom interop script and styles in the `~/Pages/_Layout.cshtml` page.
- b) For .NET 5 / .NET Core SDK 3.1 Blazor Server app, add the custom interop script and styles in the `~/Pages/_Host.cshtml` page.
- c) For Blazor WebAssembly app, add the custom interop script and styles in the `~/wwwroot/index.html` page.

HTML

```
<head>  
....  
....  
<link href="material.css" rel="stylesheet" />  
<script src="syncfusion-blazor.min.js" type="text/javascript"></script>  
</head>
```

How to use CDN resources in the Blazor application

The same theme and script files can be referred through the CDN version by setting the `IgnoreScriptIsolation` to true in `AddSyncfusionBlazor` service in `~/Startup.cs` file for Blazor Server app or `~/Program.cs` file for Blazor WebAssembly app which is in step 2.

- a) For .NET 6 Blazor Server app, add the CDN interop script and styles in the `~/Pages/_Layout.cshtml` page.

b) For .NET 5 / .NET Core SDK 3.1 Blazor Server app, add the CDN interop script and styles in the `~/Pages/_Host.cshtml` page.

c) For Blazor WebAssembly app, add the CDN interop script and styles in the `~/wwwroot/index.html` page.

HTML

```
<head>
...
...
<link href="https://cdn.syncfusion.com/blazor/{{ site.blazorversion
}}/styles/bootstrap4.css" rel="stylesheet" />
<script src="https://cdn.syncfusion.com/blazor/{{ site.blazorversion
}}/syncfusion-blazor.min.js" type="text/javascript"></script>
</head>
```

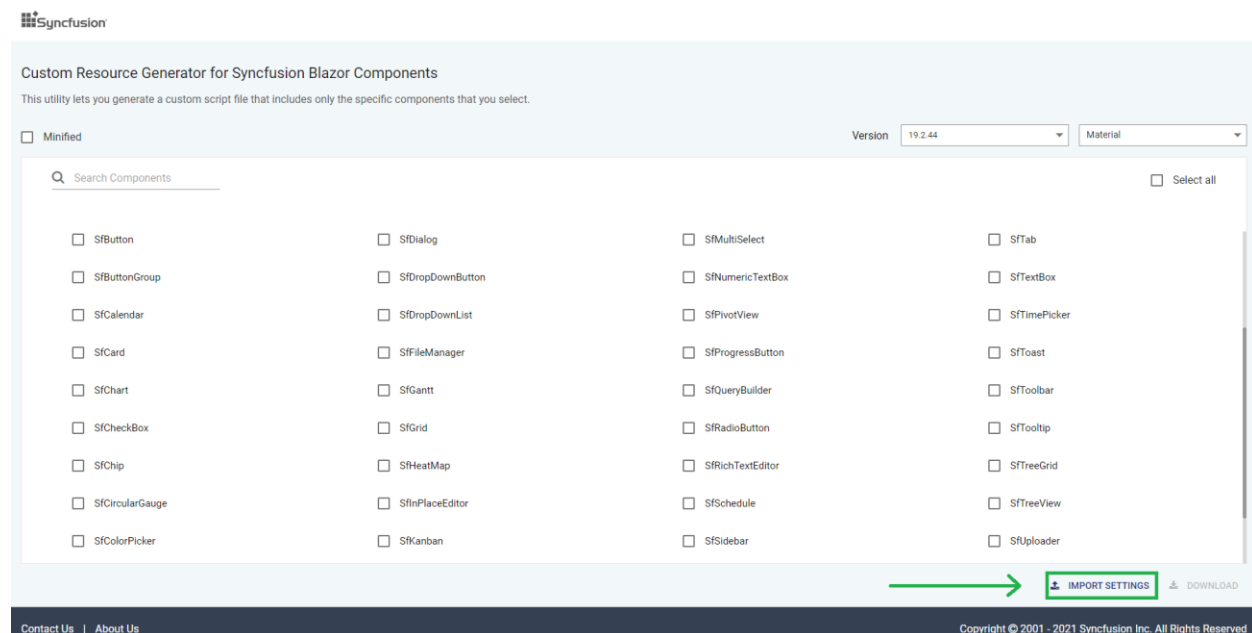
4. Run the application and it will load the resources with application required components.

Import previously generated settings into CRG

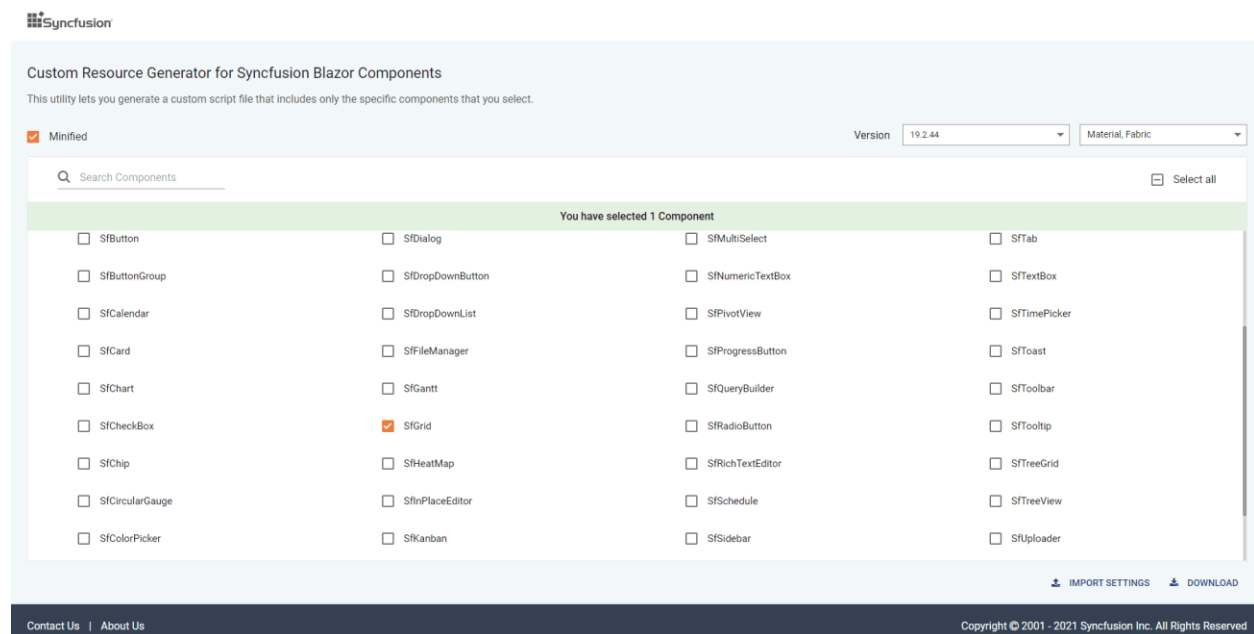
To add more components or upgrade the latest Syncfusion Blazor library resources, it is not necessary to generate from the scratch in the CRG. Just import the old **import.json** file, make the changes, and then download it again from the CRG application.

Refer to the following steps to import previous settings in CRG:

1. Click the **IMPORT SETTINGS** button at the bottom of the page.



2. Upload the **import.json** file, so that the previously stored data will be restored in the CRG application. Now, add more components and export the resources again.



Accessibility support with ADA compliance in Syncfusion Blazor components

All the Syncfusion Blazor components follow the WAI-ARIA accessibility standard by default. This enables you to build accessible web applications, which are fully navigable by users with disabilities.

Section 508

Section 508 is part of the U.S. Rehabilitation Act of 1973 introduced in 1998, requires Federal agencies to make their electronic and information technology (EIT) accessible to people with disabilities. This law defines the need for both members of the public and Federal employees in all Federal agencies to have access to, develop, maintain, procure, or use electronic and information technology. Under Section 508, Federal agencies must develop a website that can be used by people with disabilities.

All our Syncfusion Blazor components comply with the standard of Section 508 accessibility by default. This enables you to build accessible web applications, which are fully navigable by users with disabilities.

Keyboard navigation

Keyboard navigation support enables users to interact with components using keyboard shortcuts. Each component has its own set of shortcuts. Refer to the following documents to get the details of each component.

- [Accordion](#)
- [AutoComplete](#)
- [Calendar](#)
- [CheckBox](#)
- [Chip](#)
- [Circular Gauge](#)
- [ColorPicker](#)
- [ComboBox](#)
- [ContextMenu](#)
- [Dashboard Layout](#)
- [DatePicker](#)
- [DateRangePicker](#)

- [DateTimePicker](#)
- [Dialog](#)
- [DropDownButton](#)
- [DropDownList](#)
- [FileManager](#)
- [Grid](#)
- [Linear Gauge](#)
- [List Box](#)

Localization (Multi-Language) support in Syncfusion Blazor components

Localization (L10N) is the process of adapting application components and content to the desired language with its corresponding region. This page shows you how to use the Localization feature in your application.

How to enable Localization in Blazor application

To get started with Localization in Blazor application, you have to familiarize yourself with the Localization and its architecture in the framework. Refer to the below resources to know more about Localization in the Blazor framework.

- [Blazor Localization](#)
- [ASP.Net Core Localization](#)

The culture can be set using one of the following approaches:

- [Cookies](#)
- [Provide UI to choose the culture](#)

How Syncfusion Blazor UI Component supports Localization

- The Syncfusion Blazor UI components can translate its UI element content based on the user-defined language or culture. The Localization support is processed by using Resource **.resx** files. These resource files contain the key-value pair of locale content in the following format.

C#

```
using Syncfusion.Blazor;  
using System.Globalization;  
using ApplicationNamespace.Shared;  
using Microsoft.Extensions.Options;  
using Microsoft.AspNetCore.Localization;  
public class Startup  
{  
    .....  
    .....  
    public void ConfigureServices(IServiceCollection services)  
    {  
        services.AddControllers();  
        #region Localization  
        // Set the resx file folder path to access  
        services.AddLocalization(options => options.ResourcesPath = "Resources")  
        ;  
    }  
}
```

```

services.AddSyncfusionBlazor();
// Register the Syncfusion locale service to customize the SyncfusionBlazor
component locale culture
services.AddSingleton(typeof(ISyncfusionStringLocalizer), typeof
(SyncfusionLocalizer));
services.Configure<RequestLocalizationOptions>(options =>
{
// Define the list of cultures your app will support
var supportedCultures = new List<CultureInfo>()
{
new CultureInfo("en-US"),
new CultureInfo("de")
};
// Set the default culture
options.DefaultRequestCulture = new RequestCulture("en-US");
options.SupportedCultures = supportedCultures;
options.SupportedUICultures = supportedCultures;
});
#endregion
.....
.....
}
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
#region Localization
app.UseRequestLocalization(app.ApplicationServices.
GetService<IOptions<RequestLocalizationOptions>>().Value);
#endregion
.....
.....
app.UseEndpoints(endpoints =>
{
// Adds endpoints for controller actions to the IEndpointRouteBuilder
without specifying any routes.
endpoints.MapControllers();
endpoints.MapBlazorHub();
endpoints.MapFallbackToPage("/_Host");
});
}
}

```

Here, the `ApplicationNamespace` is your application name.

2. Create `~/Shared/SyncfusionLocalizer.cs` file and implement `ISyncfusionStringLocalizer` to the class. This acts as a middleware to connect the Syncfusion Blazor UI components and resource files.

Map the `SfResources.ResourceManager` to this interface `Manager`.

C#

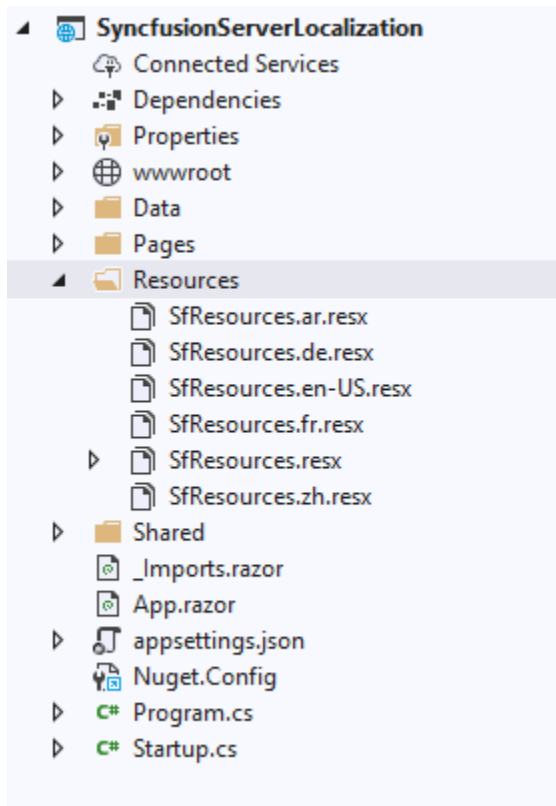
```

using Syncfusion.Blazor;
public class SyncfusionLocalizer : ISyncfusionStringLocalizer
{
// To get the locale key from mapped resources file

```

```
public string GetText(string key)
{
    return this.ResourceManager.GetString(key);
}
// To access the resource file and get the exact value for locale key
public System.Resources.ResourceManager ResourceManager {
    get
    {
        // Replace the ApplicationNamespace with your application name.
        return ApplicationNamespace.Resources.SfResources.ResourceManager;
    }
}
}
```

3. Add the resource files in the `~/Resources` folder. The locale resource files for different cultures are available in this [GitHub](#) repository. You can get any culture resource file from there and utilize it in your application.



After adding the resource file in the application we need to generate the designer class for the resources. To generate the designer class, open the default `resx` file in Visual Studio, and set its `Access Modifier` to `Public`. This will create an entry in your `.csproj` file similar to the following.

XML

```
<ItemGroup>
<EmbeddedResource Update="Resources\SfResources.en-US.resx">
<Generator>PublicResXFileCodeGenerator</Generator>
```

```
</EmbeddedResource>
<EmbeddedResource Update="Resources\SfResources.resx">
<Generator>PublicResXFileCodeGenerator</Generator>
<LastGenOutput>SfResources.Designer.cs</LastGenOutput>
</EmbeddedResource>
</ItemGroup>
```

4. Create `~/Pages/_Host.cshtml.cs` file and use cookies to store the user-selected culture.

C#

```
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Localization;
using Microsoft.AspNetCore.Mvc.RazorPages;
using System.Globalization;
public class HostModel : PageModel
{
    public void OnGet()
    {
        HttpContext.Response.Cookies.Append(
            CookieRequestCultureProvider.DefaultCookieName,
            CookieRequestCultureProvider.MakeCookieValue(
                new RequestCulture(
                    CultureInfo.CurrentCulture,
                    CultureInfo.CurrentUICulture)));
    }
}
```

5. Create `~/Controllers/CultureController.cs` file and configure the controller to switch the culture using UI.

C#

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Localization;
[Route("[controller]/[action]")]
public class CultureController : Controller
{
    public IActionResult SetCulture(string culture, string redirectUri)
    {
        if (culture != null)
        {
            HttpContext.Response.Cookies.Append(
                CookieRequestCultureProvider.DefaultCookieName,
                CookieRequestCultureProvider.MakeCookieValue(
                    new RequestCulture(culture)));
        }
        return LocalRedirect(redirectUri);
    }
}
```

6. Create a Blazor component `CultureSwitcher.razor` in the `~/Shared/` folder.

ASPX-CS

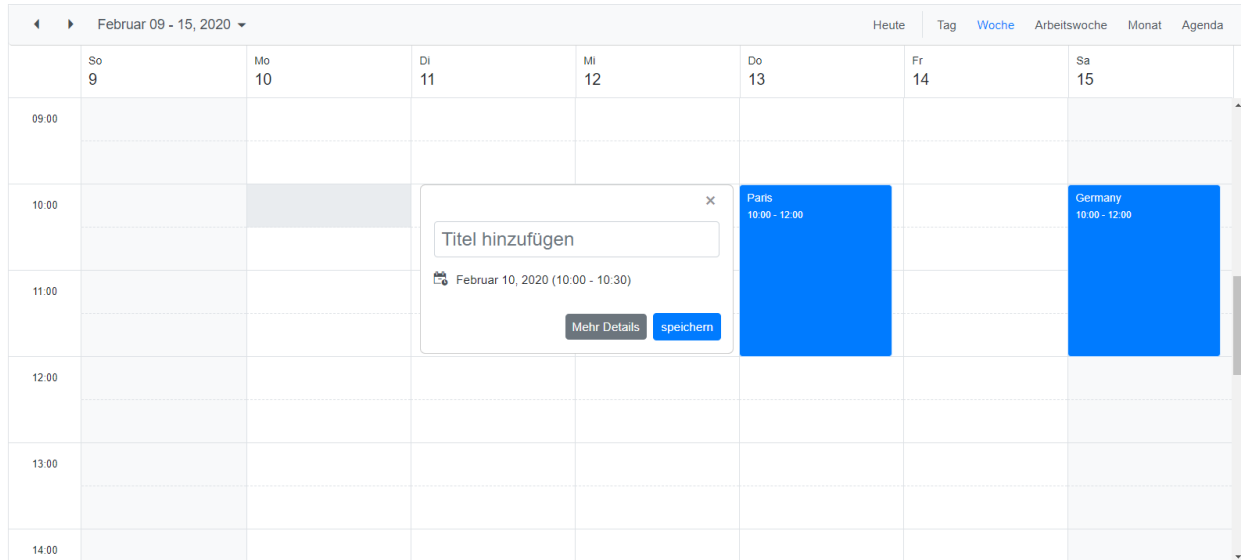
```
@inject NavigationManager NavigationManager
<h3>Select your language</h3>
<select @onchange="OnSelected">
  <option>Select Culture</option>
  <option value="en-US">English</option>
  <option value="de">German</option>
</select>
@code {
  private void OnSelected(ChangeEventArgs e)
  {
    var culture = (string)e.Value;
    var uri = new Uri(NavigationManager.Uri)
    .GetComponents(UriComponents.PathAndQuery, UriFormat.Unescaped);
    var query = $"?culture={Uri.EscapeDataString(culture)}&" +
    $"redirectUri={Uri.EscapeDataString(uri)}";
    NavigationManager.NavigateTo("/Culture/SetCulture" + query, forceLoad:
    true);
  }
}
```

7. Add `CultureSwitcher` component to `~/Shared/MainLayout.razor` file to enable the culture switcher in all pages.

ASPX-CS

```
<div class="main">
  <div class="top-row px-4">
    <CultureSwitcher />
    <a href="https://docs.microsoft.com/aspnet/" target="_blank">About</a>
  </div>
  <div class="content px-4">
    @Body
  </div>
</div>
```

8. Add Syncfusion Blazor UI components in `~/Pages/*.razor` and run the application. The following screenshot illustrates the output of the German `de-DE` culture.



Download Blazor Server app App from [Github](#).

Enable Localization in Blazor WebAssembly application

1. Add the Localization service configuration in the `~/Program.cs` file.

C#

```
using Syncfusion.Blazor;
using Microsoft.JSInterop;
using System.Globalization;
using ApplicationNamespace.Shared;
public class Program
{
    public static async Task Main(string[] args)
    {
        var builder = WebAssemblyHostBuilder.CreateDefault(args);
        .....
        builder.Services.AddSyncfusionBlazor();
        #region Localization
        // Register the Syncfusion locale service to customize the SyncfusionBlazor
        // component locale culture
        builder.Services.AddSingleton(typeof(ISyncfusionStringLocalizer),
            typeof(SyncfusionLocalizer));
        // Set the default culture of the application
        CultureInfo.DefaultThreadCurrentCulture = new CultureInfo("en-US");
        CultureInfo.DefaultThreadCurrentUICulture = new CultureInfo("en-US");
        // Get the modified culture from culture switcher
        var host = builder.Build();
        var jsInterop = host.Services.GetRequiredService<IJSRuntime>();
        var result = await jsInterop.InvokeAsync<string>("cultureInfo.get");
        if (result != null)
        {
            // Set the culture from culture switcher
            var culture = new CultureInfo(result);
            CultureInfo.DefaultThreadCurrentCulture = culture;
        }
    }
}
```

```
CultureInfo.DefaultThreadCurrentUICulture = culture;
}
#endregion
await builder.Build().RunAsync();
}
```

Here, the `ApplicationNamespace` is your application name.

2. Create `~/Shared/SyncfusionLocalizer.cs` file and implement `ISyncfusionStringLocalizer` to the class. This acts as a middleware to connect the Syncfusion Blazor UI components and resource files.

Map the `SfResources.ResourceManager` to this interface `Manager`.

C#

```
using Syncfusion.Blazor;
public class SyncfusionLocalizer : ISyncfusionStringLocalizer
{
    // To get the locale key from mapped resources file
    public string GetText(string key)
    {
        return this.ResourceManager.GetString(key);
    }
    // To access the resource file and get the exact value for locale key
    public System.Resources.ResourceManager ResourceManager {
        get
        {
            // Replace the ApplicationNamespace with your application name.
            return ApplicationNamespace.Resources.SfResources.ResourceManager;
        }
    }
}
```

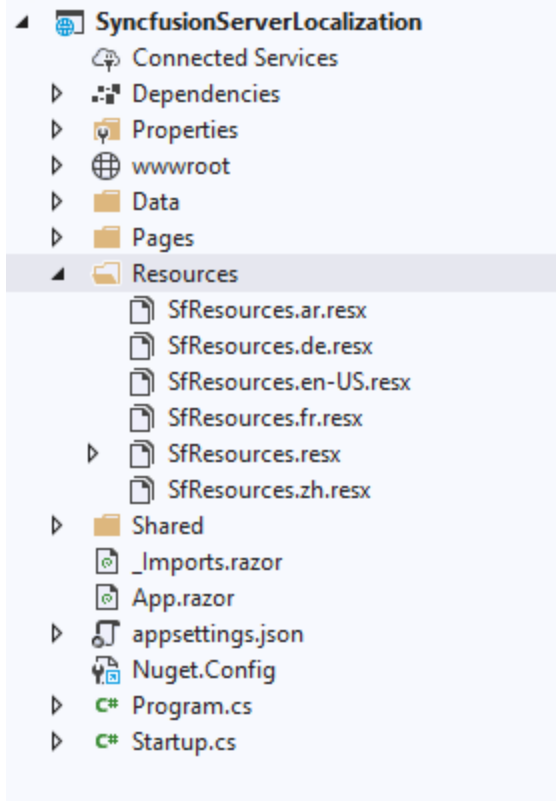
For .NET 5.0 Blazor Webassembly globalization, we should configure the `BlazorWebAssemblyLoadAllGlobalizationData` in the project file when the application uses large resources and dynamic culture changes.

XML

```
<PropertyGroup>
<BlazorWebAssemblyLoadAllGlobalizationData>true</BlazorWebAssemblyLoadAllGlobalizationData>
</PropertyGroup>
```

Refer [here](#) for more details.

3. Add the resource files in the `~/Resources` folder. The locale resource files for different cultures are available in this [GitHub](#) repository. You can get any culture resource file from there and utilize it in your application.



After adding the resource file in the application we need to generate the designer class for the resources. To generate the designer class, open the default `resx` file in Visual Studio, and set its **Access Modifier** to **Public**. This will create an entry in your `.csproj` file similar to the following.

XML

```
<ItemGroup>
  <Compile Update="Resources\SfResources.Designer.cs">
    <DesignTime>True</DesignTime>
    <AutoGen>True</AutoGen>
    <DependentUpon>SfResources.resx</DependentUpon>
  </Compile>
</ItemGroup>
<ItemGroup>
  <EmbeddedResource Update="Resources\SfResources.resx">
    <Generator>PublicResXFileCodeGenerator</Generator>
    <LastGenOutput>SfResources.Designer.cs</LastGenOutput>
  </EmbeddedResource>
</ItemGroup>
```

4. Add the custom JavaScript interop function to get or set the culture in `~/wwwrooot/index.html`.

HTML

```
<body>
  . . . . .
```

```
.....
<script src="_framework/blazor.webassembly.js"></script>
<script>
window.cultureInfo = {
get: () => window.localStorage['BlazorCulture'],
set: (value) => window.localStorage['BlazorCulture'] = value
};
</script>
</body>
```

5. Create a Blazor component **CultureSwitcher.razor** in the **~/Shared/** folder.

ASPX-CS

```
@using System.Globalization
@inject IJSRuntime JSRuntime
@inject NavigationManager NavigationManager
<select @bind="Culture">
@foreach (var culture in supportedCultures)
{
<option value="@culture">@culture.DisplayName</option>
}
</select>
@code {
private CultureInfo[] supportedCultures = new[]
{
new CultureInfo("en-US"),
new CultureInfo("de")
};
private CultureInfo Culture
{
get => CultureInfo.CurrentCulture;
set
{
if (CultureInfo.CurrentCulture != value)
{
var js = (IJSInProcessRuntime)JSRuntime;
js.InvokeVoid("cultureInfo.set", value.Name);
NavigationManager.NavigateTo(NavigationManager.Uri, forceLoad: true);
}
}
}
}
```

6. Add **CultureSwitcher** component to **~/Shared/MainLayout.razor** file to enable the culture switcher in all pages.

ASPX-CS

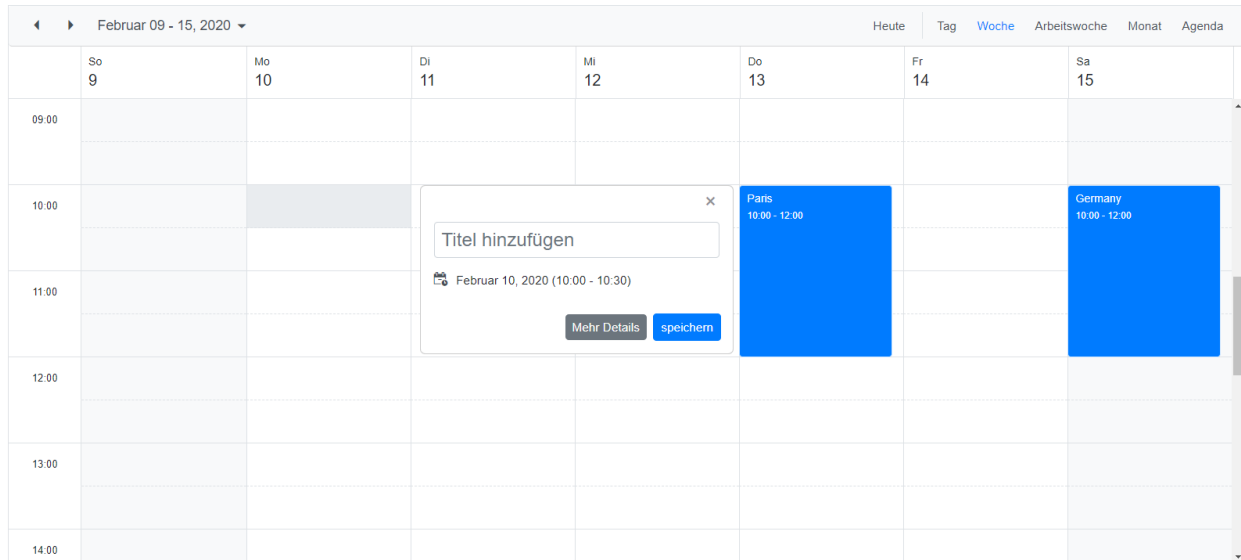
```
<div class="main">
<div class="top-row px-4">
<CultureSwitcher />
<a href="https://docs.microsoft.com/aspnet/" target="_blank">About</a>
```

```

</div>
<div class="content px-4">
@Body
</div>
</div>

```

7. Add Syncfusion Blazor UI components in `~/Pages/*.razor` and run the application. The following screenshot illustrates the output of the German `de-DE` culture.



Download Blazor WebAssembly App from [Github](#).

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 Blazor UI components are specific to the American English (en-US) culture by default. It utilizes the `Blazor Internationalization` package to parse and format the number and date objects based on the chosen culture.
- Suppose, if you want to change any specific culture, then add the corresponding culture resource (.resx) file by using the below reference.
- [Changing culture and Adding Resx file in the application](#)

Right-To-Left support in Syncfusion Blazor Components

The right-to-left (RTL) support can be enabled for Syncfusion Blazor components by setting `EnableRtl` property to `true`. This will render all the Syncfusion Blazor components in the right-to-left direction.

Enable RTL for all components

You can enable right to left (RTL) for all Syncfusion components used in the application by setting `EnableRtl` global option as `true` while adding Syncfusion blazor service using `AddSyncfusionBlazor()`.

Blazor Server App

- If you're using .NET 6 Blazor Server App, set [EnableRtl](#) property as `true` using `AddSyncfusionBlazor` service method in `~/Program.cs` file.

C#

```
using Microsoft.AspNetCore.Components;
using Microsoft.AspNetCore.Components.Web;
using Syncfusion.Blazor;
var builder = WebApplication.CreateBuilder(args);
builder.Services.AddRazorPages();
builder.Services.AddServerSideBlazor();
....
builder.Services.AddSyncfusionBlazor(options => { options.EnableRtl = true;
});
var app = builder.Build();
....
```

- If you're using .NET 5 or .NET Core 3.1 Blazor Server App, set [EnableRtl](#) property as `true` using `AddSyncfusionBlazor` service method in `~/Startup.cs` file.

C#

```
using Syncfusion.Blazor;
namespace WebApplication1
{
    public class Startup
    {
        public void ConfigureServices(IServiceCollection services)
        {
            ....
            services.AddSyncfusionBlazor(options => { options.EnableRtl = true; });
        }
    }
}
```

Blazor WebAssembly App

If you're using Blazor WebAssembly App, set [EnableRtl](#) property as `true` using `AddSyncfusionBlazor` service method in `~/Program.cs` file.

- For .NET 6 Blazor WebAssembly App

C#

```
using Microsoft.AspNetCore.Components;
using Microsoft.AspNetCore.Components.Web;
using Syncfusion.Blazor;
var builder = WebApplication.CreateBuilder(args);
builder.Services.AddRazorPages();
builder.Services.AddServerSideBlazor();
....
```

```
builder.Services.AddSyncfusionBlazor(options => { options.EnableRtl = true;
});
var app = builder.Build();
....
```

- For .NET 5 or .NET Core 3.1 Blazor WebAssembly App

C#

```
using Syncfusion.Blazor;
....
public static async Task Main(string[] args)
{
var builder = WebAssemblyHostBuilder.CreateDefault(args) ;
....
builder.Services.AddSyncfusionBlazor(options => { options.EnableRtl = true;
});
await builder.Build().RunAsync();
}
```

The above configuration enables the Right-To-Left (RTL) support globally for all the Syncfusion Blazor components. For illustration, the Syncfusion Blazor DataGrid component is displayed with Right-To-Left (RTL).

Drag a column header here to group its column			
Freight	Order Date	Customer Name	Order ID
€ 2,10	18.03.2020	ALFKI	1001
€ 4,20	17.03.2020	BOLID	1002
€ 6,30	16.03.2020	BLONP	1003
€ 8,40	15.03.2020	ANTON	1004
€ 10,50	14.03.2020	BOLID	1005
€ 12,60	13.03.2020	ALFKI	1006
€ 14,70	12.03.2020	ALFKI	1007
€ 16,80	11.03.2020	ANANTR	1008

of 4 pages (25 items) 1

« < 4 3 2 1 > »

Enable RTL to individual component

You can enable right to left (RTL) for particular component by setting component's `EnableRtl` property to `true`.

In the below code example, right to left enabled for `SfDropDownList` component by directly setting `EnableRtl` property.

ASPX-CS

```
@using Syncfusion.Blazor.DropDowns
<SfDropDownList TValue="string" Placeholder="Select the country"
TItem="Countries" DataSource="@CountryList" EnableRtl="true">
```

```

<DropDownListFieldSettings Text="Name"
Value="Code"></DropDownListFieldSettings>
</SfDropDownList>
@code {
SfDropDownList<string, Countries> dropdownObj;
public class Countries
{
public string Name { get; set; }
public string Code { get; set; }
}
List<Countries> CountryList = new List<Countries>
{
new Countries() { Name = "Australia", Code = "AU" },
new Countries() { Name = "Bermuda", Code = "BM" },
new Countries() { Name = "Canada", Code = "CA" },
new Countries() { Name = "Cameroon", Code = "CM" }
};
}

```

Select the country

State Persistence

The Syncfusion Blazor library supports persisting a component's state across page refreshes or navigation. To enable this feature, set the `EnablePersistence` property to `true` to the required component. This will store the component's state in the browser's `localStorage` object on-page `unload` event. For example, persistence has been enabled to the grid component in the following code.

The state of a component will be retained during navigation or refreshment based on the ID. Make sure to set an ID for the component to store the component's state in the browser.

C#

```

@using Syncfusion.Blazor.Grids
<SfGrid ID="grid" EnablePersistence="true" AllowPaging="true"
DataSource="@Orders">
<GridPageSettings PageSize="8"></GridPageSettings>
<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
Width="100"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" Width="100"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) Format="C2"
Width="120"></GridColumn>
</GridColumns>
</SfGrid>
@code {
public List<Order> Orders { get; set; }
protected override async Task OnInitializedAsync()
{
await base.OnInitializedAsync();
Orders = Enumerable.Range(1, 25).Select(x => new Order()
{

```

```

OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
OrderDate = DateTime.Now.AddDays(-x),
}).ToList();
}
public class Order
{
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
}

```

State Persistence Supported Components and Properties

The following table demonstrates the list of Syncfusion Blazor components that are supported with state persistence and describes the list of properties stored in the `localStorage`.

<!-- markdownlint-disable MD033 -->

Component Name	Properties
SfGrid	<ul style="list-style-type: none"> Columns GridFilterSettings GridSortSettings GridGroupSettings GridPageSettings

<!-- markdownlint-enable MD033 -->

Input Form Validation and Data Annotation

The Syncfusion Blazor UI input and editor components can be validated by the standards defined in the Blazor [Form Validation](#). The user's input value can be validated based on the [DataAnnotation attributes](#) defined in the model class.

How to Validate Syncfusion Blazor UI Components

1. Create a model class and set [DataAnnotation attributes](#) to its properties.

ASPX-CS

```

@using System.ComponentModel.DataAnnotations;
@code {
private EmployeeDetails employeeModel = new EmployeeDetails();
public class EmployeeDetails
{
[Required(ErrorMessage = "Please enter {0}.")]
public string FirstName { get; set; }
[Required(ErrorMessage = "Please enter {0}.")]
public string LastName { get; set; }
}
}

```

```

    ....
    ....
}
}

```

2. Add the **EditForm** component in the view page and assign the model object to its **Model** property. Also, declare the **DataAnnotationsValidator** and **ValidationSummary** components inside the **EditForm** component.

ASPX-CS

```

<EditForm Model="@employeeDetails">
  <DataAnnotationsValidator />
  <ValidationSummary/>
</EditForm>

```

[DataAnnotationsValidator](#) - Attaches validation support to an **EditContext** using data annotations.

[ValidationSummary](#) - Used to display a summarized list of all validation messages.

3. Add any desired [Syncfusion components](#) inside the **EditForm** and provide the values through **@bind-** property.

ASPX-CS

```

<EditForm Model="@employeeModel">
  <DataAnnotationsValidator />
  <ValidationSummary />
  <div class="form-group">
    <label for="first-name">First Name:</label>
    <SfTextBox ID="first-name" @bind-
Value="employeeModel.FirstName"></SfTextBox>
  </div>
  <div class="form-group">
    <label for="last-name">Last Name:</label>
    <SfTextBox ID="last-name" @bind-Value="employeeModel.LastName"></SfTextBox>
  </div>
  <SfButton>Submit</SfButton>
</EditForm>

```

Refer to [this table](#) for the Syncfusion Blazor components which supports the form validation and its **@bind** property details.

4. The **EditForm** validates the input values based on its edit context and displays the validation message on the **ValidationSummary** component when submitting the form.

Validation Failure:

- Please enter FirstName.
- Please enter LastName.

First Name:

Last Name:

Submit

Validation Success:

First Name:

Last Name:

Submit

Green color border around the textbox indicates validation success state and Red color indicates validation failure state.

5. You can also use the `ValidationMessage` component to display the validation error message for each input component instead of displaying a validation summary using the `ValidationSummary` component.

ASPX-CS

```
<EditForm Model="@employeeModel">
<DataAnnotationsValidator />
<div class="form-group">
<label for="first-name">First Name:</label>
<SfTextBox ID="first-name" @bind-
Value="employeeModel.FirstName"></SfTextBox>
<ValidationMessage For="@(() =>
employeeModel.FirstName)"></ValidationMessage>
</div>
<div class="form-group">
```

```

<label for="last-name">Last Name:</label>
<SfTextBox ID="last-name" @bind-Value="employeeModel.LastName"></SfTextBox>
<ValidationMessage For="@(() =>
employeeModel.LastName)"></ValidationMessage>
</div>
<SfButton>Submit</SfButton>
</EditForm>

```

- The `EditForm`'s submit events [OnValidSubmit](#) and [OnInvalidSubmit](#) can be used to get the validation success and failure `EditContext` details.

ASPX-CS

```

<EditForm Model="@employeeDetails" OnValidSubmit="FormValidSubmit"
OnInvalidSubmit="FormInvalidSubmit">
....
....
</EditForm>
@code {
private void FormValidSubmit(EditContext context)
{
// Triggers when form input has valid values.
}
private void FormInvalidSubmit(EditContext context)
{
// Triggers when form input has invalid values.
}
}

```

You can also replace the above two events with [OnSubmit](#) handler and validate the form manually.

ASPX-CS

```

<EditForm Model="@employeeDetails" OnSubmit="FormSubmit">
....
....

```

```

<EditForm>
@code {
private void FormSubmit(EditContext context)
{
// Validates the EditContext and returns bool to indicate whether it has
valid or invalid input values.
bool isValid = context.Validate();
if (isValid)
{
// Form has valid inputs.
}
else
{
// Form has invalid inputs.
}
}
}
}

```

Warning: You shouldn't use the `OnSubmit` handler along with `OnValidSubmit` and `OnInvalidSubmit` event handlers in the `EditForm` component. It will throw a runtime error.

Blazor Form Validation Supported Components

The following section provides the details about the Syncfusion Blazor UI components that are supported with form validation, corresponding `@bind` properties, and validation examples.

<!-- markdownlint-disable MD033 -->

NuGet Package Name	Component Name	Property Name
Syncfusion.Blazor.Buttons	<ul style="list-style-type: none"> SfCheckBox SfRadioButton SfSwitch 	<ul style="list-style-type: none"> Checked
Syncfusion.Blazor.Calendars	<ul style="list-style-type: none"> SfCalendar SfDatePicker SfDateTimePicker SfTimePicker 	<ul style="list-style-type: none"> Value
	<ul style="list-style-type: none"> SfDateRangePicker 	<ul style="list-style-type: none"> StartDate EndDate
Syncfusion.Blazor.DropDowns	<ul style="list-style-type: none"> SfAutoComplete SfComboBox SfDropDownList SfListBox SfMultiSelect 	<ul style="list-style-type: none"> Value
Syncfusion.Blazor.InPlaceEditor	<ul style="list-style-type: none"> SfInPlaceEditor 	<ul style="list-style-type: none"> Value

<u>Syncfusion.Blazor.Inputs</u>	<ul style="list-style-type: none"> • <u>SfColorPicker</u> • <u>SfMaskedTextBox</u> • <u>SfNumericTextBox</u> • <u>SfSlider</u> • <u>SfTextBox</u> 	<ul style="list-style-type: none"> • Value
<u>Syncfusion.Blazor.RichTextEditor</u>	<ul style="list-style-type: none"> • <u>SfRichTextEditor</u> 	<ul style="list-style-type: none"> • Value

<!-- markdownlint-enable MD033 -->

Make sure to use nullable integer `int?` when you use the `Required` attribute on the `Value` property. The default value for `int` is 0. So, it will pass the validation when you submit the form.

SfAutoComplete

The Autocomplete component provides a list of suggestions when the user types the value on the input field. It uses the `@bind-Value` parameter to validate the form model `EditContext`.

ASPX-CS

```
@using System.ComponentModel.DataAnnotations;
<EditForm Model="@dropdownModel">
<DataAnnotationsValidator />
<div class="form-group">
<label for="country">Country:</label>
<SfAutoComplete ID="country" @bind-Value="@dropdownModel.Country"
DataSource="@Countries">
<AutoCompleteFieldSettings Value="Name"></AutoCompleteFieldSettings>
</SfAutoComplete>
<ValidationMessage For="@(() => dropdownModel.Country)"></ValidationMessage>
</div>
<SfButton>Submit</SfButton>
</EditForm>

@code {
private DropdownModel dropdownModel = new DropdownModel();
public class DropdownModel
{
[Required(ErrorMessage = "Please select your country.")]
public string Country { get; set; }
}
public class Country
{
public string Name { get; set; }
public string Code { get; set; }
}
public List<Country> Countries = new List<Country>
{
new Country() { Name = "Australia", Code = "AU" },
new Country() { Name = "Bermuda", Code = "BM" },
new Country() { Name = "Canada", Code = "CA" }
};
}
```

SfCalendar

The Calendar is a component to display the date and days of the week. It uses the `@bind-Value` parameter to validate the form model `EditContext`.

If you are using the `Required` attribute, make sure to use the `DateTime?` type. You can also specify the `Range` attribute to validate between two specific date values.

ASPX-CS

```
@using System.ComponentModel.DataAnnotations;
<EditForm Model="@dateModel">
  <DataAnnotationsValidator />
  <div class="form-group">
    <label for="calendar">Calendar:</label>
    <SfCalendar ID="calendar" @bind-Value="@dateModel.Date"></SfCalendar>
    <ValidationMessage For="@(() => dateModel.Date)"></ValidationMessage>
  </div>
  <SfButton>Submit</SfButton>
</EditForm>
@code {
private DateModel dateModel = new DateModel();
public class DateModel
{
  [Required]
  [Range(typeof(DateTime), "1/1/1950", "12/31/2000",
    ErrorMessage = "The {0} value must be between {1:dd/MM/yyyy} and {2:dd/MM/yyyy}.")]
  public DateTime? Date { get; set; }
}
```

SfCheckBox

The Checkbox component is used to select one or more options from a list of predefined choices. It uses the `@bind-Checked` parameter to validate the form model `EditContext`.

ASPX-CS

```
@using System.ComponentModel.DataAnnotations;
<EditForm Model="@myModel">
  <DataAnnotationsValidator />
  <div class="form-group">
    <SfCheckBox Label="I accept the Terms and Conditions" @bind-Checked="@myModel.IsAccepted"></SfCheckBox>
    <ValidationMessage For="@(() => myModel.IsAccepted)"></ValidationMessage>
  </div>
  <SfButton>Submit</SfButton>
</EditForm>
@code {
private MyModel myModel = new MyModel();
public class MyModel
{
  [Required]
  [Range(typeof(bool), "true", "true",
    ErrorMessage = "You need to accept the Terms and Conditions.")]
  public bool IsAccepted { get; set; }
}
```

```
}
```

SfColorPicker

The ColorPicker component allows you to choose a specific color value instead of input. It uses the `@bind-Value` parameter to validate the form model `EditContext`.

ASPX-CS

```
@using System.ComponentModel.DataAnnotations;
<EditForm Model="@personal">
<DataAnnotationsValidator />
<div class="form-group">
<label for="fav-color">Favorite Color:</label>
<SfColorPicker ID="fav-color" @bind-
Value="@personal.FavoriteColor"></SfColorPicker>
<ValidationMessage For="@(() =>
personal.FavoriteColor)"></ValidationMessage>
</div>
<SfButton>Submit</SfButton>
</EditForm>
@code {
private Personal personal = new Personal();
public class Personal
{
[Required(ErrorMessage = "Please specify your favorite color.")]
public string FavoriteColor { get; set; }
}
}
```

SfComboBox

The Combobox component is an editable dropdown list that also allows users to select an option from a predefined popup list. It uses the `@bind-Value` parameter to validate the form model `EditContext`.

ASPX-CS

```
@using System.ComponentModel.DataAnnotations;
<EditForm Model="@dropdownModel">
<DataAnnotationsValidator />
<div class="form-group">
<label for="country">Country:</label>
<SfComboBox ID="country" @bind-Value="@dropdownModel.Country"
DataSource="@Countries">
<ComboBoxFieldSettings Value="Name"></ComboBoxFieldSettings>
</SfComboBox>
<ValidationMessage For="@(() => dropdownModel.Country)"></ValidationMessage>
</div>
<SfButton>Submit</SfButton>
</EditForm>
@code {
private DropdownModel dropdownModel = new DropdownModel();
public class DropdownModel
{
[Required(ErrorMessage = "Please select your country.")]
public string Country { get; set; }
}
}
```

```
public class Country
{
    public string Name { get; set; }
    public string Code { get; set; }
}
public List<Country> Countries = new List<Country>
{
    new Country() { Name = "Australia", Code = "AU" },
    new Country() { Name = "Bermuda", Code = "BM" },
    new Country() { Name = "Canada", Code = "CA" },
    new Country() { Name = "Cameroon", Code = "CM" }
};
}
```

SfDatePicker

The DatePicker component allows you to enter or select a date value on the input field. It uses the `@bind-Value` parameter to validate the form model `EditContext`.

If you are using the `Required` attribute, make sure to use the `DateTime?` type. You can also specify the `Range` attribute to validate between two specific date values.

ASPX-CS

```
@using System.ComponentModel.DataAnnotations;
<EditForm Model="@dateModel">
    <DataAnnotationsValidator />
    <div class="form-group">
        <label for="dob">Date of Birth:</label>
        <SfDatePicker ID="dob" @bind-Value="@dateModel.Date"></SfDatePicker>
        <ValidationMessage For="@(() => dateModel.Date)"></ValidationMessage>
    </div>
    <SfButton>Submit</SfButton>
</EditForm>
@code {
    private DateModel dateModel = new DateModel();
    public class DateModel
    {
        [Required]
        [Range(typeof(DateTime), "1/1/1950", "12/31/2000",
            ErrorMessage = "The {0} value must be between {1:dd/MM/yyyy} and {2:dd/MM/yyyy}.")]
        public DateTime? Date { get; set; }
    }
}
```

SfDateRangePicker

The DateRangePicker component allows you to enter or select a range of start and end date values on the input field. It uses `@bind-StartDate` and `@bind-EndDate` parameters to validate the form model `EditContext`.

If you are using the `Required` attribute, make sure to use the `DateTime?` type. You can also specify the `Range` attribute to validate between two specific date values.

ASPX-CS

```

@using System.ComponentModel.DataAnnotations;
<EditForm Model="@dateRange">
<DataAnnotationsValidator />
<div class="form-group">
<label for="date-range">Date Range:</label>
<ValidationMessage For="@(() => dateRange.StartDate)"></ValidationMessage>
<SfDateRangePicker ID="date-range" @bind-StartDate="@dateRange.StartDate"
@bind-EndDate="@dateRange.EndDate"></SfDateRangePicker>
<ValidationMessage For="@(() => dateRange.EndDate)"></ValidationMessage>
</div>
<SfButton>Submit</SfButton>
</EditForm>
@code {
private DateModel dateRange = new DateModel();
public class DateModel
{
[Required]
[Range(typeof(DateTime), "1/1/1990", "12/31/2000",
ErrorMessage = "The {0} value should be between {1:dd/MM/yyyy} and
{2:dd/MM/yyyy}.")]
public DateTime? StartDate { get; set; }
[Required]
[Range(typeof(DateTime), "1/1/1990", "12/31/2000",
ErrorMessage = "The {0} value should be between {1:dd/MM/yyyy} and
{2:dd/MM/yyyy}.")]
public DateTime? EndDate { get; set; }
}
}

```

SfDateTimePicker

The `DateTimePicker` component allows you to enter or select date and time values on the input field. It uses the `@bind-Value` parameter to validate the form model `EditContext`.

If you are using the `Required` attribute, make sure to use the `DateTime?` type. You can also specify the `Range` attribute to validate between two specific date values.

ASPX-CS

```

@using System.ComponentModel.DataAnnotations;
<EditForm Model="@dateModel">
<DataAnnotationsValidator />
<div class="form-group">
<label for="date-time">Date and Time:</label>
<SfDateTimePicker ID="date-time" @bind-
Value="@dateModel.DateTime"></SfDateTimePicker>
<ValidationMessage For="@(() => dateModel.DateTime)"></ValidationMessage>
</div>
<SfButton>Submit</SfButton>
</EditForm>
@code {
private DateModel dateModel = new DateModel();
public class DateModel
{
[Required]
[Range(typeof(DateTime), "1/1/1990 09:00:00 AM", "12/31/2000 06:00:00 PM",

```



```
ErrorMessage = "The {0} value should be between {1:dd/MM/yyyy hh:mm tt} and {2:dd/MM/yyyy hh:mm tt}."}]
public DateTime? DateTime { get; set; }
}
}
```

SfDropDownList

The DropDownList component allows users to select an option from a predefined popup list. It uses the `@bind-Value` parameter to validate the form model `EditContext`.

ASPX-CS

```
@using System.ComponentModel.DataAnnotations;
<EditForm Model="@dropdownModel">
<DataAnnotationsValidator />
<div class="form-group">
<label for="country">Country:</label>
<SfDropDownList ID="country" @bind-Value="@dropdownModel.Country"
DataSource="@Countries">
<DropDownListFieldSettings Value="Name"></DropDownListFieldSettings>
</SfDropDownList>
<ValidationMessage For="@(( ) => dropdownModel.Country)"></ValidationMessage>
</div>
<SfButton>Submit</SfButton>
</EditForm>
@code {
private DropdownModel dropdownModel = new DropdownModel();
public class DropdownModel
{
[Required(ErrorMessage = "Please select your country.")]
public string Country { get; set; }
}
public class Country
{
public string Name { get; set; }
public string Code { get; set; }
}
public List<Country> Countries = new List<Country>
{
new Country() { Name = "Australia", Code = "AU" },
new Country() { Name = "Bermuda", Code = "BM" },
new Country() { Name = "Canada", Code = "CA" },
new Country() { Name = "Cameroon", Code = "CM" }
};
}
```

SfInPlaceEditor

The In-PlaceEditor component allows users to dynamically edit within its context. It uses the `@bind-Value` parameter to validate the form model `EditContext`.

ASPX-CS

```
@using System.ComponentModel.DataAnnotations;
<EditForm Model="@editorModel">
<DataAnnotationsValidator />
```

```

<div class="form-group">
<label for="name">Employee Name:</label>
<SfInPlaceEditor ID="name"
Type="Syncfusion.Blazor.InPlaceEditor.InputType.Text" @bind-
Value="@editorModel.Name">
<EditorComponent>
<SfTextBox @bind-Value="@editorModel.Name"></SfTextBox>
</EditorComponent>
</SfInPlaceEditor>
<ValidationMessage For="@(() => editorModel.Name)"></ValidationMessage>
</div>
<SfButton>Submit</SfButton>
</EditForm>
@code {
private EditorModel editorModel = new EditorModel();
public class EditorModel
{
[Required(ErrorMessage = "Employee name is required.")]
public string Name { get; set; }
}
}

```

SfListBox

The **ListBox** component allows users to select multiple options from a list of predefined values. It uses the **@bind-Value** parameter to validate the form model **EditContext**.

ASPX-CS

```

@using System.ComponentModel.DataAnnotations;
<EditForm Model="@dropdownModel">
<DataAnnotationsValidator />
<div class="form-group">
<label for="games">Sports:</label>
<SfListBox ID="games" DataSource="@Games" @bind-
Value="@dropdownModel.Games">
<ListBoxFieldSettings Text="Text" Value="ID" />
</SfListBox>
<ValidationMessage For="@(() => dropdownModel.Games)"></ValidationMessage>
</div>
<SfButton>Submit</SfButton>
</EditForm>
@code {
private DropdownModel dropdownModel = new DropdownModel();
public class DropdownModel
{
[Required]
[MinLength(3, ErrorMessage = "Please select atleast 3 games.")]
public string[] Games { get; set; }
}
public class GameFields
{
public string ID { get; set; }
public string Text { get; set; }
}
public List<GameFields> Games = new List<GameFields>()
{

```

```

new GameFields() { ID= "Game1", Text= "American Football" },
new GameFields() { ID= "Game2", Text= "Badminton" },
new GameFields() { ID= "Game3", Text= "Basketball" },
new GameFields() { ID= "Game4", Text= "Cricket" },
new GameFields() { ID= "Game5", Text= "Football" },
new GameFields() { ID= "Game6", Text= "Golf" },
new GameFields() { ID= "Game7", Text= "Hockey" },
new GameFields() { ID= "Game8", Text= "Rugby" },
new GameFields() { ID= "Game9", Text= "Snooker" },
new GameFields() { ID= "Game10", Text= "Tennis" },
};
}

```

SfMaskedTextBox

The Masked TextBox component provides an easy and reliable way to collect user input based on a standard mask. It uses the `@bind-Value` parameter to validate the form model `EditContext`.

ASPX-CS

```

@using System.ComponentModel.DataAnnotations;
<EditForm Model="@cardModel">
  <DataAnnotationsValidator />
  <div class="form-group">
    <label for="credit-card">Credit Card:</label>
    <SfMaskedTextBox ID="credit-card" Mask="0000-0000-0000-0000" @bind-
    Value="@cardModel.CreditCard"></SfMaskedTextBox>
    <ValidationMessage For="@(() => cardModel.CreditCard)"></ValidationMessage>
  </div>
  <div class="form-group">
    <label for="pin-number">Pin Number:</label>
    <SfMaskedTextBox ID="pin-number" Mask="0000" @bind-
    Value="@cardModel.Pin"></SfMaskedTextBox>
    <ValidationMessage For="@(() => cardModel.Pin)"></ValidationMessage>
  </div>
  <div class="form-group">
    <label for="phone">Phone:</label>
    <SfMaskedTextBox ID="phone" Mask="+1-000-000-0000" @bind-
    Value="@cardModel.Phone"></SfMaskedTextBox>
    <ValidationMessage For="@(() => cardModel.Phone)"></ValidationMessage>
  </div>
  <SfButton>Submit</SfButton>
</EditForm>
@code {
private CardModel cardModel = new CardModel();
public class CardModel
{
  [Required]
  [RegularExpression("^4[0-9]{12}(?:[0-9]{3})?$", ErrorMessage = "Please enter
  a valid credit card number.")]
  public string CreditCard { get; set; }
  [Required]
  [RegularExpression("[0-9]{4}$", ErrorMessage = "Invalid pin number.")]
  public string Pin { get; set; }
  [MinLength(10, ErrorMessage = "Please enter a valid phone number.")]
  public string Phone { get; set; }
}

```

```
}

```

SfMultiSelect

The MultiSelect component allows the user to type or select multiple values from a list of predefined options. It uses the `@bind-Value` parameter to validate the form model `EditContext`.

ASPX-CS

```
@using System.ComponentModel.DataAnnotations;
<EditForm Model="@dropdownModel">
<DataAnnotationsValidator />
<div class="form-group">
<label for="sports">Favorite Sports:</label>
<SfMultiSelect ID="sports" @bind-Value="@dropdownModel.Games"
DataSource="@Games">
<MultiSelectFieldSettings Text="Text" Value="ID"></MultiSelectFieldSettings>
</SfMultiSelect>
<ValidationMessage For="@((() => dropdownModel.Games))"></ValidationMessage>
</div>
<SfButton>Submit</SfButton>
</EditForm>
@code {
private DropdownModel dropdownModel = new DropdownModel();
public class DropdownModel
{
[Required]
[MinLength(3, ErrorMessage = "Please select atleast 3 games.")]
public string[] Games { get; set; }
}
public class GameFields
{
public string ID { get; set; }
public string Text { get; set; }
}
public List<GameFields> Games = new List<GameFields>()
{
new GameFields(){ ID= "Game1", Text= "American Football" },
new GameFields(){ ID= "Game2", Text= "Badminton" },
new GameFields(){ ID= "Game3", Text= "Basketball" },
new GameFields(){ ID= "Game4", Text= "Cricket" },
new GameFields(){ ID= "Game5", Text= "Football" },
new GameFields(){ ID= "Game6", Text= "Golf" },
new GameFields(){ ID= "Game7", Text= "Hockey" },
new GameFields(){ ID= "Game8", Text= "Rugby"},
new GameFields(){ ID= "Game9", Text= "Snooker" },
new GameFields(){ ID= "Game10", Text= "Tennis"},
};
}
```

SfNumericTextBox

The Numeric TextBox component is used to get number inputs from the users. It uses the `@bind-Value` parameter to validate the form model `EditContext`.

ASPX-CS

```

@using System.ComponentModel.DataAnnotations;
<EditForm Model="@numericModel">
<DataAnnotationsValidator />
<div class="form-group">
<label for="age">Age:</label>
<SfNumericTextBox ID="age" @bind-Value=@numericModel.Age></SfNumericTextBox>
<ValidationMessage For="@(() => numericModel.Age)"></ValidationMessage>
</div>
<SfButton>Submit</SfButton>
</EditForm>
@code {
private NumericModel numericModel = new NumericModel();
public class NumericModel
{
[Required]
[Range(typeof(int), "22", "50", ErrorMessage = "The {0} value should be
between {1} and {2}.")]
public int Age { get; set; }
}
}

```

SfRadioButton

The Radio Button component allows users to select one option from a list of predefined choices. It uses the `@bind-Checked` parameter to validate the form model `EditContext`.

ASPX-CS

```

@using System.ComponentModel.DataAnnotations;
<EditForm Model="@myModel">
<DataAnnotationsValidator />
<div class="form-group">
<SfRadioButton Label="Small" Name="size" Value="1" @bind-
Checked="@myModel.Value"></SfRadioButton>
<SfRadioButton Label="Medium" Name="size" Value="2" @bind-
Checked="@myModel.Value"></SfRadioButton>
<SfRadioButton Label="Large" Name="size" Value="3" @bind-
Checked="@myModel.Value"></SfRadioButton>
<SfRadioButton Label="Extra Large" Name="size" Value="4" @bind-
Checked="@myModel.Value"></SfRadioButton>
<ValidationMessage For="@(() => myModel.Value)"></ValidationMessage>
</div>
<SfButton>Submit</SfButton>
</EditForm>
@code {
private MyModel myModel = new MyModel();
public class MyModel
{
[Required]
[Range(typeof(int), "1", "3", ErrorMessage = "Extra-large size product is
currently unavailable. Please select any other size.")]
public int Value { get; set; } = 1;
}
}

```

SfRichTextEditor

The RichTextEditor component is used to create blogs, forum posts, notes sections, support tickets (incidents), comment sections, messaging applications, and more. It uses the `@bind-Value` parameter to validate the form model `EditContext`.

ASPX-CS

```
@using System.ComponentModel.DataAnnotations;
<EditForm Model="@editorModel">
  <DataAnnotationsValidator />
  <div class="form-group">
    <label for="comments">Comments:</label>
    <SfRichTextEditor ID="comments" @bind-Value="@editorModel.Feedback" />
    <ValidationMessage For="() => editorModel.Feedback" />
  </div>
  <SfButton>Submit</SfButton>
</EditForm>
@code {
private EditorModel editorModel = new EditorModel();
public class EditorModel
{
  [Required]
  [MaxLength(100, ErrorMessage = "The maximum length of this field is {1} character.")]
  public string Feedback { get; set; }
}
```

SfSlider

The Slider component allows you to select a value or range of values between specified min and max values. It uses the `@bind-Value` parameter to validate the form model `EditContext`.

You can create your own [custom validation](#) logic to validate the Range type slider.

ASPX-CS

```
@using System.ComponentModel.DataAnnotations;
<EditForm Model="@myModel">
  <DataAnnotationsValidator />
  <div class="form-group">
    <label for="volume">Volume:</label>
    <SfSlider ID="volume" @bind-Value="@myModel.Volume"
      Type="SliderType.MinRange">
      <SliderTicks Placement="Placement.After" ShowSmallTicks="true"
        LargeStep="10" SmallStep="5"></SliderTicks>
    </SfSlider>
    <ValidationMessage For="() => myModel.Volume" />
  </div>
  <SfButton>Submit</SfButton>
</EditForm>
@code {
private MyModel myModel = new MyModel();
public class MyModel
{
  [Required]
```

```
[Range(0, 70, ErrorMessage = "Reached higher volume limit. Please reduce to below {2}.")]
public int Volume { get; set; }
}
```

SfSwitch

The Switch component allows you to perform a toggle (on/off) action between checked and unchecked states. It uses the `@bind-Checked` parameter to validate the form model `EditContext`.

ASPX-CS

```
@using System.ComponentModel.DataAnnotations;
<EditForm Model="@myModel">
<DataAnnotationsValidator />
<div class="form-group">
<SfSwitch @bind-Checked="@myModel.IsAccepted">I agree to receive the
newsletter.</SfSwitch>
<ValidationMessage For="@(( ) => myModel.IsAccepted)"></ValidationMessage>
</div>
<SfButton>Submit</SfButton>
</EditForm>
@code {
private MyModel myModel = new MyModel();
public class MyModel
{
[Range(typeof(bool), "true", "true", ErrorMessage = "You need to agree to
receive the newsletter.")]
public bool IsAccepted { get; set; }
}
}
```

SfTextBox

The TextBox is a component for editing, displaying, or entering plain text on forms to capture usernames, phone numbers, email, and more. It uses the `@bind-Value` parameter to validate the form model `EditContext`.

ASPX-CS

```
@using System.ComponentModel.DataAnnotations;
<EditForm Model="@editorModel">
<DataAnnotationsValidator />
<div class="form-group">
<label for="employee-name">Employee Name:</label>
<SfTextBox ID="employee-name" @bind-Value="editorModel.Name"></SfTextBox>
<ValidationMessage For="@(( ) => editorModel.Name)"></ValidationMessage>
</div>
<SfButton>Submit</SfButton>
</EditForm>
@code {
private EditorModel editorModel = new EditorModel();
public class EditorModel
{
[Required(ErrorMessage = "Enter employee name.")]
[MinLength(3, ErrorMessage = "Name should have more than 2 characters.")]
}
```

```
public string Name { get; set; }
}
```

SfTimePicker

The TimePicker component allows users to select a time value either from a pop-up time list or by entering the value directly in the text box. It uses the `@bind-Value` parameter to validate the form model `EditContext`.

If you are using the `Required` attribute, make sure to use the `DateTime?` type. You can also specify the `Range` attribute to validate between two specific date and time values.

ASPX-CS

```
@using System.ComponentModel.DataAnnotations;
<EditForm Model="@dateModel">
<DataAnnotationsValidator />
<div class="form-group">
<label for="work-hours">Work Hours:</label>
<SfTimePicker ID="work-hours" @bind-
Value="@dateModel.WorkHours"></SfTimePicker>
<ValidationMessage For="@(() => dateModel.WorkHours)"></ValidationMessage>
</div>
<SfButton>Submit</SfButton>
</EditForm>
@code {
private DateModel dateModel = new DateModel();
public class DateModel
{
[Required]
[Range(typeof(DateTime), "09:00:00 AM", "06:00:00 PM", ErrorMessage = "The
work hours should be between {1} to {2}.")]
public DateTime? WorkHours { get; set; }
}
}
```

Apply Custom Validation Attributes

You can implement your custom validation logic and apply the [custom attributes](#) on the model class properties. Refer to the following steps to create and apply custom attributes to a model property.

1. Create a class and inherit it from [ValidationAttribute](#).
2. Override the [IsValid](#) method to implement your custom validation logic.
3. Add the custom attribute to the model class property.

ASPX-CS

```
@using System.ComponentModel.DataAnnotations;
<EditForm Model="@rangeModel">
<DataAnnotationsValidator />
<div class="form-group">
<label for="price">Price:</label>
<SfSlider ID="price" @bind-Value="@rangeModel.Price" Type="SliderType.Range"
Min="0" Max="1000">
</SfSlider>
</div>
</EditForm>
```



```

<SliderTicks Placement="Placement.After" ShowSmallTicks="true"
LargeStep="100" SmallStep="50"></SliderTicks>
</SfSlider>
<ValidationMessage For="@(() => rangeModel.Price)"></ValidationMessage>
</div>
<SfButton>Submit</SfButton>
</EditForm>
@code {
private RangeModel rangeModel = new RangeModel();
public class RangeModel
{
[Required]
[CustomRange(300, 700)]
public int[] Price { get; set; } = { 500, 600 };
}
public class CustomRangeAttribute : ValidationAttribute
{
public int StartRange { get; }
public int EndRange { get; }
public CustomRangeAttribute(int startRange, int endRange)
{
StartRange = startRange;
EndRange = endRange;
}
protected override ValidationResult IsValid(object value, ValidationContext
validationContext)
{
var inputValues = (int[])value;
if (inputValues[0] < StartRange || inputValues[1] > EndRange)
{
return new ValidationResult($"The {validationContext.DisplayName} should be
between {StartRange} and {EndRange}.");
}
return ValidationResult.Success;
}
}
}
}

```

Price:

0 100 200 300 400 500 600 700 800 900 1000

The Price should be between 300 and 700.

Submit

Validate Syncfusion Components Inside Another Component

You can create a new Blazor component and use [Syncfusion Blazor components](#) on top of it for your custom implementation. In this case, you may need to validate your Blazor component that should validate the Syncfusion Blazor components.

Refer to the following steps to create and validate the Syncfusion Blazor component on your custom Blazor component.

1. Right-click on the ~/Pages/ folder in the Visual Studio and navigate to Add -> Razor Component. Specify the component name and create it.
2. Add Syncfusion Blazor component in the new component and assign the Value, ValueChanged, and ValueExpression properties. Refer [here](#), to know more about data binding in component parameters.

ASPX-CS

```
@using System.Linq.Expressions;
<label for="@ID">@Label</label>
<SfTextBox ID="@ID" Value="@Text" ValueChanged="@((text) => Text = text)"
ValueExpression="@TextExpression"></SfTextBox>
@code {
    [Parameter]
    public string ID { get; set; }
    [Parameter]
    public string Label { get; set; }
    [Parameter]
    public string Text
    {
        get => text;
        set
        {
            if (text != value)
            {
                text = value;
                TextChanged.InvokeAsync(value);
            }
        }
    }
    private string text;
    [Parameter]
    public EventCallback<string> TextChanged { get; set; }
    [Parameter]
    public Expression<Func<string>> TextExpression { get; set; }
}
```

3. Add your new component in the view page ~/Pages/Index.razor and implement validation using DataAnnotation.

ASPX-CS

```
@using System.ComponentModel.DataAnnotations;
<EditForm Model="@editorModel">
    <DataAnnotationsValidator />
    <div class="form-group">
        <MyTextBox ID="name" Label="Name" @bind-
        Text="@editorModel.Name"></MyTextBox>
        <ValidationMessage For="@(() => editorModel.Name)"></ValidationMessage>
    </div>
    <SfButton>Submit</SfButton>
</EditForm>
@code {
```

```
private EditorModel editorModel = new EditorModel();
public class EditorModel
{
    [Required(ErrorMessage = "Enter employee name.")]
    [MinLength(3, ErrorMessage = "Name should have more than 2 characters.")]
    public string Name { get; set; }
}
```

See Also

- [Model validation in ASP.NET Core and Razor Pages](#)
- [Blazor Forms and Validation](#)

Default HTML Attributes and DOM Events

The Syncfusion Blazor UI components provide the most useful [public API](#) for component implementation and customization. Apart from this public API, the Syncfusion Blazor UI components can support the use of default [HTML attributes](#) and [DOM events](#) in the root element of its component.

Using HTML Attributes and DOM Events in the Input Element

The following is a list of Syncfusion Blazor UI components that use the standard HTML `input` element. You can apply the [HTML input attributes](#) and DOM events directly to the input element used on these Syncfusion Blazor components.

- [SfAutoComplete](#)
- [SfCheckBox](#)
- [SfComboBox](#)
- [SfDatePicker](#)
- [SfDateRangePicker](#)
- [SfDateTimePicker](#)
- [SfDropDownList](#)
- [SfMaskedTextBox](#)
- [SfMultiSelect](#)
- [SfNumericTextBox](#)
- [SfRadioButton](#)
- [SfTextBox](#)
- [SfTimePicker](#)
- [SfUpload](#)

Available Syncfusion Properties Equivalent to HTML Attributes

The following table illustrates the HTML attributes and their equivalent Syncfusion API.

<!-- markdownlint-disable MD033 -->

HTML Attribute	Syncfusion API	Components
<u>id</u>	ID	<ul style="list-style-type: none"> • All Components

<u>autocomplete</u>	Autocomplete	<ul style="list-style-type: none"> SfTextBox
<u>checked</u>	Checked	<ul style="list-style-type: none"> SfCheckBox SfRadioButton
<u>disabled</u>	Disabled	<ul style="list-style-type: none"> SfCheckBox SfRadioButton
	Enabled	<ul style="list-style-type: none"> Others
<u>max</u>	Max	<ul style="list-style-type: none"> SfDatePicker SfDateRangePicker SfDateTimePicker SfNumericTextBox SfTimePicker
<u>min</u>	Min	<ul style="list-style-type: none"> SfDatePicker SfDateRangePicker SfDateTimePicker SfNumericTextBox SfTimePicker
<u>multiple</u>	Multiple	<ul style="list-style-type: none"> SfUpload
<u>placeholder</u>	Placeholder	<ul style="list-style-type: none"> Except below components: SfCheckBox SfRadioButton SfUpload
<u>readonly</u>	ReadOnly	<ul style="list-style-type: none"> Except below components: SfCheckBox SfRadioButton SfUpload
<u>step</u>	Step	<ul style="list-style-type: none"> SfNumericTextBox
<u>value</u>	Value	<ul style="list-style-type: none"> Except below component: SfUpload
<u>width</u>	Width	<ul style="list-style-type: none"> Except below components: SfCheckBox SfRadioButton SfUpload

<!-- markdownlint-enable MD033 -->

If you specify both HTML attribute and Syncfusion API in the component, then the Syncfusion API will get higher priority and will be applied to the DOM element.

ASPX-CS

```
<SfTextBox ID="textbox" name="first-name" title="First name." minlength="15"
Autocomplete="AutoComplete.Off"></SfTextBox>
```

The textbox will be rendered with the following output.

HTML

```
<span class="....">
<input id="textbox" class="...." name="first-name" type="text"
autocomplete="off" title="First name." minlength="15" .... />
</span>
```

In some cases, you may need to add HTML attributes to the root/container element of the above input-based components. For this, you can use [HtmlAttributes](#) Syncfusion API to add HTML attributes to the root/container element.

ASPX-CS

```
<SfTextBox HtmlAttributes="@ (new () { { "style", "background:aliceblue;" }
}) "></SfTextBox>
```

The textbox will be rendered with the following output.

HTML

```
<span class="...." style="background: aliceblue;" ....>
<input .... />
</span>
```

Input DOM Events

The Syncfusion Blazor UI component supports binding the native [DOM events](#) on the input element.

ASPX-CS

```
<div class="form-group">
<SfTextBox @onfocus="OnTextFocus" @onblur="OnTextFocusOut"></SfTextBox>
</div>
@if (eventName != string.Empty)
{
<div class="alert alert-info">@eventName event is triggered on the
TextBox.</div>
}
@code {
private string eventName = string.Empty;
void OnTextFocus ()
{
eventName = "Focus";
}
void OnTextFocusOut ()
```

```
{  
    eventName = "Focusout";  
}
```

Using HTML Attributes and DOM Events in the Root Element

The HTML attributes and DOM events can be applied directly to the component's root element.

ASPX-CS

```
<SfChip @onmouseover="OnChipHover" style="border: 1px solid tomato">  
    <ChipItems>  
        <ChipItem Text="Apple"></ChipItem>  
        <ChipItem Text="Mango"></ChipItem>  
        <ChipItem Text="Banana"></ChipItem>  
    </ChipItems>  
</SfChip>  
  
@code {  
    public void OnChipHover(MouseEventArgs args)  
    {  
        // Do something here on chips hover.  
    }  
}
```

The chip container/root element will be rendered with the following output.

HTML

```
<div style="border: 1px solid tomato" class="...." >  
    <div class="...." role="listbox">  
        ....  
        ....  
    </div>  
</div>
```

Deployment in Blazor

This section provides information about deploying Blazor applications with the Syncfusion Blazor components.

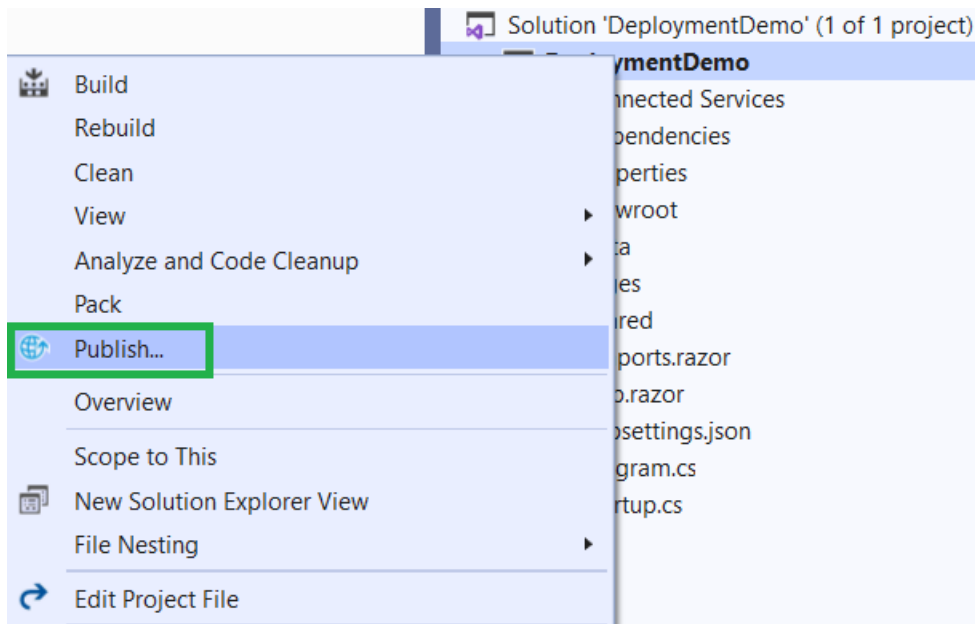
You can get more information about deploying Blazor applications [Here](#).

Publish Blazor Application with Visual Studio 2019

- Create the Blazor application with [Syncfusion Blazor Components](#).

You have to change the base path of the application. Refer to the [MSDN](#) for more details.

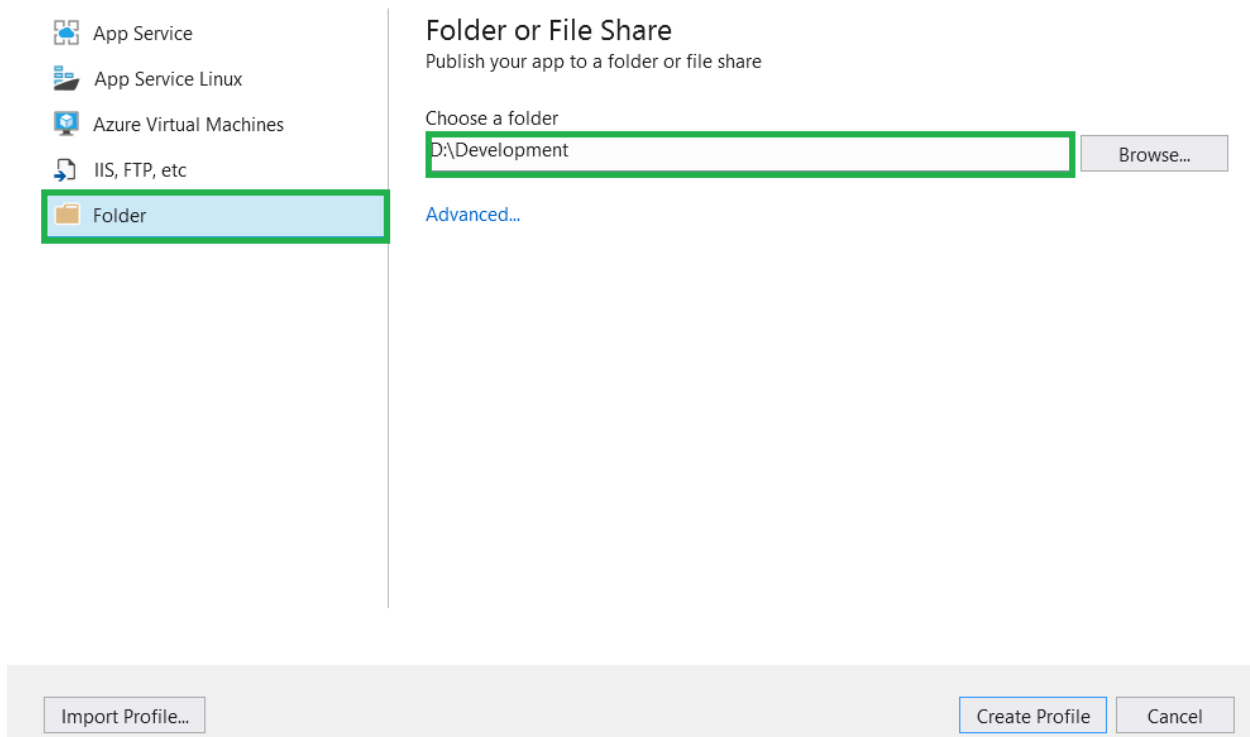
- Right-click on the project in the **Solution Explorer** and select **Publish**.



- Then, select the **Folder** option and select the publishing target location.

x

Pick a publish target



- Check the configuration as Release by clicking the **Advanced...** option below the target location.



Publish

Settings **AdvancedSettings ***

Configuration: **Release**

Target Framework: **netcoreapp3.1**

Deployment Mode: **Framework-Dependent**
[Learn about deployment modes](#)

Target Runtime: **Portable**

File Publish Options

Databases

Discovering Data Contexts...

< Prev Next > Save Cancel

- For **Blazor Server** side application, Set Deployment Mode as **Self-Contained**. Because some dependencies are not loaded properly when we host the published folder.

Deployment Mode: **Self-Contained**

Target Runtime: **Self-Contained**

- Then, click **Save** and **Publish**.

Refer [here](#) for publishing the application to Azure App Service using Visual Studio.

Publish Blazor Application with CLI

Packing the application and its dependencies into a folder for deployment to a hosting system by using the **dotnet publish** command.

For CLI deployment, run the following command from your root directory.

Data Binding

SQL

```
Create Table Orders (  
  OrderID BigInt Identity(1,1) Primary Key Not Null,  
  CustomerID Varchar(100) Not Null,  
  Freight int Null,  
  OrderDate datetime null
```


)

Now, the Orders table design will look like below. Click on the **Update** button.

dbo.Orders [Design] ✕					
Update		Script File: dbo.Orders.sql			
	Name	Data Type	Allow Nulls	Default	
	OrderID	bigint	<input type="checkbox"/>		
	CustomerID	varchar(100)	<input type="checkbox"/>		
	Freight	int	<input checked="" type="checkbox"/>		
	OrderDate	datetime	<input checked="" type="checkbox"/>		
			<input type="checkbox"/>		

Now, click on **Update Database**.

Preview Database Updates ? ✕

Highlights
None

User actions
Create
[dbo].[Orders] (Table)

Supporting actions
None

☒ Include transactional scripts

Generate Script Update Database Cancel

Create OData service project

Open Visual Studio 2019 and create an empty ASP.NET Core Web Application and name it as ODataServiceProject. After creating the application, install **Microsoft.AspNetCore.OData** package by running the following command in the Package Manager Console.

- **Install-Package Microsoft.AspNetCore.OData -Version 7.3.0:** This package contains everything you need to create OData v4.0 endpoints using ASP.NET Core MVC and to support OData query syntax for your web APIs.

Generate DbContext and model class from the database

Now, we are going to scaffold **DbContext** and **model classes** from the existing **OrdersDetails** database. To perform scaffolding and work with the SQL Server database in our application, we need to install the following NuGet packages.

Run the following commands in the **Package Manager Console**.

- **Install-Package Microsoft.EntityFrameworkCore.Tools -Version 3.0.0:** This package creates database context and model classes from the database.
- **Install-Package Microsoft.EntityFrameworkCore.SqlServer -Version 3.0.0:** The database provider that allows Entity Framework Core to work with SQL Server.

Once the above packages are installed, we can scaffold DbContext and Model classes. Run the following command in the **Package Manager Console**.

C#

```
using Microsoft.AspNetCore.OData;
using System.Threading.Tasks;
using ODataServiceProject.Models;
using Microsoft.AspNetCore.Mvc;
// For more information on enabling Web API for empty projects, visit
// https://go.microsoft.com/fwlink/?LinkID=397860
namespace ODataServiceProject.Controllers
{
    [Route("api/[controller]")]
    public class OrdersController : ODataController
    {
        private OrdersDetailsContext _db;
        public OrdersController(OrdersDetailsContext context)
        {
            _db = context;
        }
        [HttpGet]
        [EnableQuery]
        public IActionResult Get()
        {
            return Ok(_db.Orders);
        }
        [EnableQuery]
        public async Task
```

```
book.Patch(entity);
await _db.SaveChangesAsync();
return Updated(entity);
}
[EnableQuery]
public long Delete([FromODataUri] long key)
{
    var deleterow = _db.Orders.Find(key);
    _db.Orders.Remove(deleterow);
    _db.SaveChangesAsync();
    return key;
}
}
```

Add the following line in the **launchSettings.json** file.

JSON

```
{
  "iisSettings": {
    "windowsAuthentication": false,
    "anonymousAuthentication": true,
    "iisExpress": {
      "applicationUrl": "http://localhost:59323",
      "sslPort": 44392
    }
  },
  "profiles": {
    "IIS Express": {
      "commandName": "IISExpress",
      "launchBrowser": true,
      "launchUrl": "odata/orders",
      "environmentVariables": {
        "ASPNETCORE_ENVIRONMENT": "Development"
      }
    },
    "ODataServiceProject": {
      "commandName": "Project",
      "dotnetRunMessages": "true",
      "launchBrowser": true,
      "applicationUrl": "https://localhost:5001;http://localhost:5000",
      "environmentVariables": {
        "ASPNETCORE_ENVIRONMENT": "Development"
      }
    }
  }
}
```

Open **Startup.cs** file and configure by referring to the following codes.

C#

```
namespace ODataServiceProject
{
    public class Startup
```

```

{
    public Startup(IConfiguration configuration)
    {
        Configuration = configuration;
    }
    public IConfiguration Configuration { get; }
    public void ConfigureServices(IServiceCollection services)
    {
        services.AddDbContext<OrdersDetailsContext>(option =>
            option.UseSqlServer(Configuration.GetConnectionString("OrdersDetailsDatabase
            ")));
        services.AddOData();
        services.AddMvc(option => option.EnableEndpointRouting =
            false).SetCompatibilityVersion(CompatibilityVersion.Version_2_1);
    }
    private static IEdmModel GetEdmModel()
    {
        ODataConventionModelBuilder builder = new ODataConventionModelBuilder();
        var books = builder.EntitySet<Orders>("Orders");
        FunctionConfiguration myFirstFunction =
            books.EntityType.Collection.Function("MyFirstFunction");
        myFirstFunction.ReturnsCollectionFromEntitySet<Orders>("Orders");
        return builder.GetEdmModel();
    }
    // This method gets called at the runtime. Use this method to configure the
    // HTTP request pipeline.
    public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
    {
        if (env.IsDevelopment())
        {
            app.UseDeveloperExceptionPage();
        }
        app.UseMvc(routes =>
        {
            routes.MapRoute(
                name: "default",
                template: "{controller=Home}/{action=Index}/{id?}");
            routes.Count().Filter().OrderBy().Expand().Select().MaxTop(null);
            routes.MapODataServiceRoute(
                "odata",
                "odata",
                model: GetEdmModel()
            );
        });
    }
}

```

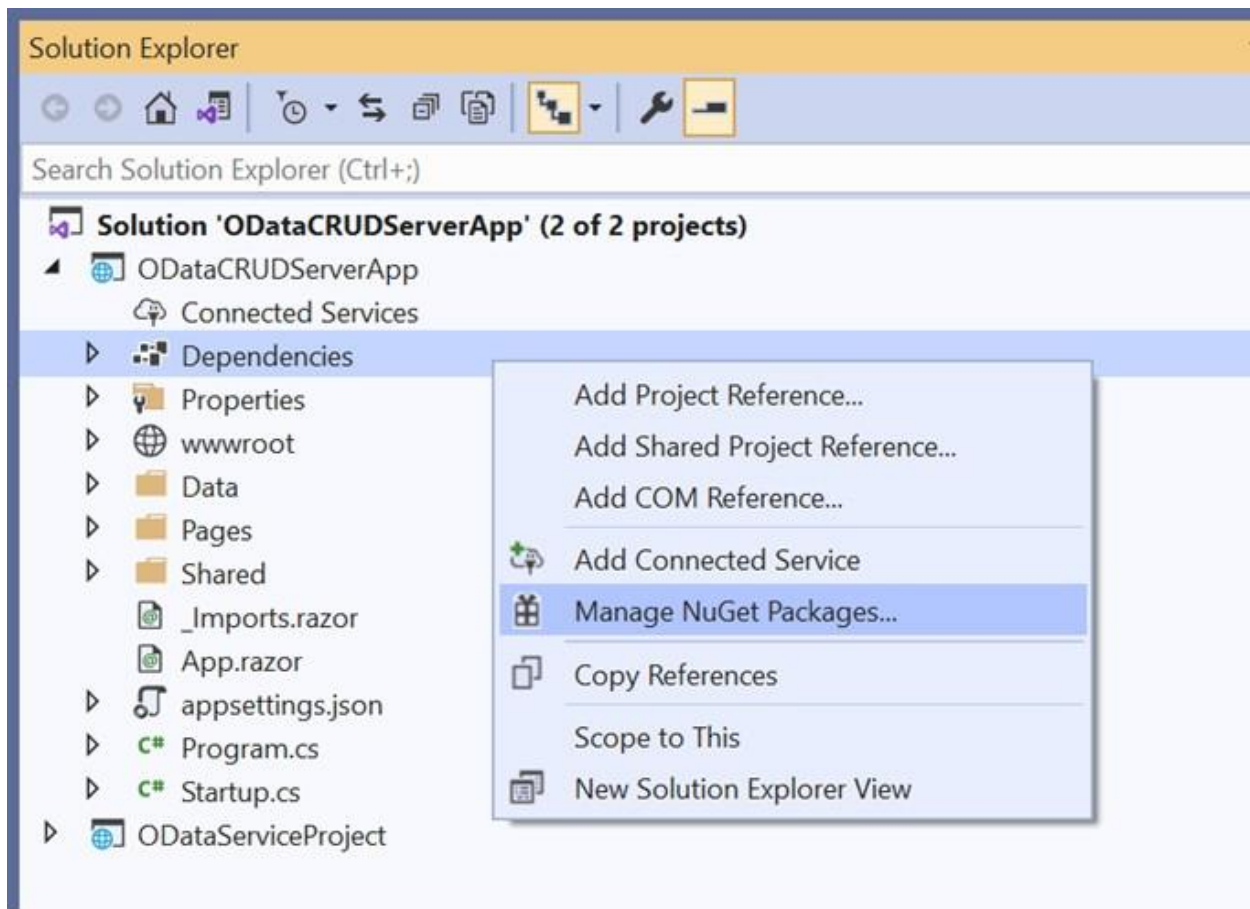
Create Blazor Server Application

Open Visual Studio 2019 and follow the steps in the below documentation to create the Blazor Server Application.

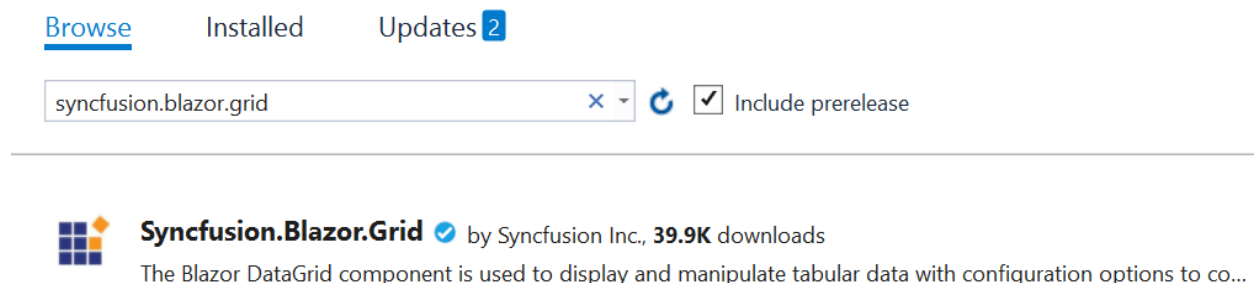
[Getting Started](#)

Add Syncfusion Blazor DataGrid package

To add Syncfusion components into the project, right-click **Dependencies** and select **Manage NuGet Packages**.



Now, in the **Browse** tab, search and install the **Syncfusion.Blazor.Grid** NuGet package.



For this demo, we have used Syncfusion.Blazor(19.1.0.66) NuGet package. We have released a new **Syncfusion.Blazor** NuGet package with new enhancement in our every-week release and main release. So, you can check and update to the latest versions by using this [link](#).

Open **_Import.razor** file and add the following namespaces which are required to use Syncfusion Blazor components in this application.

ASPX-CS

```
@using Syncfusion.Blazor
@using Syncfusion.Blazor.Grids
@using Syncfusion.Blazor.Data
@using ODataServiceProject.Models
```

Open **Startup.cs** file and register the Syncfusion service in the **ConfigureServices** method as follows.

C#

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddRazorPages();
    services.AddServerSideBlazor();
    services.AddSingleton<WeatherForecastService>();
    services.AddSyncfusionBlazor();
}
```

Themes provide life to components. Syncfusion Blazor has different themes. They are:

- Bootstrap4
- Material
- Office 365
- Bootstrap
- High Contrast

In this demo application, the **Bootstrap4** theme will be used. To add the theme, open **Pages/_Host.cshtml** file and add the following CSS reference code.

HTML

```
<link href="_content/Syncfusion.Blazor.Themes/fabric.css" rel="stylesheet" />
```

Add Syncfusion Blazor DataGrid component to an application

In previous steps, we have successfully configured the Syncfusion Blazor package in the application. Now, we can add the grid component to the **Index.razor** page.

ASPX-CS

```
<SfGrid TValue="Orders"></SfGrid>
```

Binding data to Blazor DataGrid component using ODataV4Adaptor

To consume data from the OData Controller, we need to add the **SfDataManager** with **ODataV4Adaptor**. Please refer to the following documentation for more details on ODataV4Adaptor.

[ODataV4Adaptor](#)

ASPX-CS

```
<SfGrid TValue="Orders">
  <SfDataManager Url="https://localhost:44392/odata/orders"
    Adaptor="Adaptors.ODataV4Adaptor"></SfDataManager>
</SfGrid>
```

In the above code example, we have used our localhost address from our application. Instead of localhost, you can give the exact URL of your OData service.

Grid columns can be defined by using the [GridColumn](#) component. We are going to create columns using the following code.

ASPX-CS

```
<SfGrid TValue="Orders">
  <SfDataManager Url="https://localhost:44392/odata/orders"
  Adaptor="Adaptors.ODataV4Adaptor"></SfDataManager>
  <GridColumn>
    <GridColumn Field=@nameof(Orders.OrderId) HeaderText="Order ID"
    IsPrimaryKey="true" Visible="false" TextAlign="TextAlign.Right"
    Width="120"></GridColumn>
    <GridColumn Field=@nameof(Orders.CustomerId) HeaderText="Customer Name"
    Width="150"></GridColumn>
    <GridColumn Field=@nameof(Orders.OrderDate) HeaderText="Order Date"
    Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
    Width="130"></GridColumn>
    <GridColumn Field=@nameof(Orders.Freight) HeaderText="Freight" Format="C2"
    TextAlign="TextAlign.Right" Width="120"></GridColumn>
  </GridColumn>
</SfGrid>
```

When you run the application, the **Get()** method will be called in your OData controller.

C#

```
[Route("api/[controller]")]
public class OrdersController : ODataController
{
  private OrdersDetailsContext _db;
  public OrdersController(OrdersDetailsContext context)
  {
    _db = context;
  }
  [HttpGet]
  [EnableQuery]
  public IActionResult Get()
  {
    return Ok(_db.Orders);
  }
  ...
}
```

Handling CRUD operations with our Syncfusion Blazor DataGrid component

We can enable editing in the grid component using the [GridEditSettings](#) component. Grid provides various modes of editing options such as [Inline/Normal](#), [Dialog](#), and [Batch](#) editing.

Here, we are using **Inline** edit mode and used **Toolbar** property to show toolbar items for editing.

We have added the DataGrid Editing and Toolbar code with previous Grid model.

ASPX-CS

```
<SfGrid TValue="Orders" Toolbar="@ (new List<string>() { "Add", "Edit",
"Delete", "Cancel", "Update" }) ">
<SfDataManager Url="https://localhost:44392/odata/orders"
Adaptor="Adaptors.ODataV4Adaptor"></SfDataManager>
<GridEditSettings AllowAdding="true" AllowDeleting="true"
AllowEditing="true" Mode="EditMode.Normal"></GridEditSettings>
<GridColumn>
<GridColumn Field=@nameof(Orders.OrderId) HeaderText="Order ID"
IsPrimaryKey="true" Visible="false" TextAlign="TextAlign.Right"
Width="120"></GridColumn>
<GridColumn Field=@nameof(Orders.CustomerId) HeaderText="Customer Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Orders.OrderDate) HeaderText="Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Orders.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumn>
</SfGrid>
```

Normal editing is the default edit mode for the DataGrid component. Set the [IsPrimaryKey](#) property of Column as **true** for a particular column, whose value is a unique value for editing purposes.

Insert a row

To insert a new row, click the **Add** toolbar button. The new record edit form will look like below.

Customer Name	Order Date	Freight
VINET	6/8/2021	3

Clicking the **Update** toolbar button will insert the record in the Orders table by calling the below **POST** method of the OData controller.

C#

```
[EnableQuery]
public async Task<IActionResult> Post([FromBody] Orders book)
{
    _db.Orders.Add(book);
    _db.SaveChanges();
    return Created(book);
}
```

Customer Name	Order Date	Freight
VINET	6/8/2021	\$3.00

Update a row

To edit a row, select any row and click the **Edit** toolbar button. The edit form will look like below. Edit the Customer Name column.

+ Add Edit Delete Cancel Update		
Customer Name	Order Date	Freight
VINET edited	6/8/2021	3

Clicking the **Update** toolbar button will update the record in the Orders table by calling the below **PATCH** method of the OData controller.

C#

```
[EnableQuery]
public async Task<IActionResult> Patch([FromODataUri] long key, [FromBody]
Delta<Orders> book)
{
    var entity = await _db.Orders.FindAsync(key);
    book.Patch(entity);
    await _db.SaveChangesAsync();
    return Updated(entity);
}
```

The resultant grid will look like below.

+ Add Edit Delete Cancel Update		
Customer Name	Order Date	Freight
VINET edited	6/8/2021	\$3.00

Delete a row

To delete a row, select any row and click the **Delete** toolbar button. Deleting operation will send a **DELETE** request to the OData controller with the selected record's primary key value to remove the corresponding record from the Orders table.

C#

```
[EnableQuery]
public long Delete([FromODataUri] long key)
{
    var deleterow = _db.Orders.Find(key);
    _db.Orders.Remove(deleterow);
    _db.SaveChanges();
    return key;
}
```

Please find the sample from this [Github](#) location.

Bind data to the Syncfusion Blazor components using WebApiAdaptor of SfDataManager and perform CRUD operations

In this topic, we are going to learn how to retrieve data from WebApi Controller, bind to Grid component using [WebApiAdaptor](#) of [SfDataManger](#), and perform CRUD operations.

You can use the WebApiAdaptor of SfDataManager to interact with Web APIs created with OData endpoint. The WebApiAdaptor is extended from the ODataAdaptor. Hence, to use WebApiAdaptor, the endpoint should understand the OData formatted queries sent along with the request.

To enable the OData query option for Web API, please refer to this [documentation](#).

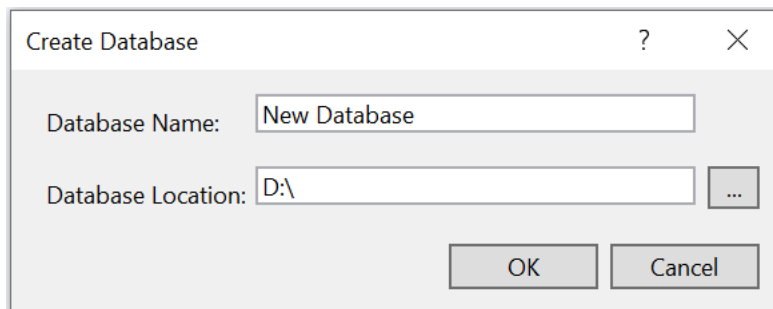
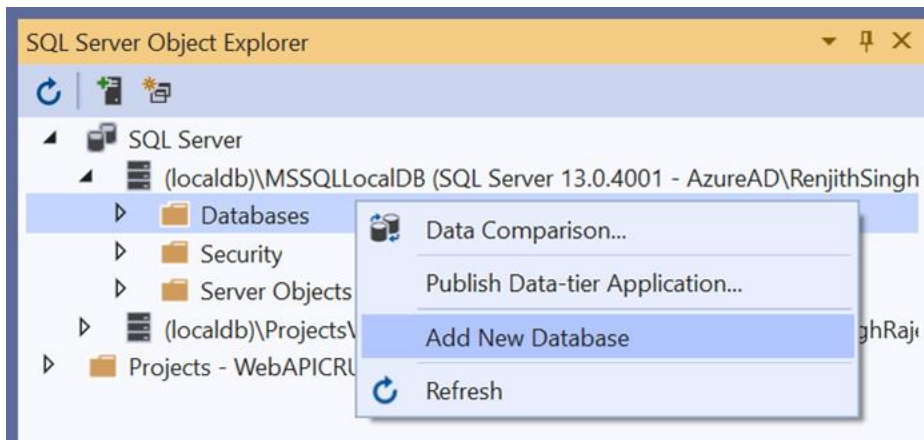
Prerequisite software

The following software are needed

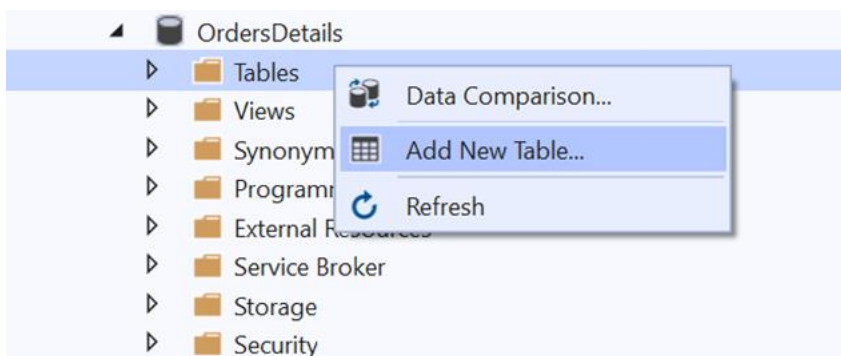
- Visual Studio 2019 v16.8.0 Preview 3.0 or later
- .NET SDK 5.0 RC2 or later.

Create the database

Open Visual Studio 2019 Preview, select **View -> SQL Server Object Explorer**. Right-click on the Databases folder to create a new Database and name it as OrdersDetails.



Right-click on the **Tables** folder of the created database and click **Add New Table**.



Use the following query to add a new table named **Orders**.

SQL

```

Create Table Orders (
OrderID BigInt Identity(1,1) Primary Key Not Null,
CustomerID Varchar(100) Not Null,
Freight int Null,
OrderDate datetime null
)

```

Now, the Orders table design will look like below. Click on the **Update** button.

dbo.Orders [Design] ✕					
Update Script File: dbo.Orders.sql					
	Name	Data Type	Allow Nulls	Default	
	OrderID	bigint	<input type="checkbox"/>		
	CustomerID	varchar(100)	<input type="checkbox"/>		
	Freight	int	<input checked="" type="checkbox"/>		
	OrderDate	datetime	<input checked="" type="checkbox"/>		
			<input type="checkbox"/>		

Now, click on **Update Database**.

Preview Database Updates
?
✕

Highlights
None

User actions
Create
[dbo].[Orders] (Table)

Supporting actions
None

☒ Include transactional scripts

Generate Script
Update Database
Cancel

Create Blazor Server Application

Open Visual Studio 2019 and follow the steps in the below documentation to create the Blazor Server Application.

Getting Started

Generate DbContext and model class from the database

Now, we are going to scaffold **DbContext** and **model classes** from the existing **OrdersDetails** database. To perform scaffolding and work with the SQL Server database in our application, we need to install the following NuGet packages.

Run the following commands in the **Package Manager Console**.

- **Install-Package Microsoft.EntityFrameworkCore.Tools -Version 3.0.0:** This package creates database context and model classes from the database.
- **Install-Package Microsoft.EntityFrameworkCore.SqlServer -Version 3.0.0:** The database provider that allows Entity Framework Core to work with SQL Server.

Once the above packages are installed, we can scaffold DbContext and Model classes. Run the following command in the **Package Manager Console**.

C#

```
using Microsoft.AspNetCore.Mvc;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using WebAPICRUDServerApp.Data;
namespace WebAPICRUDServerApp
{
    [Route("api/[controller]")]
    [ApiController]
    public class OrdersController : ControllerBase
    {
        private OrdersDetailsContext _context;
        public OrdersController(OrdersDetailsContext context)
        {
            _context = context;
        }
        // GET: api/<OrdersController>
        [HttpGet]
        public object Get()
        {
            return new { Items = _context.Orders, Count = _context.Orders.Count() };
        }
        // POST api/<OrdersController>
        [HttpPost]
        public void Post([FromBody] Orders book)
        {
            _context.Orders.Add(book);
            _context.SaveChanges();
        }
        // PUT api/<OrdersController>
        [HttpPut]
        public void Put(long id, [FromBody] Orders book)
```

```
{
Orders _book = _context.Orders.Where(x =>
x.OrderId.Equals(book.OrderId)).FirstOrDefault();
_book.CustomerId = book.CustomerId;
_book.Freight = book.Freight;
_book.OrderDate = book.OrderDate;
_context.SaveChanges();
}
// DELETE api/<OrdersController>
[HttpDelete("{id}")]
public void Delete(long id)
{
Orders _book = _context.Orders.Where(x =>
x.OrderId.Equals(id)).FirstOrDefault();
_context.Orders.Remove(_book);
_context.SaveChanges();
}
}
```

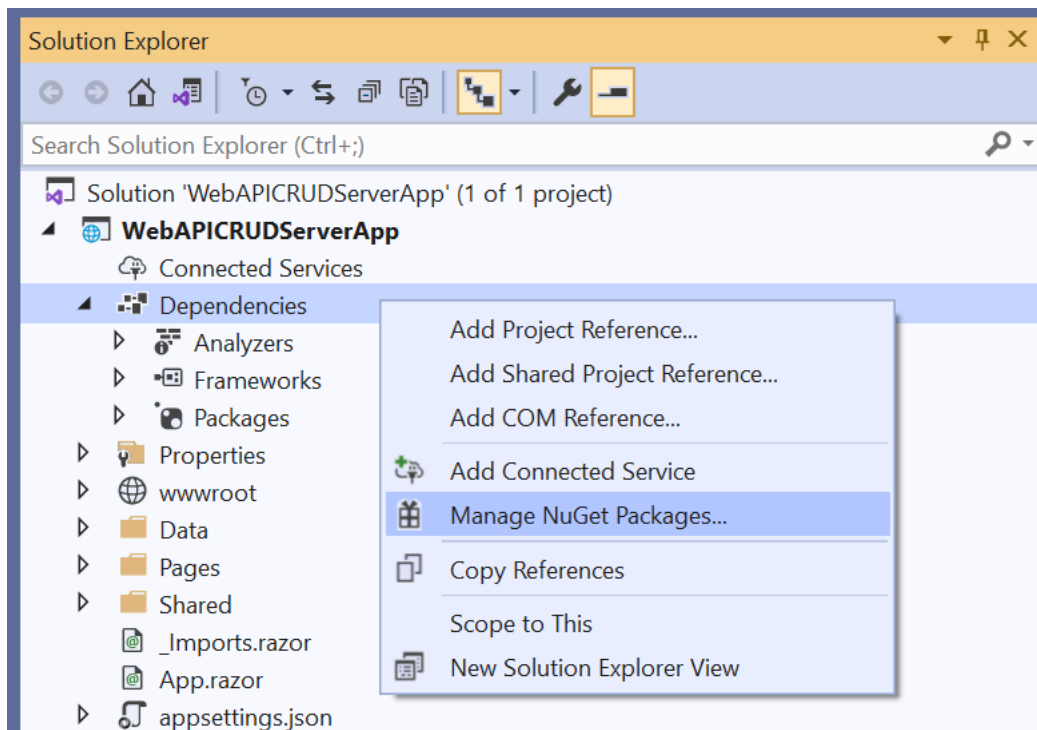
Open **Startup.cs** file and add **MapDefaultControllerRoute** in **Configure** method as follows.

C#

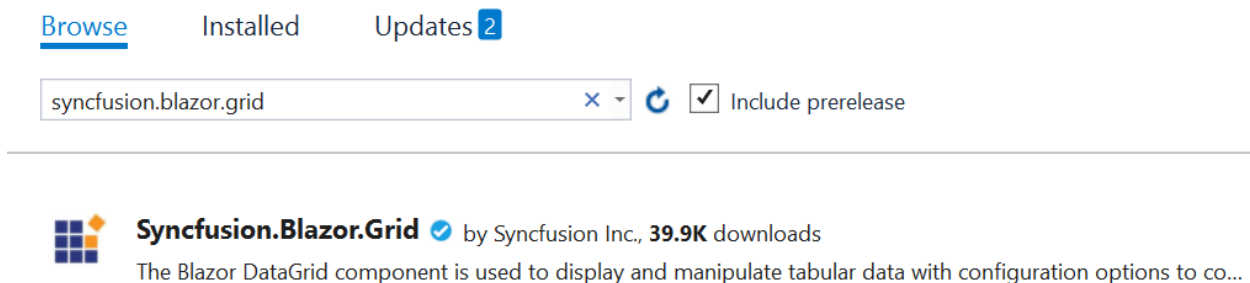
```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
...
app.UseEndpoints(endpoints =>
{
endpoints.MapDefaultControllerRoute();
endpoints.MapBlazorHub();
endpoints.MapFallbackToPage("/_Host");
});
}
```

[Add Syncfusion Blazor DataGrid package](#)

To add Syncfusion components into the project, right-click **Dependencies** and select the **Manage NuGet Packages**.



Now, in the **Browse** tab, search and install the Syncfusion.Blazor.Grid NuGet package.



For this demo, we have used Syncfusion.Blazor(**19.1.0.66**) NuGet package. We have released a new **Syncfusion.Blazor** NuGet package with new enhancement in our every-week release and main release. So, you can check and update to the latest versions by using this [link](#).

Open **_Import.razor** file and add the following namespaces which are required to use Syncfusion Blazor components in this application.

ASPX-CS

```
@using Syncfusion.Blazor
@using Syncfusion.Blazor.Data
@using Syncfusion.Blazor.Grids
@using WebAPICRUDServerApp.Data
```

Open **Startup.cs** file and register the Syncfusion service in the **ConfigureServices** method as follows.

C#

```
public void ConfigureServices(IServiceCollection services)
```

```
{
    services.AddDbContext<OrdersDetailsContext>(option =>
        option.UseSqlServer(Configuration.GetConnectionString("OrdersDetailsDatabase")));
    services.AddRazorPages();
    services.AddServerSideBlazor();
    services.AddSingleton<WeatherForecastService>();
    services.AddSyncfusionBlazor();
}
```

Themes provide life to components. Syncfusion Blazor has different themes. They are:

- Bootstrap4
- Material
- Office 365
- Bootstrap
- High Contrast

In this demo application, the **Bootstrap4** theme will be used. To add the theme, open **Pages/_Host.cshtml** file and add the following CSS reference code.

HTML

```
<link href="_content/Syncfusion.Blazor.Themes/fabric.css" rel="stylesheet" />
```

Add Syncfusion Blazor DataGrid component to an application

In previous steps, we have successfully configured the Syncfusion Blazor package in the application. Now, we can add the grid component to the **Index.razor** page.

ASPX-CS

```
<SfGrid TValue="Orders"></SfGrid>
```

Binding data to Blazor DataGrid component using WebApiAdaptor

To consume data from the WebApi Controller, we need to add the **SfDataManager** with **WebApiAdaptor**. Please refer to the following documentation for more details on WebApiAdaptor.

[WebApiAdaptor](#)

ASPX-CS

```
<SfGrid TValue="Orders">
    <SfDataManager Url="api/Orders"
        Adaptor="Adaptors.WebApiAdaptor"></SfDataManager>
</SfGrid>
```

Grid columns can be defined by using the [GridColumn](#) component. We are going to create columns using the following code.

ASPX-CS

```
<SfGrid TValue="Orders">
```

```
<SfDataManager Url="api/Orders"
Adaptor="Adaptors.WebApiAdaptor"></SfDataManager>
<GridColumn>
<GridColumn Field=@nameof(Orders.OrderId) HeaderText="Order ID"
IsPrimaryKey="true" Visible="false" TextAlign="TextAlign.Right"
Width="120"></GridColumn>
<GridColumn Field=@nameof(Orders.CustomerId) HeaderText="Customer Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Orders.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Orders.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumn>
</SfGrid>
```

When you run the application, the `Get()` method will be called in your API controller.

C#

```
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using WebAPICRUDServerApp.Data;
namespace WebAPICRUDServerApp
{
[Route("api/[controller]")]
[ApiController]
public class OrdersController : ControllerBase
{
private OrdersDetailsContext _context;
public OrdersController(OrdersDetailsContext context)
{
_context = context;
}
// GET: api/<OrdersController>
[HttpGet]
public object Get()
{
return new { Items = _context.Orders, Count = _context.Orders.Count() };
}
...
}
}
```

The response object from the Web API should contain the properties, `Items` and `Count`, whose values are a collection of entities and the total count of the entities, respectively.

The sample response object should look like this:

C#

```
{
"Items": [{..}, {..}, {..}, ...],
"Count": 830
}
```


Handling CRUD operations with our Syncfusion Blazor DataGrid component

We can enable editing in the grid component using the [GridEditSettings](#) component. Grid provides various modes of editing options such as [Inline/Normal](#), [Dialog](#), and [Batch](#) editing.

Here, we are using **Inline** edit mode and used Toolbar property to show toolbar items for editing.

We have added the DataGrid Editing and Toolbar code with previous Grid model.

ASPX-CS

```
<SfGrid TValue="Orders" Toolbar="@ (new List<string>() { "Add", "Edit",
"Delete", "Cancel", "Update" })">
<SfDataManager Url="api/Orders"
Adaptor="Adaptors.WebApiAdaptor"></SfDataManager>
<GridEditSettings AllowAdding="true" AllowDeleting="true"
AllowEditing="true" Mode="EditMode.Normal"></GridEditSettings>
<GridColumn>
<GridColumn Field=@nameof(Orders.OrderId) HeaderText="Order ID"
IsPrimaryKey="true" Visible="false" TextAlign="TextAlign.Right"
Width="120"></GridColumn>
<GridColumn Field=@nameof(Orders.CustomerId) HeaderText="Customer Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Orders.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Orders.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumn>
</SfGrid>
```

Normal editing is the default edit mode for the DataGrid component. Set the [IsPrimaryKey](#) property of Column as **true** for a particular column, whose value is a unique value for editing purposes.

Insert a row

To insert a new row, click the **Add** toolbar button. The new record edit form will look like below.

Customer Name	Order Date	Freight
VINET	6/8/2021	3

Clicking the **Update** toolbar button will insert the record in the Orders table by calling the following **POST** method of the Web API.

C#

```
[HttpPost]
public void Post([FromBody] Orders book)
{
    _context.Orders.Add(book);
    _context.SaveChanges();
}
```

Add Edit Delete Cancel Update		
Customer Name	Order Date	Freight
VINET	6/8/2021	\$3.00

Update a row

To edit a row, select any row and click the **Edit** toolbar button. The edit form will look like below. Edit the Customer Name column.

Add Edit Delete Cancel Update		
Customer Name	Order Date	Freight
<input type="text" value="VINET edited"/>	6/8/2021	3

Clicking the **Update** toolbar button will update the record in the Orders table by calling the following **PUT** method of the Web API.

C#

```
[HttpPut]
public void Put(long id, [FromBody] Orders book)
{
    Orders _book = _context.Orders.Where(x =>
        x.OrderId.Equals(book.OrderId)).FirstOrDefault();
    _book.CustomerId = book.CustomerId;
    _book.Freight = book.Freight;
    _book.OrderDate = book.OrderDate;
    _context.SaveChanges();
}
```

The resultant grid will look like below.

Add Edit Delete Cancel Update		
Customer Name	Order Date	Freight
VINET edited	6/8/2021	\$3.00

Delete a row

To delete a row, select any row and click the **Delete** toolbar button. Deleting operation will send a **DELETE** request to the Web API with the selected record's primary key value to remove the corresponding record from the Orders table.

C#

```
[HttpDelete("{id}")]
public void Delete(long id)
{
    Orders _book = _context.Orders.Where(x =>
        x.OrderId.Equals(id)).FirstOrDefault();
    _context.Orders.Remove(_book);
    _context.SaveChanges();
}
```

Please find the sample from this [Github](#) location.

SQL server data binding and performing CRUD operations

Introduction

This topic gives a clear idea about how to consume data from [SQL Server](#) using Microsoft SQL Client, bind it to a Syncfusion Component, and perform CRUD operations.

Prerequisite software

The following software are needed:

- Microsoft.EntityFrameworkCore.SqlServer
- Visual Studio 2019 v16.9.0 or later
- .NET SDK 5.0 or later.

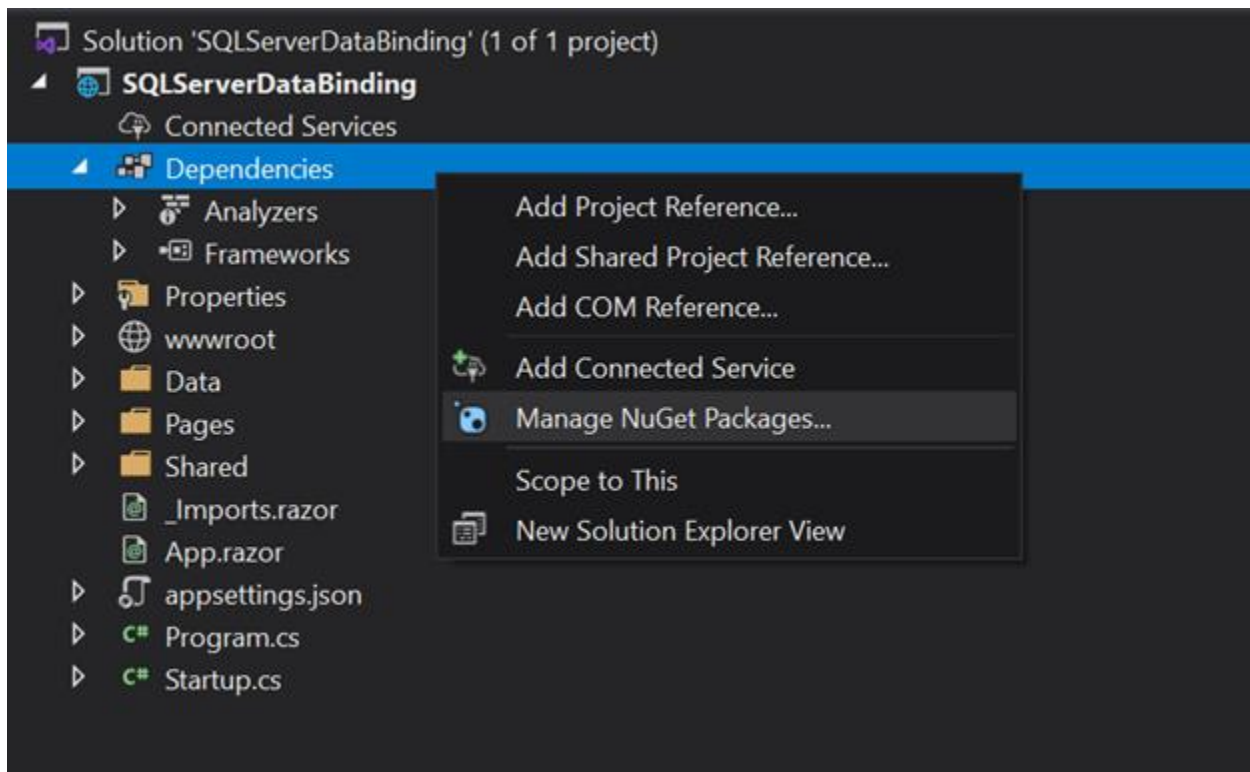
Create Blazor Server Application

Open Visual Studio 2019 and follow the steps in the [documentation](#) to create the Blazor Server Application.

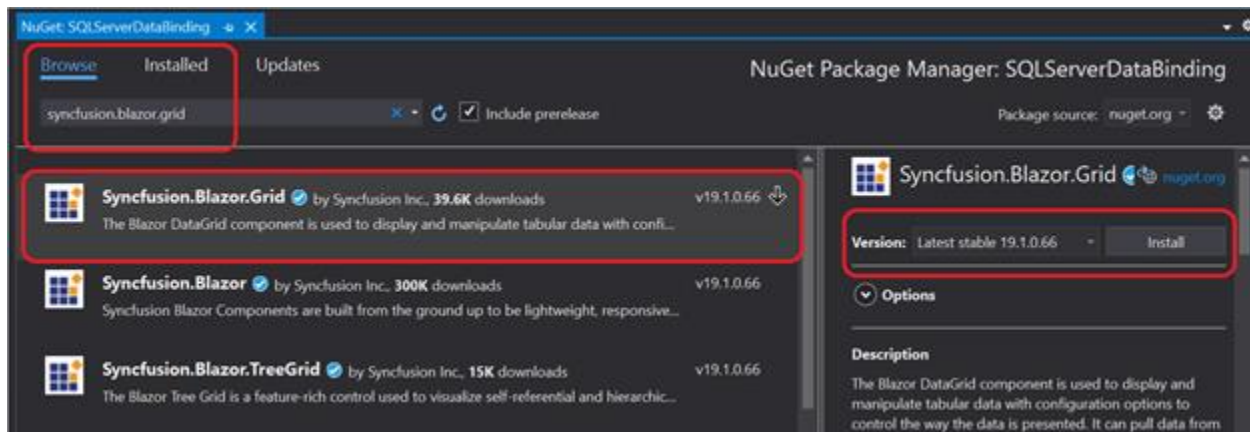
Add Syncfusion Blazor DataGrid package

The SQL server data binding process using the Blazor DataGrid component and the method of performing the CRUD operations in it are explained as follows.

The first approach is to install the necessary packages alone. Now, right-click Dependencies in the project and select Manage NuGet Packages.



Now, in the Browse tab, search and install the Syncfusion.Blazor.Grid NuGet package.



For this demo, use Syncfusion.Blazor.Grid(19.1.0.66) NuGet package. A new Syncfusion.Blazor.Grid NuGet package with new enhancement will be released in our every-week release and main release. So, check and update to the [latest versions](#).

Adding Syncfusion Blazor DataGrid Component into the application

Open **_Import.razor** file and add the following namespaces which are required to use Syncfusion Blazor DataGrid Component in this application.

ASPX-CS

```
@using Syncfusion.Blazor
@using Syncfusion.Blazor.Grids
```

Open **Startup.cs** file and register the Syncfusion service in the ConfigureServices method as follows.

C#

```
using Syncfusion.Blazor;
public class Startup
{
    public void ConfigureServices(IServiceCollection services)
    {
        services.AddRazorPages();
        services.AddServerSideBlazor();
        services.AddSyncfusionBlazor();
    }
}
```

Themes provide life to components. Syncfusion Blazor has different themes. They are as follows:

- Bootstrap4
- Material
- Office 365
- Bootstrap
- High Contrast

To add the theme, open the **Pages/_Host.cshtml** file and add the following CSS reference code.

HTML

```
<link href="_content/Syncfusion.Blazor/styles/bootstrap4.css"
rel="stylesheet" />
```

In previous steps, Syncfusion Blazor DataGrid package is successfully configured in the application. Now, add the DataGrid Component to the **Index.razor**.

ASPX-CS

```
<SfGrid TValue="Order" AllowPaging="true">
</SfGrid >
```

Binding SQL data to the Blazor DataGrid Component

Now, get the SQL data from the SQL server and bind it to the DataGrid component as a datasource by using the Custom adaptor feature. The Custom Adaptor can be created as a [Component](#). Please refer the [Grid Custom Binding](#) and [Custom adaptor as component](#) documentation for more details on the Custom adaptor.

Grid columns can be defined using the [GridColumn](#) component. Create columns using the following code. The properties used and their usage are discussed below.

ASPX-CS

```
[Index.razor]
<SfGrid @ref="Grid" TValue="Order" AllowPaging="true" >
<SfDataManager Adaptor="Adaptors.CustomAdaptor">
<CustomAdaptorComponent></CustomAdaptorComponent>
</SfDataManager>
<GridColumn>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
IsIdentity="true" IsPrimaryKey="true" TextAlign="TextAlign.Right"
Width="120">
</GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>
</GridColumn>
</SfGrid>
@code{
SfGrid<Order> Grid { get; set; }
public static List<Order> Orders { get; set; }
public class Order
{
public int? OrderID { get; set; }
public string CustomerID { get; set; }
}
}
```

In the custom adaptor's **Read** method, you can get the Grid action details like paging, filtering, sorting information, etc., using **DataManagerRequest**.

- Based on the DataManagerRequest, form a SQL query string (to perform paging) and execute the SQL query. Retrieve the data from the database using SqlDataAdapter.
- The Fill method of the DataAdapter is used to populate a DataSet with the results of the SelectCommand of the DataAdapter, then convert the DataSet into the List.

- Return the response in Result and Count pair object in Read method to bind the data to the DataGrid.

ASPX-CS

```
[CustomAdaptorComponent.razor]
@using Syncfusion.Blazor;
@using Syncfusion.Blazor.Data;
@using Newtonsoft.Json
@using static EFGrid.Pages.Index;
@using Microsoft.Data.SqlClient;
@using System.Data;
@using System.IO;
@using Microsoft.AspNetCore.Hosting;
@inject IHostingEnvironment _env
@inherits DataAdaptor<Order>
//Here, we are rendering the CustomAdaptorComponent as a child component for
the SfDataManager
<CascadingValue Value="@this">
@ChildContent
</CascadingValue>
@code {
[Parameter]
[JsonIgnore]
public RenderFragment ChildContent { get; set; }
public static DataSet CreateCommand(string queryString, string
connectionString)
{
using (SqlConnection connection = new SqlConnection(
connectionString))
{
SqlDataAdapter adapter = new SqlDataAdapter(queryString, connection);
DataSet dt = new DataSet();
try
{
connection.Open();
// Using sqlDataAdapter, we process the query string and fill the data into
the dataset
adapter.Fill(dt);
}
catch (SqlException se)
{
Console.WriteLine(se.ToString());
}
finally
{
connection.Close();
}
return dt;
}
}
// Performs data Read operation
// DataManagerRequest defines the members of the query
public override object Read(DataManagerRequest DataManagerReq, string Key =
null)
{

```

```

string AppData = _env.ContentRootPath;
string DatabasePath = Path.Combine(AppData, "App_Data\\NORTHWND.MDF");
string ConnectionStr = $"Data
Source=(LocalDB)\\MSSQLLocalDB;AttachDbFilename='{DatabasePath}';Integrated
Security=True;Connect Timeout=30";
// Here, we formed the SQL query string based on the skip and take count
from the DataManagerRequest
string QueryStr = "SELECT OrderID, CustomerID FROM dbo.Orders ORDER BY
OrderID OFFSET " + DataManagerReq.Skip + " ROWS FETCH NEXT " +
DataManagerReq.Take + " ROWS ONLY;";
DataSet Data = CreateCommand(QueryStr, ConnectionStr);
Orders = Data.Tables[0].AsEnumerable().Select(r => new Order
{
    OrderID = r.Field<int>("OrderID"),
    CustomerID = r.Field<string>("CustomerID")
}).ToList(); // Here, we convert dataset into list
IEnumerable<Order> DataSource = Orders;
SqlConnection Con = new SqlConnection(ConnectionStr);
Con.Open();
SqlCommand Cmd = new SqlCommand("SELECT COUNT(*) FROM dbo.Orders", Con);
Int32 Count = (Int32)Cmd.ExecuteScalar();
return DataManagerReq.RequiresCounts ? new DataResult() { Result =
DataSource, Count = Count } : (object)DataSource;
}
}

```

While running the application, the grid will be displayed as follows.

Order ID	Customer Name
10256	WELLI
10257	HILAA
10258	ERNBHI
10259	CENTC
10260	OTTIK
10261	QUEDE
10262	RATTIC
10263	ERNBHI
10264	FOLKID
10265	BLCNP
10266	WARTH
10267	FRANK

Handling CRUD operations with our Syncfusion Blazor DataGrid component

Enable editing in the grid component using the [GridEditSettings](#) component. Grid provides various modes of editing options such as Inline/Normal, Dialog and Batch editing. Kindly refer the [Grid Editing](#) documentation for reference.

Here, inline edit mode and [Toolbar](#) property are used to show toolbar items for editing.

ASPX-CS

```

<SfGrid @ref="Grid" TValue="Order" AllowPaging="true" Toolbar="@ (new
List<string>() { "Add", "Edit", "Delete", "Update", "Cancel" }) ">

```

```
<GridEditSettings AllowAdding="true" AllowDeleting="true"
AllowEditing="true"></GridEditSettings>
</SfGrid>
```

Normal editing is the default edit mode for the DataGrid component. Also, to perform CRUD operations, set `IsPrimaryKey` property as `True` for a particular `GridColumn`, whose value is a unique.

The CRUD operations can be performed and customized on our own by overriding the following CRUD methods of the `DataAdaptor` abstract class.

- `Insert/InsertAsync`
- `Remove/RemoveAsync`
- `Update/UpdateAsync`
- `BatchUpdate/BatchUpdateAsync`

Let's see how to perform CRUD operation using SQL server data with Syncfusion Blazor DataGrid component

Insert Operation

To Perform the Insert operation, override the `Insert/InsertAsync` method of the custom adaptor and add the following code in the `CustomAdaptorComponent.razor`.

C#

```
// Performs Insert operation
//You will get the DataManager instance in the DataManager parameter
//You will get the record in the Value parameter
public override object Insert(DataManager DataManager, object Value, string
Key)
{
    //Here, you can implement your own code to update the record from the grid.
    string AppData = _env.ContentRootPath;
    string DatabasePath = Path.Combine(AppData, "App_Data\\NORTHWND.MDF");
    string ConnectionStr = $"Data
Source=(LocalDB)\\MSSQLLocalDB;AttachDbFilename='{DatabasePath}';Integrated
Security=True;Connect Timeout=30";
    string QueryStr = $"Insert into Orders(CustomerID) values('{(Value as
Order).CustomerID}')"
    SqlConnection Con = new SqlConnection(ConnectionStr);
    try
    {
        Con.Open();
        SqlCommand Cmd = new SqlCommand(QueryStr, Con);
        Cmd.ExecuteNonQuery();
    }
    catch (SqlException Exception)
    {
        Console.WriteLine(Exception.ToString());
    }
    finally
    {
        Con.Close();
    }
    return Value;
}
```


The resultant grid will look like below.

+ Add Edit Delete Update X Cancel	
Order ID	Customer Name
11071	LIAS
11072	ERN9H
11073	PERIC
11074	SIMOB
11075	RIC9J
11076	BONAP
11077	RATTG
11079	Check
11081	VIGN
11091	added

69 of 69 pages (826 items)

Update Operation

To Perform the Update operation, override the Update/UpdateAsync method of the custom adaptor and add the following code in the CustomAdaptorComponent.razor.

C#

```
// Performs Update operation
//You will get the DataManager instance in the DataManager parameter
//You will get the edited record in the Value parameter
//You will get the PrimaryKey field in the KeyField parameter
public override object Update(DataManager DataManager, object Value, string
KeyField, string Key)
{
    //Here, you can implement your own code to update the record from the grid.
    string AppData = _env.ContentRootPath;
    string DatabasePath = Path.Combine(AppData, "App_Data\\NORTHWND.MDF");
    string ConnectionStr = $"Data
Source=(LocalDB)\\MSSQLLocalDB;AttachDbFilename='{DatabasePath}';Integrated
Security=True;Connect Timeout=30";
    string QueryStr = $"Update Orders set CustomerID='{(Value as
Order).CustomerID}' where OrderID='{(Value as Order).OrderID}";
    SqlConnection Con = new SqlConnection(ConnectionStr);
    try
    {
        Con.Open();
        SqlCommand Cmd = new SqlCommand(QueryStr, Con);
        Cmd.ExecuteNonQuery();
    }
    catch (SqlException Exception)
    {
        Console.WriteLine(Exception.ToString());
    }
    finally
    {
        Con.Close();
    }
    return Value;
}
```

The resultant grid will look like below.

		Order ID	Customer Name
		10255	test
		10256	WELLI
		10257	HILAA
		10258	ERNBH
		10259	GENTC
		10260	OTTIK
		10261	QUEDE
		10262	RATTC
		10263	ERNBH
		10264	FOLKD
		10265	BLONP
		10266	WARTH

1 of 69 pages (826 items)

Delete Operation

To Perform the Delete operation, override the Remove/RemoveAsync method of the custom adaptor and add the following code in the CustomAdaptorComponent.razor.

C#

```
// Performs Remove operation
//You will get the DataManager instance in the DataManager parameter
//You will get the record in the Value parameter
//You will get the PrimaryKey field in the KeyField parameter
public override object Remove(DataManager DataManager, object Value, string
KeyField, string Key)
{
    //Here, you can implement your own code to delete the record from the grid.
    string AppData = _env.ContentRootPath;
    string DatabasePath = Path.Combine(AppData, "App_Data\\NORTHWND.MDF");
    string Connectionstr = $"Data
Source=(LocalDB)\\MSSQLLocalDB;AttachDbFilename='{DatabasePath}';Integrated
Security=True;Connect Timeout=30";
    string QueryStr = $"Delete from Orders where OrderID={Value}";
    SqlConnection Con = new SqlConnection(Connectionstr);
    try
    {
        Con.Open();
        SqlCommand Cmd = new SqlCommand(QueryStr, Con);
        Cmd.ExecuteNonQuery();
    }
    catch (SqlException Exception)
    {
        Console.WriteLine(Exception.ToString());
    }
    finally
    {
        Con.Close();
    }
    return Value;
}
```

The resultant grid will look like below.

+ Add Edit Delete Update X Cancel	
Order ID	Customer Name
10256	WELLI
10257	HLAA
10258	ERNSH
10259	CENTC
10260	OTTIK
10261	QUEDE
10262	RATTIC
10263	ERNSH
10264	FOLKO
10265	BLCNP
10266	WARTH
10267	FRANK

1 of 69 pages (825 items)

You can find the sample in this [GitHub location](#)

Bind data from SQL server to Syncfusion Blazor components

In this topic, we are going to learn how to retrieve data from SQL database using [Entity Framework](#) to bind it to the Grid component and perform CRUD operations.

Entity Framework is an open-source object-relational mapper (O/RM) from Microsoft. Entity Framework works with many databases. But here, we are going to discuss the step-by-step procedure to create an Entity Framework using the [MS SQL Server](#) database and connect it to the Syncfusion component to perform CRUD operations in a Blazor Server Application.

Prerequisite software

The following software are needed

- Visual Studio 2019 v16.9.0 or later
- .NET SDK 5.0 or later.
- SQL Server 2019

Create the database

The first step is to create a Library database and a table named Book to hold a list of books.

- Open SQL Server 2019.
- Now, create a new database named Library.
- Right-click on the created database and select New Query.
- Use the following SQL query to create a table named Book.

C#

```
using Microsoft.EntityFrameworkCore;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
namespace LibraryManagement.Models
{
    public class LibraryService : ILibraryService
    {

```

```
private LibraryContext _context;
public LibraryService(LibraryContext context)
{
    _context = context;
}
public void DeleteBook(long id)
{
    try
    {
        Book ord = _context.Books.Find(id);
        _context.Books.Remove(ord);
        _context.SaveChanges();
    }
    catch
    {
        throw;
    }
}
public IEnumerable<Book> GetBooks()
{
    try
    {
        return _context.Books.ToList();
    }
    catch
    {
        throw;
    }
}
public void InsertBook(Book book)
{
    try
    {
        _context.Books.Add(book);
        _context.SaveChanges();
    }
    catch
    {
        throw;
    }
}
public Book SingleBook(long id)
{
    throw new NotImplementedException();
}
public void UpdateBook(long id, Book book)
{
    try
    {
        var local = _context.Set<Book>().Local.FirstOrDefault(entry =>
            entry.Id.Equals(book.Id));
        // check if local is not null
        if (local != null)
        {
            // detach
            _context.Entry(local).State = EntityState.Detached;
        }
    }
}
```

```
_context.Entry(book).State = EntityState.Modified;
_context.SaveChanges();
}
catch
{
    throw;
}
}
}
```

Register the Service in Startup.cs

Now, we need to register the **LibraryService** and **ILibraryService** as services in the **startup.cs** file. Kindly register the Scoped Services like below.

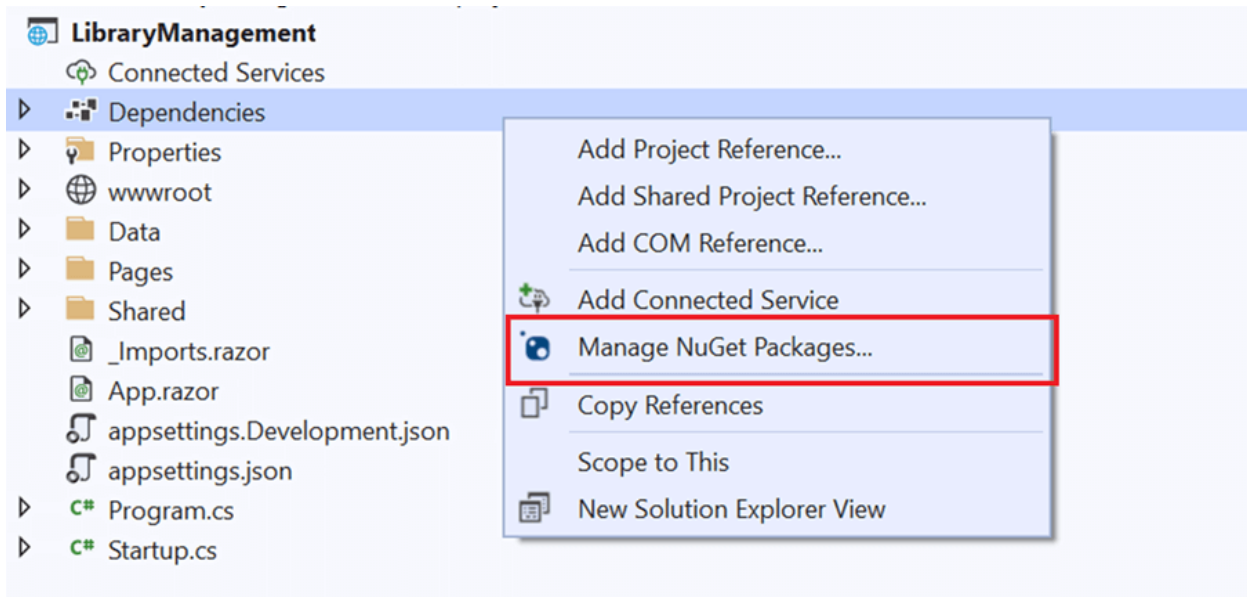
```
namespace LibraryManagement
{
    public class Startup
    {
        public Startup(IConfiguration configuration)
        {
            Configuration = configuration;
        }

        public IConfiguration Configuration { get; }

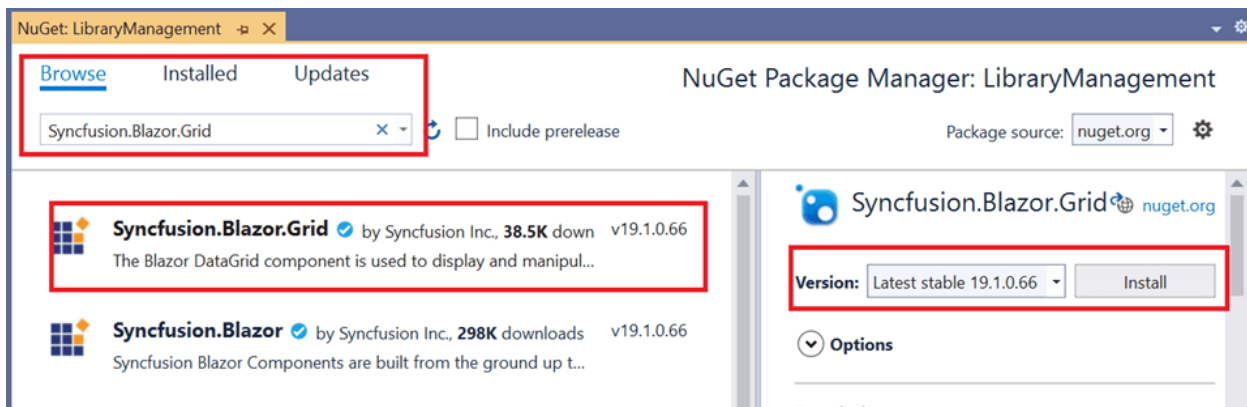
        // This method gets called by the runtime. Use this method to add services to the container.
        // For more information on how to configure your application, visit https://go.microsoft.com/fwlink/?LinkID=398940
        public void ConfigureServices(IServiceCollection services)
        {
            services.AddScoped<ILibraryService, LibraryService>();
            services.AddDbContext<LibraryContext>(option =>
                option.UseSqlServer(Configuration.GetConnectionString("LibraryDatabase")));
            services.AddRazorPages();
            services.AddServerSideBlazor();
            services.AddSingleton<WeatherForecastService>();
        }
    }
}
```

Add Syncfusion Blazor DataGrid package

Before adding Syncfusion Blazor components to the application, we need to move Book.cs file to the LibraryManagement.Models project. Now, we are going to add the **Syncfusion DataGrid** component into the project. Since we are going to explain this process (Data binding and CRUD operation) with the help of the DataGrid component, Right-click on Dependencies and select Manage NuGet Packages to load the required assemblies.



Now, in the Browse tab, search and install the **Syncfusion.Blazor.Grid** NuGet package.



For this demo, we have used **Syncfusion.Blazor.Grid(19.1.0.66)** NuGet package. We have released a new Syncfusion.Blazor NuGet package with new enhancement in our every-week release and main release. So, you can check and update to the latest versions by using the following link.

[Grid Editing](#)

Open **_Import.razor** file and add the following namespaces which are required to use the Syncfusion Blazor components in this application.

ASPX-CS

```
@using Syncfusion.Blazor
@using Syncfusion.Blazor.Grids
```

Open **Startup.cs** file and register the Syncfusion service in the **ConfigureServices** method as follows.

C#

```
using Syncfusion.Blazor;
namespace LibraryManagement
{
```

```
public class Startup
{
    public Startup(IConfiguration configuration)
    {
        Configuration = configuration;
    }
    public IConfiguration Configuration { get; }
    // This method gets called at the runtime. Use this method to add services
    // to the container.
    // For more information on how to configure your application, visit
    // https://go.microsoft.com/fwlink/?LinkID=398940
    public void ConfigureServices(IServiceCollection services)
    {
        services.AddRazorPages();
        services.AddServerSideBlazor();
        services.AddSyncfusionBlazor();
        services.AddSingleton<WeatherForecastService>();
    }
    // This method gets called at the runtime. Use this method to configure the
    // HTTP request pipeline.
    public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
    {
    }
}
```

Themes provide life to components. Syncfusion Blazor has different themes. They are:

- Fabric
- Bootstrap4
- Material
- Bootstrap
- High Contrast

In this demo application, the Fabric theme will be used. To add the theme, open the **Pages/_Host.cshtml** file and add the following CSS reference code.

HTML

```
<link href="_content/Syncfusion.Blazor.Themes/fabric.css" rel="stylesheet" />
```

Add Syncfusion Blazor DataGrid component to an application

In previous steps, we have successfully configured the Syncfusion Blazor package in the application. Now, we can add the grid component to the **Index.razor** page.

ASPX-CS

```
<SfGrid TValue="Book">
</SfGrid>
```

Bind data to Blazor DataGrid component using Entity Framework

To consume data from the database using **Entity Framework**, we need to inject the **LibraryService** into the razor page and assign it to the DataGrid's datasource variable. Here, we have used the **DataSource**

property of the DataGrid component to bind the SQL data using Entity Framework in the Server-side application

ASPX-CS

```
@using LibraryManagement.Models
@inject ILibraryService LibraryService
<SfGrid DataSource="@LibraryBooks" TValue="Book">
</SfGrid>
@code
{
public IEnumerable<Book> LibraryBooks { get; set; }
protected override void OnInitialized()
{
LibraryBooks = LibraryService.GetBooks();
}
}
```

Grid columns can be defined using the **GridColumn** component. We are going to create columns using the following code. Let us see the properties used and their usage.

- **Field** property specifies the column name of the Book table to display in the grid column.
- **IsPrimaryKey** property specifies that the given column is a primary key column. Here, Id column is a primary key column.
- **Visible** property specifies the column visibility. Setting as false will hide the column at the user end.
- **Width** property specifies the column width.
- **Format** property helps to format number, currencies, and date in a particular culture. Here, we have formatted the Price column.
- **DisplayAsCheckBox** property renders checkbox in cells and sets check state based on the property value. Here, Available column is rendered as a checkbox column.

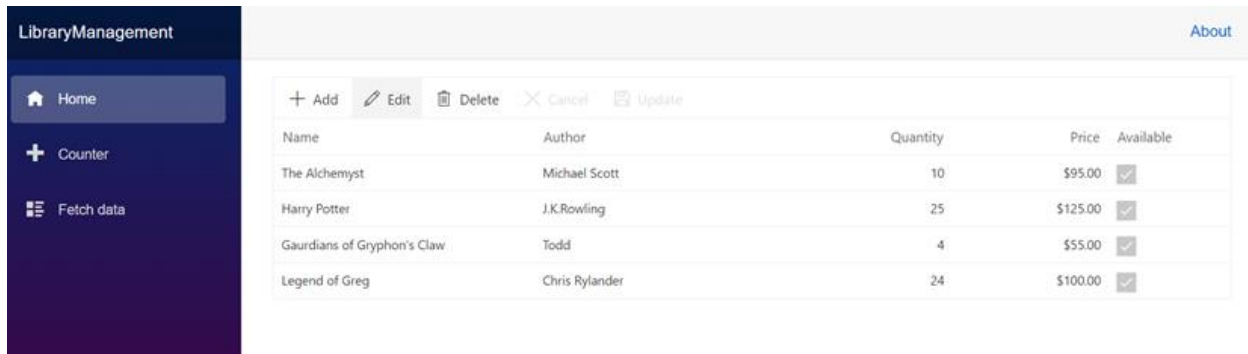
ASPX-CS

```
@using LibraryManagement.Models
@inject ILibraryService LibraryService
<SfGrid DataSource="@LibraryBooks" TValue="Book">
<GridColumn>
<GridColumn Field="@nameof(Book.Id)" IsPrimaryKey="true" IsIdentity="true"
Visible="false"></GridColumn>
<GridColumn Field="@nameof(Book.Name)" Width="150"></GridColumn>
<GridColumn Field="@nameof(Book.Author)" Width="150"></GridColumn>
<GridColumn Field="@nameof(Book.Quantity)" Width="90"
TextAlign="TextAlign.Right"></GridColumn>
<GridColumn Field="@nameof(Book.Price)" Width="90" Format="C2"
TextAlign="TextAlign.Right"></GridColumn>
<GridColumn Field="@nameof(Book.Available)" DisplayAsCheckBox="true"
Width="70"></GridColumn>
</GridColumn>
</SfGrid>
@code
{
public IEnumerable<Book> LibraryBooks { get; set; }
protected override void OnInitialized()
```



```
{
LibraryBooks = LibraryService.GetBooks();
}
}
```

Now, the data from the SQL server is loaded into the DataGrid component. Refer to the following screenshot for the output of above.



Handling CRUD operations with our Syncfusion Blazor DataGrid component

We can enable editing in the grid component using the **GridEditSettings** component. Grid provides various modes of editing options such as Inline/Normal, Dialog, and Batch editing. Kindly refer to the following documentation for your reference

[Grid Editing](#)

Here, we are using inline edit mode and **Toolbar** property to show toolbar items for editing.

While using the DataSource property of Grid, changes will be reflected only in the Grid datasource. To reflect them in the database, we need to handle the CRUD operations externally using the OnActionBegin and OnActionComplete events of Grid.

- **OnActionBegin** – This event will be triggered when the action gets initiated. So, while inserting/updating a record, RequestType Save will be sent in the event arguments to save the changes in the database. Similarly, while deleting a record, RequestType as Delete will be initiated to perform actions externally. Since for both Update and Insert action, RequestType will be Save, we can differentiate them by using the Args.Action property, which will indicate the current action.
- **OnActionComplete** – It will be triggered when certain actions are completed. Here, we can refresh the Grid component with an updated datasource to reflect the changes.

We have added the DataGrid editing, toolbar, and OnActionBegin and OnActionComplete event code with the previous Grid model.

ASPX-CS

```
@using LibraryManagement.Models
@inject ILibraryService LibraryService
<SfGrid DataSource="@LibraryBooks" Toolbar="@ (new List<string>() { "Add",
"Edit", "Delete", "Cancel", "Update" })" TValue="Book">
<GridEditSettings AllowAdding="true" AllowDeleting="true"
AllowEditing="true" Mode="EditMode.Normal"></GridEditSettings>
```

```

<GridEvents OnActionBegin="ActionBeginHandler"
OnActionComplete="ActionCompleteHandler" TValue="Book"></GridEvents>
<GridColumn>
<GridColumn Field="@nameof(Book.Id)" IsPrimaryKey="true" IsIdentity="true"
Visible="false"></GridColumn>
<GridColumn Field="@nameof(Book.Name)" Width="150"></GridColumn>
<GridColumn Field="@nameof(Book.Author)" Width="150"></GridColumn>
<GridColumn Field="@nameof(Book.Quantity)" Width="90"
TextAlign="TextAlign.Right"></GridColumn>
<GridColumn Field="@nameof(Book.Price)" Width="90" Format="C2"
TextAlign="TextAlign.Right"></GridColumn>
<GridColumn Field="@nameof(Book.Available)" DisplayAsCheckBox="true"
Width="70"></GridColumn>
</GridColumn>
</SfGrid>
@code
{
public IEnumerable<Book> LibraryBooks { get; set; }
protected override void OnInitialized()
{
LibraryBooks = LibraryService.GetBooks();
}
public void ActionBeginHandler(ActionEventArgs<Book> Args)
{
//Will be triggered when CRUD action is initiated
}
public void ActionCompleteHandler(ActionEventArgs<Book> Args)
{
//will be triggered when CRUD action is complete.
if (Args.RequestType.Equals(Syncfusion.Blazor.Grids.Action.Save))
{
LibraryBooks = LibraryService.GetBooks(); //to fetch the updated data from
db to Grid
}
}
}
}

```

Normal edit mode is the default mode of editing.

[Insert a row](#)

To insert a new row, click the **Add** toolbar button. The new record edit form will look like below.

Name	Author	Quantity	Price	Available
Harry Potter	J.K.Rowling	15	125	<input checked="" type="checkbox"/>

Clicking the **Update** toolbar button will initiate the insert action in Grid. Now, the **OnActionBegin** event will be triggered with a **RequestType** as **Save**. We can insert the record into our database (Book table) by calling the **InsertBook()** method of the **LibraryService**.

C#

```

public void ActionBeginHandler(ActionEventArgs<Book> Args)
{
    if (Args.RequestType.Equals(Syncfusion.Blazor.Grids.Action.Save))
    {
        if (Args.Action == "Add")
        {
            // Insert the changes into your database here.
            LibraryService.InsertBook(Args.Data);
        }
    }
}

```

+ Add ✎ Edit 🗑 Delete ✕ Cancel 🔄 Update				
Name	Author	Quantity	Price	Available
Harry Potter	J.K.Rowling	15	\$125.00	<input checked="" type="checkbox"/>

Update a row

To edit a row, select any row and click the **Edit** toolbar button. The edit form will look like below.

+ Add ✎ Edit 🗑 Delete ✕ Cancel 🔄 Update				
Name	Author	Quantity	Price	Available
<input type="text" value="Harry Potter"/>	<input type="text" value="J.K.Rowling"/>	<input type="text" value="15"/> ▼ ▲	<input type="text" value="250"/> ▼ ▲	<input checked="" type="checkbox"/>

Now, we have changed the Price column value to 125 from 250. Clicking the **Update** toolbar button will initiate the update action and trigger the OnActionBegin event with **Save RequestType**. Here, we can update the record in the Book table by calling the **UpdateBook()** method of the LibraryService when **Args.Action** is **Edit**. Refer to the following code example.

C#

```

public void ActionBeginHandler(ActionEventArgs<Book> Args)
{
    if (Args.RequestType.Equals(Syncfusion.Blazor.Grids.Action.Save))
    {
        if (Args.Action == "Edit")
        {
            //Update the changes into your database here.
            LibraryService.UpdateBook(Args.Data.Id, Args.Data);
        }
    }
}

```

The resultant grid will look like below.

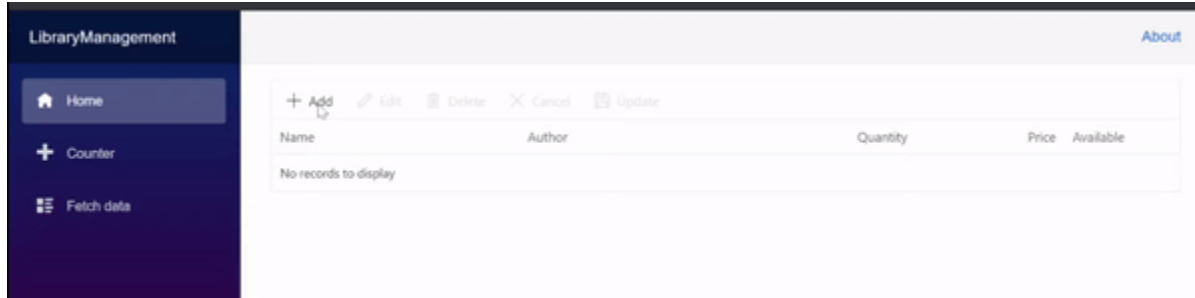
+ Add ✎ Edit 🗑 Delete ✕ Cancel 🔄 Update				
Name	Author	Quantity	Price	Available
Harry Potter	J.K.Rowling	15	\$250.00	<input checked="" type="checkbox"/>

Delete a row

To delete a row, select any row and click the **Delete** toolbar button. Deleting operation will initiate the OnActionBegin event with RequestType as Delete. Now, we can delete the record from the database by calling **DeleteBook()** method of LibraryService with the selected record's primary key value. Refer to the following code example.

C#

```
public void ActionBeginHandler (ActionEventArgs<Book> Args)
{
    if (Args.RequestType.Equals(Syncfusion.Blazor.Grids.Action.Delete))
    {
        //Remove the record from your database
        LibraryService.DeleteBook(Args.Data.Id);
    }
}
```



Please find the sample from this [Github](#) location.

How to bind data using Dapper and perform CRUD operations

In this topic, we are going to discuss how to consume data from a database using [Dapper](#), bind it to a Syncfusion Blazor Component, and perform CRUD operations.

Prerequisite software

- Visual Studio 2019.
- MS SQL Server.

Creating Blazor server-side application

Open Visual Studio 2019 and follow the steps in the below documentation to create the Blazor Server Application.

[Creating Blazor Server Application](#)

Creating the database

First, create a database named **BugTracker** and a table named **Bugs** to hold the list of bugs.

1. Open SQL Server 2017 / latest version.
2. Now, create a new database named **BugTracker**.
3. Right-click on the created database and select New Query.
4. Use the following SQL query to create a table named Bugs.

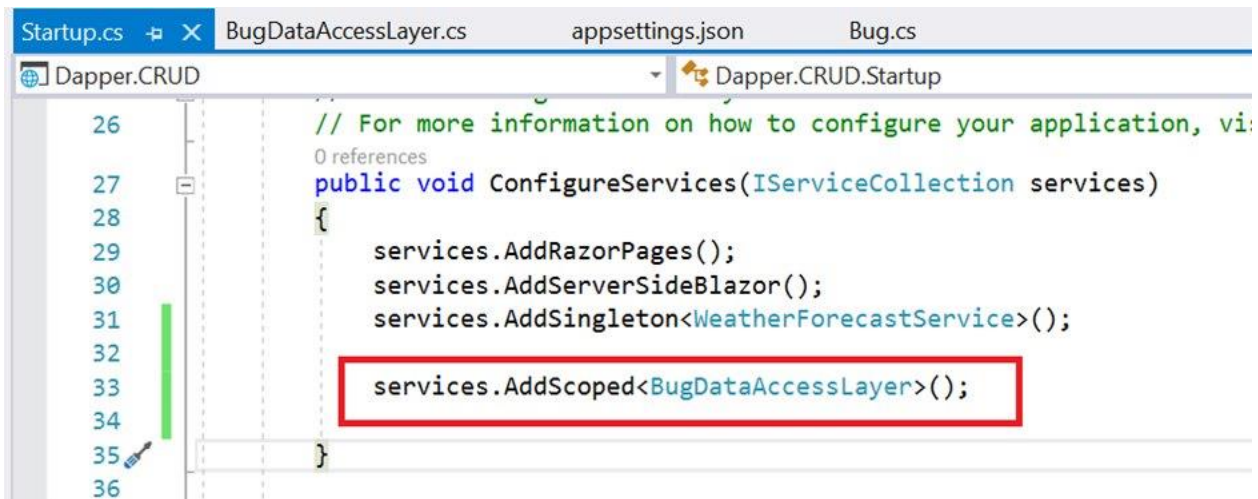
C#

```
public class BugDataAccessLayer
{
    public IConfiguration Configuration;
    private const string BUGTRACKER_DATABASE = "BugTrackerDatabase";
    private const string SELECT_BUG = "select * from bugs";
    public BugDataAccessLayer(IConfiguration configuration)
    {
        Configuration = configuration; //Inject configuration to access Connection string from appsettings.json.
    }
    public async Task<List<Bug>> GetBugsAsync()
    {
        using (IDbConnection db = new
        SqlConnection(Configuration.GetConnectionString(BUGTRACKER_DATABASE)))
        {
            db.Open();
            IEnumerable<Bug> result = await db.QueryAsync<Bug>(SELECT_BUG);
            return result.ToList();
        }
    }
    public async Task<int> GetBugCountAsync()
    {
        using (IDbConnection db = new
        SqlConnection(Configuration.GetConnectionString(BUGTRACKER_DATABASE)))
        {
            db.Open();
            int result = await db.ExecuteScalarAsync<int>("select count(*) from bugs");
            return result;
        }
    }
    public async Task AddBugAsync(Bug bug)
    {
        using (IDbConnection db = new
        SqlConnection(Configuration.GetConnectionString(BUGTRACKER_DATABASE)))
        {
            db.Open();
            await db.ExecuteNonQuery("insert into bugs (Summary, BugPriority, Assignee, BugStatus) values (@Summary, @BugPriority, @Assignee, @BugStatus)", bug);
        }
    }
    public async Task UpdateBugAsync(Bug bug)
    {
        using (IDbConnection db = new
        SqlConnection(Configuration.GetConnectionString(BUGTRACKER_DATABASE)))
        {

```

```
db.Open();
await db.ExecuteNonQuery("update bugs set Summary=@Summary,
BugPriority=@BugPriority, Assignee=@Assignee, BugStatus=@BugStatus where
id=@Id", bug);
}
}
public async Task RemoveBugAsync(int bugid)
{
    using (IDbConnection db = new
        SqlConnection(Configuration.GetConnectionString(BUGTRACKER_DATABASE)))
    {
        db.Open();
        await db.ExecuteNonQuery("delete from bugs Where id=@BugId", new { BugId =
            bugid });
    }
}
```

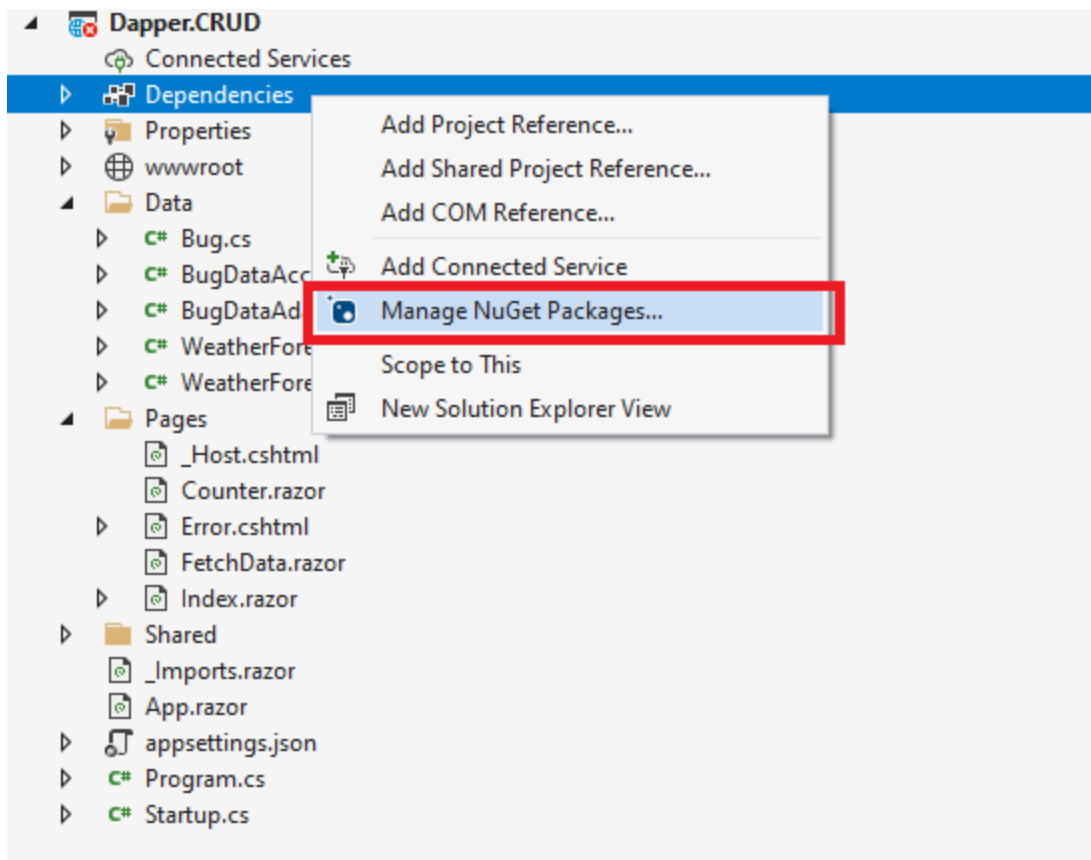
Now, register `BugDataAccessLayer` as scoped service in the `Startup.cs` as follows.



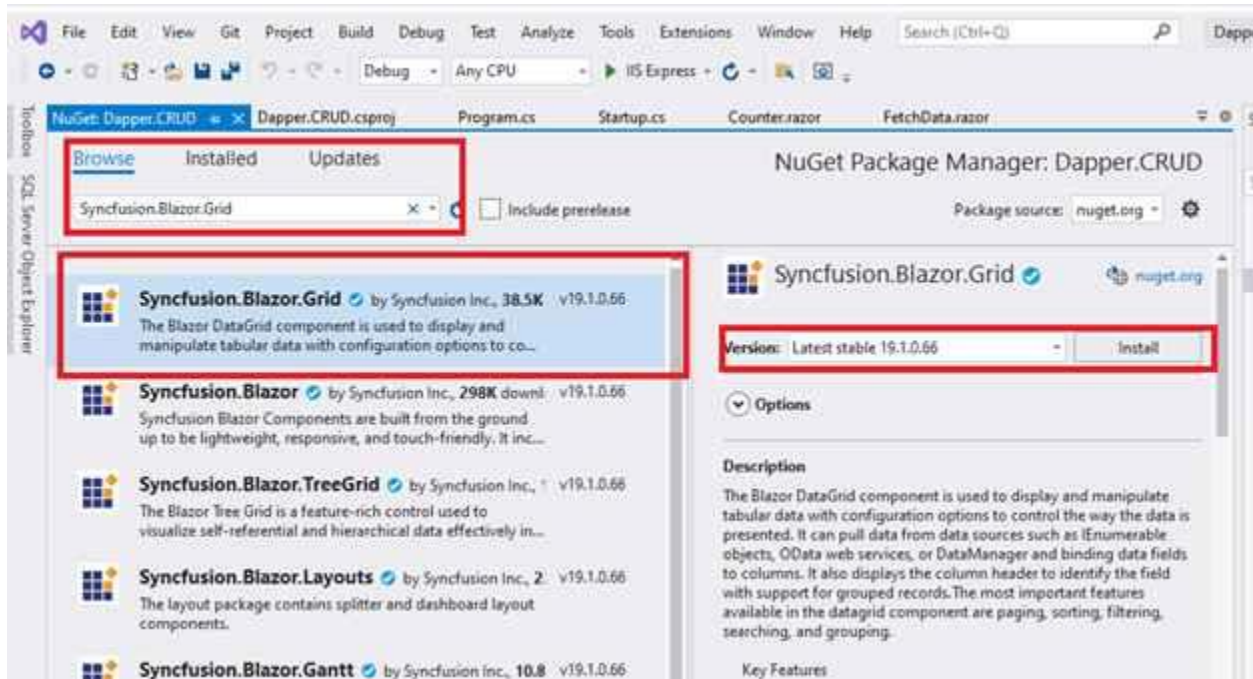
[Adding Syncfusion Blazor components Package](#)

We are going to explain this data binding process (using dapper) using the Syncfusion DataGrid component and perform CRUD operations in it.

So, we are going to install the packages required to use the Syncfusion Blazor components. Now, right-click `Dependencies` in the project and select `Manage NuGet Packages`.



Now, in the **Browse** tab, search and install the **Syncfusion.Blazor.Grid** NuGet package.



For this demo, we have used **Syncfusion.Blazor**(19.1.0.65) NuGet package. We will release a new **Syncfusion.Blazor** NuGet package with new enhancement in our every-week release and main release. So, you can check and update to the [latest versions](#).

Adding Syncfusion Blazor DataGrid component

Open **_Import.razor** file and add the following namespaces which are required to use the Syncfusion Blazor DataGrid Component in this application.

ASPX-CS

```
@using Syncfusion.Blazor
@using Syncfusion.Blazor.Grids
@using Syncfusion.Blazor.Data
```

Open **Startup.cs** file and register the Syncfusion service in the **ConfigureServices** method as follows.

C#

```
using Syncfusion.Blazor;
namespace BlazorDapper
{
    public class Startup
    {
        public void ConfigureServices(IServiceCollection services)
        {
            services.AddRazorPages();
            services.AddServerSideBlazor();
            services.AddSingleton<WeatherForecastService>();
            services.AddScoped<BugDataAccessLayer>();
            services.AddSyncfusionBlazor();
        }
    }
}
```

Syncfusion Blazor provides different themes. They are:

- Bootstrap4
- Material
- Fabric
- Bootstrap
- High Contrast

In this demo application, the Bootstrap4 theme will be used. To add the theme, open the **Pages/_Host.cshtml** file and add the following CSS reference code.

HTML

```
<link href="_content/Syncfusion.Blazor.Themes/
bootstrap4.css" rel="stylesheet" />
```

In previous steps, we have successfully configured the Syncfusion Blazor package in the application. Now, we can add the DataGrid Component to the **Index.razor**.

ASPX-CS

```
<SfGrid>
</SfGrid>
```

Binding data to the DataGrid component

Now, we are going to get SQL data using Dapper and bind it to the DataGrid component. To bind the database table to Syncfusion Blazor DataGrid, we are going to use the [custom data binding feature](#) here.

The following points must be considered for creating a custom adaptor.

- Our custom adaptor must extend the **DataAdaptor** class.
- Override available CRUD methods to handle data querying and manipulation.
- Register our custom adaptor class as a service in the **Startup.cs**.

Now, create a new class named **BugDataAdaptor.cs** under the **Data** folder and replace the following code in that class.

In the following code example,

- Extended **BugDataAdaptor** class with **DataAdaptor** base class.
- Injected **BugDataAccessLayer** instance to perform data operations.

C#

```
public class BugDataAdaptor: DataAdaptor
{
    private BugDataAccessLayer _dataLayer;
    public BugDataAdaptor(BugDataAccessLayer bugDataAccessLayer)
    {
        _dataLayer = bugDataAccessLayer;
    }
    public override async Task<object> ReadAsync(DataManagerRequest
    dataManagerRequest, string key = null)
    {
        List<Bug> bugs = await _dataLayer.GetBugsAsync();
        int count = await _dataLayer.GetBugCountAsync();
        return dataManagerRequest.RequiresCounts ? new DataResult() { Result = bugs,
        Count = count } : count;
    }
}
```

Now, Open the **Startup.cs** file and register the **BugDataAdaptor** class in the **ConfigureServices** method as follows.

C#

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddRazorPages();
    services.AddServerSideBlazor();
    services.AddSingleton<WeatherForecastService>();
    services.AddScoped<BugDataAccessLayer>();
}
```

```
services.AddSyncfusionBlazor();  
services.AddScoped<BugDataAdapter>();  
}
```

Now, we have added **SfDataManager** in Grid for binding the data to the Grid and added column definition.

In the following code example,

- Defined **SfDataManager** component to provide data source to the grid. You can see that we have specified the **AdaptorInstance** property with the type of the custom adaptor we created in the previous step and mentioned the **Adaptor** property as **Adaptors.CustomAdaptor**.
- **TValue** is specified as **Bug** class.

ASPX-CS

```
<SfGrid TValue="Bug">  
<SfDataManager AdaptorInstance="typeof (BugDataAdapter) "  
Adaptor="Adaptors.CustomAdaptor"></SfDataManager>  
</SfGrid>
```

Grid columns can be defined using the [GridColumn](#) component. We are going to create columns using the following code, let us see the properties used and their usage.

ASPX-CS

```
<SfGrid TValue="Bug">  
<SfDataManager AdaptorInstance="typeof (BugDataAdapter) "  
Adaptor="Adaptors.CustomAdaptor"></SfDataManager>  
<GridColumn>  
<GridColumn Field="@nameof (Bug.Id) " IsPrimaryKey="true"  
Visible="false"></GridColumn>  
<GridColumn Field="@nameof (Bug.Summary) " Width="100"></GridColumn>  
<GridColumn Field="@nameof (Bug.BugPriority) " HeaderText="Priority"  
Width="100"></GridColumn>  
<GridColumn Field="@nameof (Bug.Assignee) " Width="100"></GridColumn>  
<GridColumn Field="@nameof (Bug.BugStatus) " HeaderText="Status"  
Width="100"></GridColumn>  
</GridColumn>  
</SfGrid>
```

Now, the DataGrid will look like this while running the application. The displayed records are fetched from the database.

Bug Data Tracking Grid!

+ Add ✎ Edit 🗑 Delete 📅 Update ✕ Cancel			
Summary	Priority	Assignee	Status
Touch is not working	High	Vinet	In-Progress
Drag not working	Normal	Tim	Not-Started
Actions events not triggering wit...	High	Steve	Code-Review
Selection not working properly	Normal	Ram	Not-Started

Handling CRUD operations with our Syncfusion Blazor DataGrid component

We can enable editing in the grid component using the [GridEditSettings](#) component. Grid provides various modes of editing options such as Inline/Normal, Dialog, and Batch editing. Kindly refer to the following documentation for your reference.

Grid Editing

Normal editing is the default edit mode for the DataGrid component. You need to set the `IsPrimaryKey` property of Column as `True` for a particular column, whose value is a unique value for editing purposes.

Here, we are using inline edit mode and the [Toolbar](#) property to show toolbar items for editing.

ASPX-CS

```
<SfGrid TValue="Bug" Toolbar="@ (new List<string>() { "Add", "Edit",
"Delete", "Update", "Cancel" }) ">
<SfDataManager AdaptorInstance="typeof(BugDataAdapter)"
Adaptor="Adaptors.CustomAdaptor"></SfDataManager>
<GridEditSettings AllowAdding="true" AllowEditing="true"
AllowDeleting="true"></GridEditSettings>
<GridColumns>
<GridColumn Field="@nameof(Bug.Id)" IsPrimaryKey="true"
Visible="false"></GridColumn>
<GridColumn Field="@nameof(Bug.Summary)" Width="100"></GridColumn>
<GridColumn Field="@nameof(Bug.BugPriority)" HeaderText="Priority"
Width="100"></GridColumn>
<GridColumn Field="@nameof(Bug.Assignee)" Width="100"></GridColumn>
<GridColumn Field="@nameof(Bug.BugStatus)" HeaderText="Status"
Width="100"></GridColumn>
</GridColumns>
</SfGrid>
```

We have already created CRUD operations method in the data access layer section itself. Now, we are going to call those methods while performing CRUD actions in DataGrid.

Insert a row

Add the following codes(`InsertAsync`) in the `BugDataAdapter`(`CustomAdaptor`) class to perform insert operation.

C#

```
public override async Task<object> InsertAsync(DataManager dataManager,
object data, string key)
```

```
{
    await _dataLayer.AddBugAsync(data as Bug);
    return data;
}
```

To insert a new row, click the **Add** toolbar button. The new record edit form will look like below.

Bug Data Tracking Grid!

+ Add Edit Delete Update Cancel			
Summary	Priority	Assignee	Status
Validation not working	Normal	John	Not-Started
Touch is not working	High	Vinet	In-Progress
Drag not working	Normal	Tim	Not-Started
Actions events not triggering wit...	High	Steve	Code-Review
Selection not working properly	Normal	Ram	Not-Started

Clicking the **Update** toolbar button will call the **InsertAsync** method of our **BugDataAdapter** to insert the record in the **Bug** table. Now, the successfully inserted record in the grid will look like below.

+ Add Edit Delete Update Cancel			
Summary	Priority	Assignee	Status
Touch is not working	High	Vinet	In-Progress
Drag not working	Normal	Tim	Not-Started
Actions events not triggering wit...	High	Steve	Code-Review
Selection not working properly	Normal	Ram	Not-Started
Validation not working	Normal	John	Not-Started

Update a row

Add the following codes (**UpdateAsync**) in the **BugDataAdapter**(CustomAdaptor) class to perform update operation.

C#

```
public override async Task<object> UpdateAsync(DataManager dataManager,
    object data, string keyField, string key)
{
    await _dataLayer.UpdateBugAsync(data as Bug);
    return data;
}
```

To edit a row, select any row and click the **Edit** toolbar button. The edit form will look like below.

+ Add Edit Delete Update Cancel			
Summary	Priority	Assignee	Status
Touch is not working	High	Vinet	In-Progress
<input type="text" value="Drag not working"/>	<input type="text" value="Normal"/>	<input type="text" value="Tim"/>	<input type="text" value="In-Progress"/>
Actions events not triggering wit...	High	Steve	Code-Review
Selection not working properly	Normal	Ram	Not-Started
Validation not working	Normal	John	Not-Started

Here, we are changing the Status field value from Not started to In progress. Clicking the Update toolbar button will call the UpdateAsync method of our BugDataAdapter to update the record in the Bug table. Now, the successfully updated record in the grid will look like below.

+ Add Edit Delete Update Cancel			
Summary	Priority	Assignee	Status
Touch is not working	High	Vinet	In-Progress
Drag not working	Normal	Tim	In-Progress
Actions events not triggering wit...	High	Steve	Code-Review
Selection not working properly	Normal	Ram	Not-Started
Validation not working	Normal	John	Not-Started

Delete a row

Add the following codes(RemoveAsync) in the BugDataAdapter(CustomAdaptor) class to perform update operation.

C#

```
public override async Task<object> RemoveAsync(DataManager dataManager,
object primaryKeyValue, string keyField, string key)
{
    await _dataLayer.RemoveBugAsync(Convert.ToInt32(primaryKeyValue));
    return primaryKeyValue;
}
```

To delete a row, select any row and click the Delete toolbar button. Clicking the Delete toolbar button will call the RemoveAsync method of our BugDataAdapter to update the record in the Bug table.

Please find the sample from this [Github](#) location.

How to bind data to the Syncfusion Blazor components using gRPC service

In this topic, we are going to discuss how to consume data from [gRPC](#) service and bind it to a Syncfusion Blazor Component.

Prerequisite software

The following software are needed,

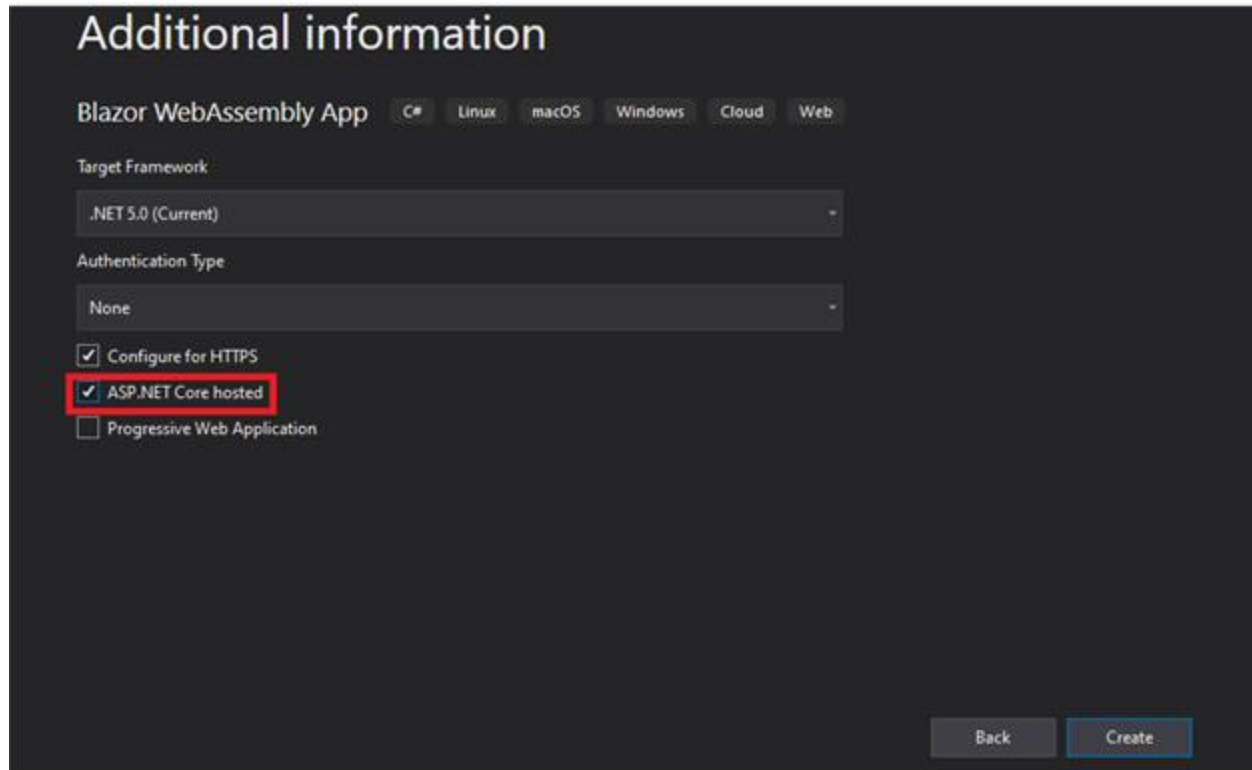
- Visual Studio 2019 v16.9.0 or later.
- .NET SDK 5.0 or later.

Creating Blazor server-side application

Open Visual Studio 2019 and follow the steps in the below documentation to **create the Blazor WebAssembly(Hosted) Application**.

[Create Blazor WebAssembly Application](#)

Finally, ensure to select the **ASP.NET Core Hosted** application.



Adding gRPC dependencies

You need to install the following dependencies on your **Client**, **Server**, and **Shared** projects.

Starting with the **Shared project**, **Right-click** the project and go to **Manage NuGet packages**. Browse for and install the following packages in the **Shared** project:

- [Google.Protobuf](#)
- [Grpc.Net.Client](#)
- [Grpc.Tools](#)

Next, follow the same process and go to your **Server** project to install these packages:

- [Grpc.AspNetCore](#)
- [Grpc.AspNetCore.Web](#)

Then, go to your **Client** project and install the following package:

- [Grpc.Net.Client.Web](#)

Add Proto file and Service to the project

gRPC works with Proto files which are written in protocol buffer language that define your messaging. To learn more about the language, please visit this [guide](#).

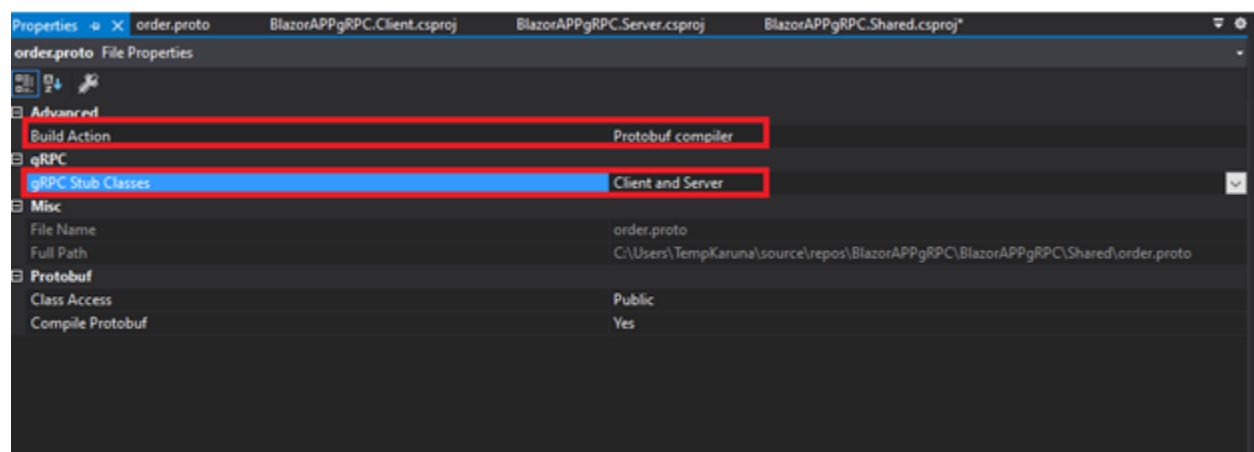
To add a proto file, **Right-click** the **Shared** project, go to **Add**, and then select **New item**. You can choose a Class file and name it as `order.proto` and select **Add**.

Remove the boiler plate code in the file. Copy the following code and paste it into your proto file.

C#

```
syntax = "proto3";
import "google/protobuf/empty.proto";
import "google/protobuf/timestamp.proto";
option csharp_namespace = "BlazorAPPgRPC.Shared";
package Orders;
service OrdersService {
  rpc GetOrders (google.protobuf.Empty) returns (OrdersResponse);
}
message OrdersResponse {
  repeated Orders = 1;
}
message Orders {
  google.protobuf.Timestamp dateTimeStamp = 1;
  int32 OrderID = 2;
  string CustomerName = 3;
  string ShipCountry = 4;
  string ShipCity = 5;
}
```

Go to the proto file properties and select the **Protobuf** compiler as the **Build Action**. Then, select the **Client and Server** option as the **gRPC Stub Classes**. Refer to the following screenshot.



Adding Orders Service

Now, add **Orders** partial class in the **Shared** project. The main properties of this class are generated from the `.proto` file. However, you can also add some extra useful properties to this partial class.

C#

```
using Google.Protobuf.WellKnownTypes;
using System;
```

```
namespace BlazorAPPgRPC.Shared
{
    public partial class Orders
    {
        public DateTime OrderDate
        {
            get => DateTimeStamp.ToDateTime();
            set { DateTimeStamp = Timestamp.FromDateTime(value.ToUniversalTime()); }
        }
    }
}
```

Create **Services** folder in the **Server** project and add **OrdersService** file in that folder. Then, copy the following code and paste it into your service file or create your own service logic.

C#

```
public class OrdersService :
    BlazorAPPgRPC.Shared.OrdersService.OrdersServiceBase
{
    private static readonly string[] Countries = new[]
    {
        "Berlin", "Tokyo", "Denmark", "Tokyo", "Olso"
    };
    private static readonly string[] Names = new[]
    {
        "VINET", "RIO", "RAJ", "MAH", "RAM"
    };
    private static readonly string[] Cities = new[]
    {
        "New York", "London", "Hue"
    };
    public override Task<OrdersResponse> GetOrders(Empty request,
        ServerCallContext context)
    {
        var response = new OrdersResponse();
        response.Orders.AddRange(GetOrders());
        return Task.FromResult<OrdersResponse>(response);
    }
    public IEnumerable<Orders> GetOrders()
    {
        var rng = new Random();
        return Enumerable.Range(1, 365).Select(index => new Orders
        {
            OrderID = index,
            OrderDate = DateTime.Now.AddDays(index),
            ShipCountry = Countries[rng.Next(Countries.Length)],
            CustomerName = Names[rng.Next(Names.Length)],
            ShipCity = Cities[rng.Next(Cities.Length)]
        });
    }
}
```

The **OrdersService** class is inherited from **BlazorAPPgRPC.Shared.OrdersService.OrdersServiceBase**, which is generated automatically from the **.proto** file.

Configure gRPC and gRPC-Web in the Server

You need to register the **gRPC service** in your **Startup.cs** file. This enables you to use dependency injection to consume the service across the app. Add the following code to your **ConfigureServices** method in the **Server Startup.cs** file:

C#

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddScoped<OrdersService>();
    services.AddGrpc();
    . . .
}
```

Then, add the gRPC-Web middleware to the apps configuration and register the gRPC service. This must be added after **UseRouting** and before **UseEndpoints** in the **Configure** method.

C#

```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    . . .
    app.UseRouting();
    app.UseGrpcWeb(new GrpcWebOptions { DefaultEnabled = true });
    app.UseEndpoints(endpoints =>
    {
        endpoints.MapGrpcService<OrdersService>();
        . . .
    });
}
```

In the **Client** project, add the **OrdersService** to the container, then create a gRPC-Web channel pointing to the back-end server and instantiate the gRPC clients for this channel. Refer to the following code to modify the **Program.cs** file.

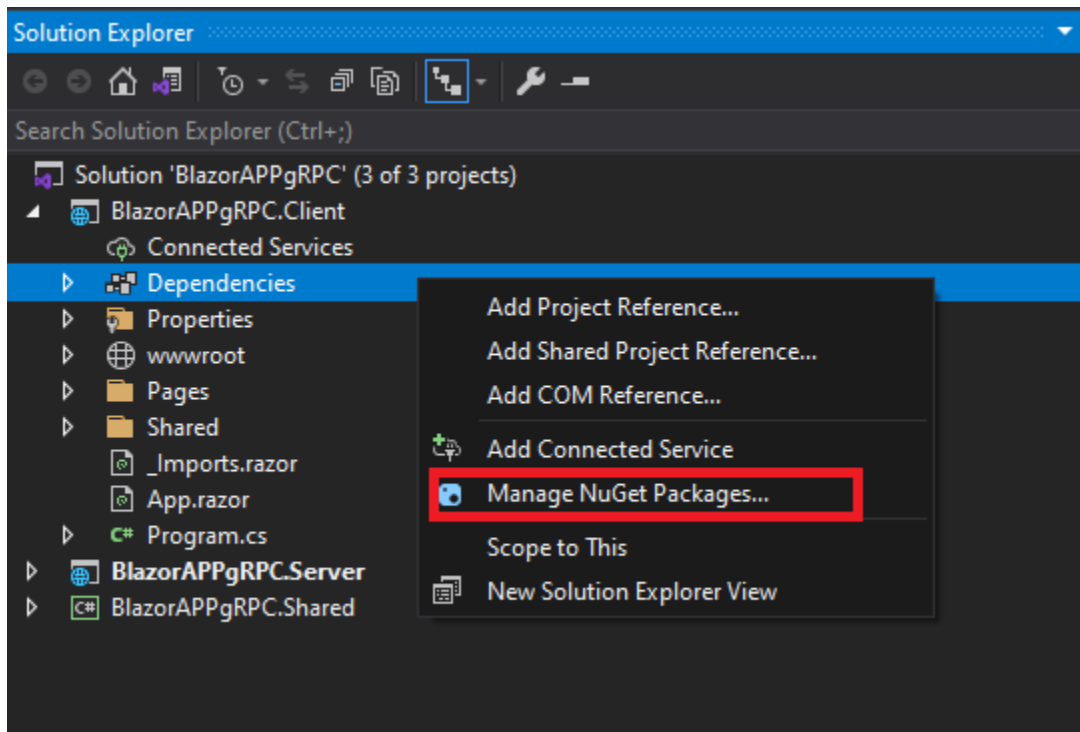
C#

```
using Grpc.Net.Client;
using Grpc.Net.Client.Web;
using Microsoft.AspNetCore.Components;
. . .
public static async Task Main(string[] args)
{
    var builder = WebAssemblyHostBuilder.CreateDefault(args);
    . . .
    builder.Services.AddSingleton(services =>
    {
        var httpClient = new HttpClient(new GrpcWebHandler(GrpcWebMode.GrpcWeb, new
        HttpClientHandler()));
        var backendUrl = services.GetRequiredService<NavigationManager>().BaseUri;
        var channel = GrpcChannel.ForAddress(backendUrl, new GrpcChannelOptions {
        HttpClient = httpClient });
        return new OrdersService.OrdersServiceClient(channel);
    });
}
```

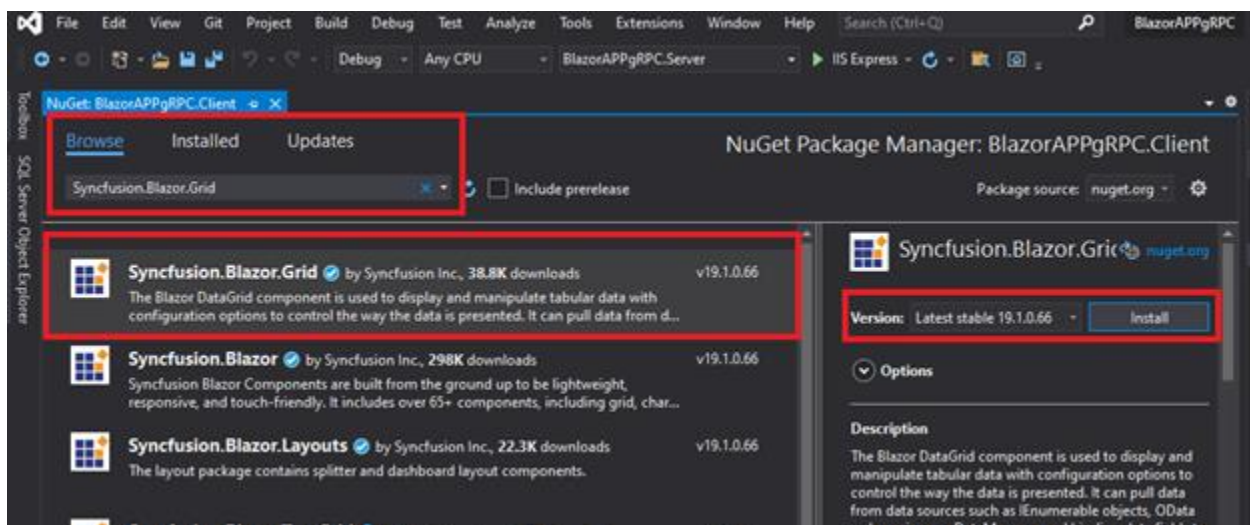
The **OrdersService.OrdersServiceClient** class is also generated automatically from the **.proto** file.

Add Syncfusion Blazor DataGrid package

We are going to explain this data (gRPC service data) binding process using the Syncfusion DataGrid component. So, we are going to install the packages required to use the Syncfusion Blazor components. Now, right-click **Dependencies** in the project and select **Manage NuGet Packages**.



In the **Browse** tab, search and install the **Syncfusion.Blazor.Grid** NuGet package.



For this demo, we have used **Syncfusion.Blazor(19.1.0.66)** NuGet package. We will release a new **Syncfusion.Blazor** NuGet package with new enhancement in our every-week release and main release. So, you can check and update to the [latest versions](#).

Adding Syncfusion Blazor DataGrid component

Open `_Import.razor` file and add the following namespaces which are required to use Syncfusion Blazor DataGrid Component in this application.

ASPX-CS

```
@using Syncfusion.Blazor
@using Syncfusion.Blazor.Grids
```

Open `Program.cs` file in a **Client** project and **register** the **Syncfusion service** in the **ConfigureServices** method as follows.

C#

```
using Syncfusion.Blazor;
namespace BlazorAPPGRPC.Client
{
    public class Program
    {
        public static async Task Main(string[] args)
        {
            var builder = WebAssemblyHostBuilder.CreateDefault(args);
            builder.RootComponents.Add<App>("#app");
            builder.Services.AddSyncfusionBlazor();
            . . .
        }
    }
}
```

In this demo application, the **Bootstrap4** theme will be used. To add the theme, open `~/wwwroot/index.html` file and add the following CSS reference code.

HTML

```
<link href="_content/Syncfusion.Blazor.Themes/
bootstrap4.css" rel="stylesheet" />
```

In previous steps, we have successfully configured the Syncfusion Blazor package in the application. Now, we can add the DataGrid Component to the `Index.razor`.

ASPX-CS

```
<SfGrid>
</SfGrid>
```

Bind data to Blazor DataGrid component

To consume data from the gRPC service, inject the **OrdersServiceClient** into the razor page and assign it to the DataGrid's `DataSource` property.

Here, we have used the `DataSource` property of the DataGrid component to bind the data to DataGrid in the WebAssembly application.

Grid columns can be defined using the [GridColumnn](#) component. Columns are created using the following code, let's see the properties used and their usage.

- **Field** property specifies the column name of the **Orders** table to display in the grid column.
- The **Width** property specifies the column width.
- **Format** property helps to format number, currencies, and date in a particular culture. Here, we have formatted the **OrderDate** column.
- **HeaderText** property specifies the column header name.

ASPX-CS

```
@inject OrdersService.OrdersServiceClient OrdersServiceClient
@using BlazorAPPGRPC.Shared
@using Google.Protobuf.WellKnownTypes
<h1>Orders</h1>
<p>This component demonstrates fetching data from the gRPC service.</p>
@if (orders == null)
{
    <p><em>Loading...</em></p>
}
else
{
    <SfGrid DataSource="@orders" AllowPaging="true">
        <GridColumn>
            <GridColumn Field=@nameof(Orders.OrderID) HeaderText="Order ID"
            TextAlign="TextAlign.Right" Width="120"></GridColumn>
            <GridColumn Field=@nameof(Orders.CustomerName) HeaderText="Customer Name"
            Width="150"></GridColumn>
            <GridColumn Field=@nameof(Orders.OrderDate) HeaderText=" Order Date"
            Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
            Width="130"></GridColumn>
            <GridColumn Field=@nameof(Orders.ShipCountry) HeaderText="Ship Country"
            TextAlign="TextAlign.Right" Width="120"></GridColumn>
            <GridColumn Field=@nameof(Orders.ShipCity) HeaderText="Ship City"
            Width="150"></GridColumn>
        </GridColumn>
    </SfGrid>
}
@code {
    private IList<Orders> orders;
    protected override async Task OnInitializedAsync()
    {
        orders = (await OrdersServiceClient.GetOrdersAsync(new Empty())).Orders;
    }
}
```

Now, the DataGrid will look like this while running the application. The displayed records are fetched from the gRPC service.

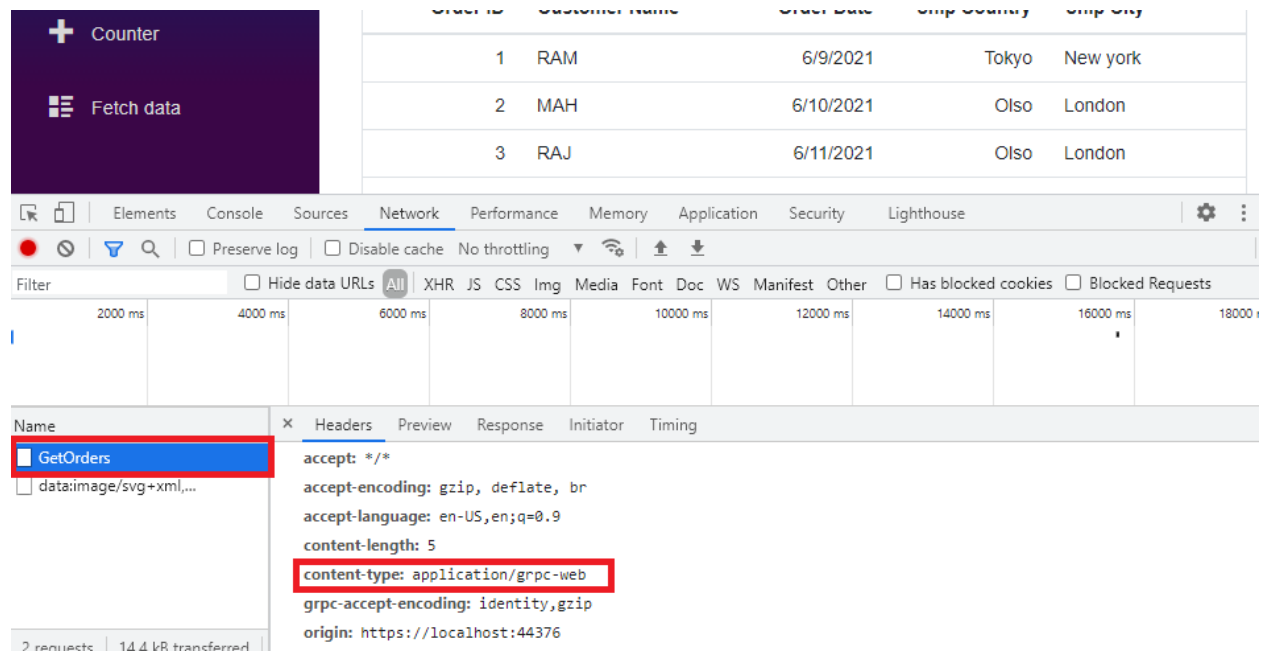
This component demonstrates fetching data from the gRPC service.

Order ID	Customer Name	Order Date	Ship Country	Ship City
1	VINET	6/9/2021	Tokyo	London
2	VINET	6/10/2021	Tokyo	New york
3	RAJ	6/11/2021	Berlin	London
4	MAH	6/12/2021	Berlin	London
5	RAM	6/13/2021	Olso	London
6	MAH	6/14/2021	Berlin	London
7	RIO	6/15/2021	Tokyo	London
8	RIO	6/16/2021	Denmark	New york
9	RAM	6/17/2021	Denmark	New york
10	VINET	6/18/2021	Denmark	New york
11	RAJ	6/19/2021	Olso	New york
12	MAH	6/20/2021	Tokyo	New york

« < 1 2 3 4 5 6 7 8 ... > »
1 of 31 pages (365 items)

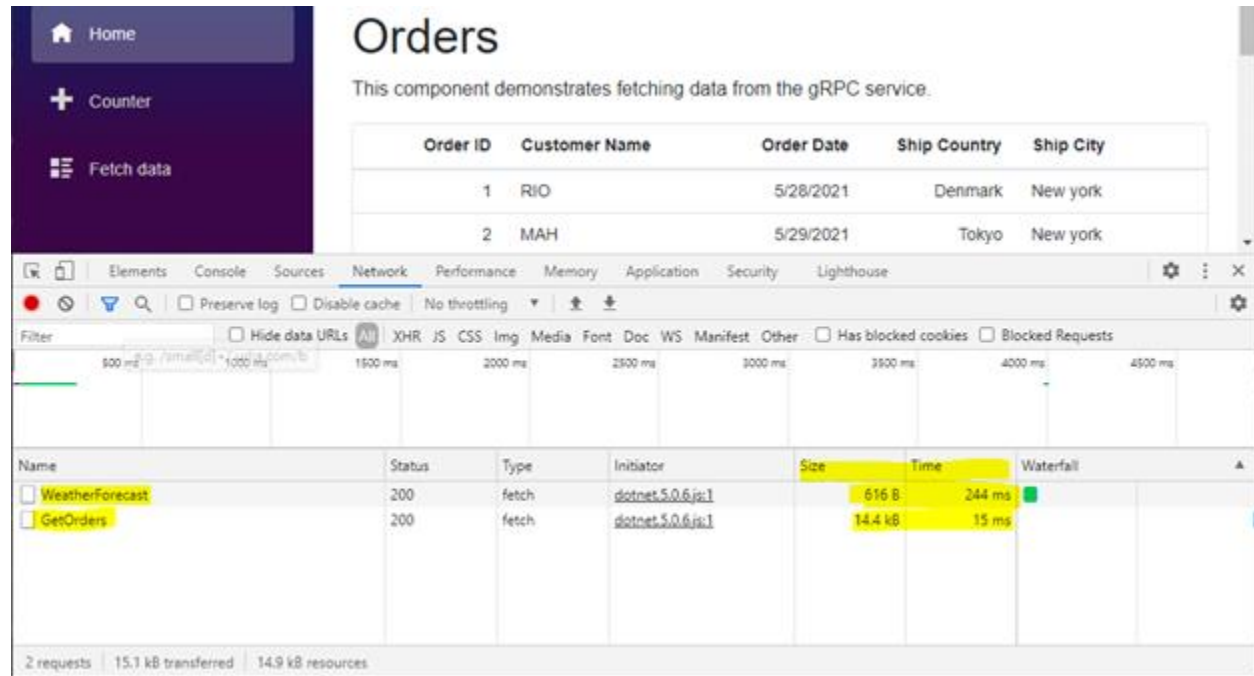
After rendering the Grid with data, you can verify that we are using the gRPC-Web service by using the following way. Open **Network** Tab and reload the index page.

The **Orders** data will be loaded, and you will see the name of the method **GetOrders**. Click on the **GetOrders** name to view the Response Headers which show the **content-type** being **application/grpc-web**.



In the Index.razor page, we have rendered **Grid with gRPC service**, and in the FetchData.razor page, we have rendered **Grid with normal REST service**.

You can see both page **payload size** and **traffic time** in the following screenshot. In the screenshot above, you can see that the **REST service** has sent **616 B**, but the **gRPC service** has sent only **14.4 kB**. Also **traffic time for REST is 244ms** and the traffic time for **gRPC is 15ms** only.



How To

Extend, Customize, and Reuse Components

The Blazor framework provides the support to extend a component or customize it within another component for a strong composite model.

Extend Syncfusion Blazor Component

The Syncfusion Blazor components can extend and customize the logic by creating a new Blazor component.

1. Right-click on the `~/Pages` folder in the Visual Studio and select `Add -> Razor Component` to create a new Razor component (SyncButton.razor).
2. Inherit any Syncfusion Blazor component and render your component based on your logic with Syncfusion Blazor API.

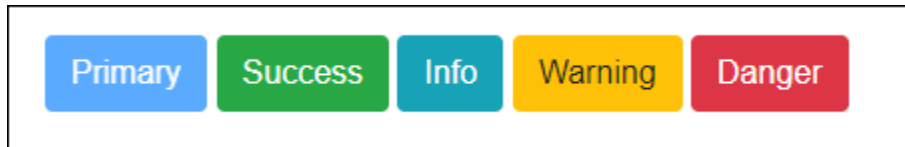
ASPX-CS

```
@inherits SfButton;
<button class="@className" disabled="@Disabled">@Content</button>
@code {
private string className = "btn";
[Parameter]
public ButtonStyles Styles { get; set; }
public enum ButtonStyles
{
    Basic,
    Success,
    Info,
    Warning,
    Danger
}
protected override void OnInitialized()
{
    base.OnInitialized();
    ApplyStyles();
}
private void ApplyStyles()
{
    if (IsPrimary)
    {
        className += " btn-primary";
    }
    else if (Styles == ButtonStyles.Success)
    {
        className += " btn-success";
    }
    else if (Styles == ButtonStyles.Info)
    {
        className += " btn-info";
    }
    else if (Styles == ButtonStyles.Warning)
    {
        className += " btn-warning";
    }
    else
    {
        className += " btn-danger";
    }
}
```

3. Render your new component in the view page `~/Pages/Index.razor` and run the application.

ASPX-CS

```
<SyncButton Content="Primary" IsPrimary="true" Disabled="true"></SyncButton>
<SyncButton Content="Success"
Styles="SyncButton.ButtonStyles.Success"></SyncButton>
<SyncButton Content="Info"
Styles="SyncButton.ButtonStyles.Info"></SyncButton>
<SyncButton Content="Warning"
Styles="SyncButton.ButtonStyles.Warning"></SyncButton>
<SyncButton Content="Danger"
Styles="SyncButton.ButtonStyles.Danger"></SyncButton>
```



Use Syncfusion Blazor Component within Another Blazor Component

The Syncfusion Blazor component can be implemented within another Blazor component.

1. Right-click on the `~/Pages` folder in the Visual Studio and select `Add -> Razor Component` to create a new Razor component (TodoList.razor).
2. Add any Syncfusion Blazor component to the newly created Blazor component.

ASPX-CS

```
<h3>Todo List</h3>
@using Syncfusion.Blazor.Inputs;
@using Syncfusion.Blazor.Buttons;
@using Syncfusion.Blazor.Lists;
<div class="form-group">
<SfTextBox @oninput="UpdateItem" @bind-Value="@item" Placeholder="Add new
item" Width="200px"></SfTextBox>
<SfButton @onclick="AddItem">Add</SfButton>
</div>
@if (items.Count > 0)
{
<SfListView DataSource="@items">
<ListViewFieldSettings TValue="ItemModel" Id="Id" Text="Text"></
ListViewFieldSettings>
</SfListView>
}
@code {
private string item;
private List<ItemModel> items = new List<ItemModel>();
private void UpdateItem(Microsoft.AspNetCore.Components.ChangeEventArgs
args)
{
item = args.Value.ToString();
}
// Add new items on button click.
private void AddItem()
```

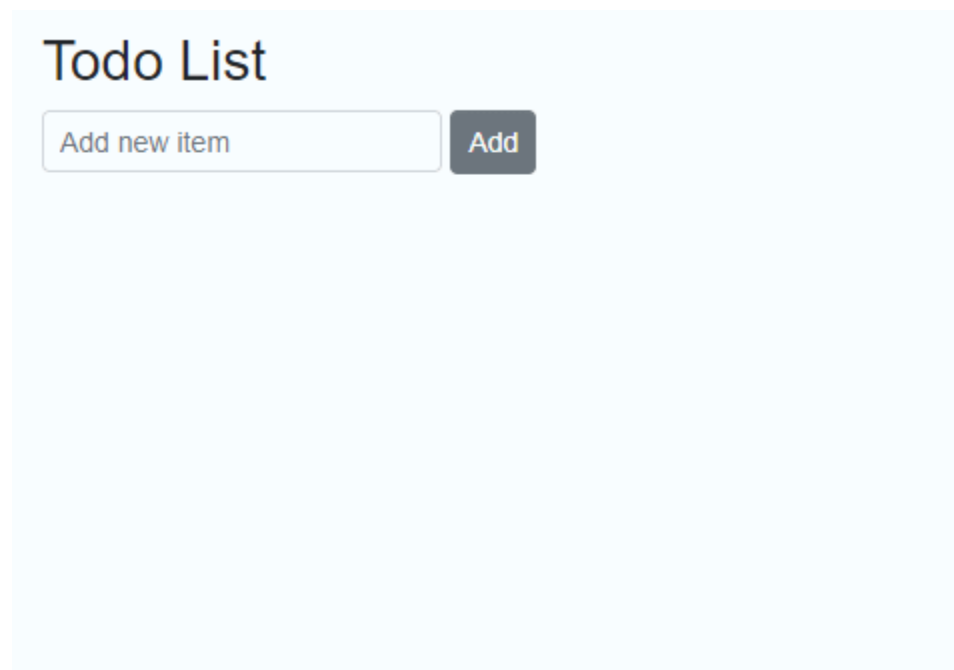


```
{
if (item != null && item.Length > 0)
{
var newItem = new ItemModel(items.Count + 1, item);
items.Add(newItem);
item = null;
}
}
// List view data source model.
public class ItemModel
{
public ItemModel(int id, string text)
{
Id = id;
Text = text;
}
public int Id { get; set; }
public string Text { get; set; }
}
}
```

3. Render your new component in the view page `~/Pages/Index.razor` and run the application.

ASPX-CS

```
<ToDoList></ToDoList>
```



Render Syncfusion Blazor Component Dynamically

The following methods can be used to render the Syncfusion Blazor components dynamically:

1. [RenderFragment](#) with [Razor Template Syntax](#)

2. [RenderFragment](#) with [RenderTreeBuilder](#) methods
3. [BuildRenderTree](#) method

[RenderFragment with Razor Syntax](#)

The **RenderFragment** represents the segments of UI content that can be reused in the view page based on the application logic.

You can use [Razor Template Syntax](#) to define the **RenderFragment** in the view page.

The following code demonstrates the **RenderFragment** that renders the **SfButton** component.

ASPX-CS

```
@{
    RenderFragment<string> ButtonFragment = content =>
    @<SfButton>@content</SfButton>;
}
@ButtonFragment("My Button")
```

The following code demonstrates the **RenderFragment** that renders the **SfGrid** component.

ASPX-CS

```
@{
    RenderFragment<List<Order>> GridFragment = dataModel =>
    @<SfGrid DataSource="@dataModel"></SfGrid>;
}
@GridFragment(orders)
@code {
    private List<Order> orders { get; set; }
    public class Order
    {
        public int? OrderID { get; set; }
        public string CustomerID { get; set; }
    }
    protected override void OnInitialized()
    {
        orders = Enumerable.Range(1, 5).Select(x => new Order()
        {
            OrderID = 1000 + x,
            CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
        }) [new Random().Next(5)]
        }).ToList();
    }
}
```

[RenderFragment with RenderTreeBuilder Methods](#)

You can define the **RenderFragment** delegate with [RenderTreeBuilder](#)'s methods.

Refer to [Manual RenderTreeBuilder logic](#) to know more about **RenderTreeBuilder**.

ASPX-CS

```
@RenderSfButton()
@code {
    private RenderFragment RenderSfButton()
```

```
{
RenderFragment button = b =>
{
b.OpenComponent<SfButton>(0);
b.AddAttribute(1, "Content", "My Button");
b.CloseComponent();
};
return button;
}
}
```

You can render the nested components by using `RenderFragment`.

ASPX-CS

```
<SfChip>
<ChipItems>
@RenderChipItem("Apple")
@RenderChipItem("Banana")
@RenderChipItem("Mango")
</ChipItems>
</SfChip>
@code {
private RenderFragment RenderChipItem(string text)
{
RenderFragment chipItem = b =>
{
b.OpenComponent<ChipItem>(0);
b.AddAttribute(1, "Text", text);
b.CloseComponent();
};
return chipItem;
}
}
```

BuildRenderTree Method

The `BuildRenderTree` is an override method that can be used to render a reusable component dynamically.

Suppose, if you want to add multiple text-boxes in your application with the same properties and values, you may feel like the code is repeatedly used. For example, the below form component has similar textbox settings and is wrapped with a bootstrap `form-group` class.

ASPX-CS

```
<div class="form-group">
<label for="first-name">First Name:</label>
<SfTextBox ID="first-name" CssClass="e-small"
Autocomplete="AutoComplete.Off" ShowClearButton="true"
Width="250px"></SfTextBox>
</div>
<div class="form-group">
<label for="last-name">Last Name:</label>
<SfTextBox ID="last-name" CssClass="e-small" Autocomplete="AutoComplete.Off"
ShowClearButton="true" Width="250px"></SfTextBox>
```

```

</div>
<div class="form-group">
<label for="address">Address:</label>
<SfTextBox ID="address" CssClass="e-small" Autocomplete="AutoComplete.Off"
ShowClearButton="true" Width="250px"></SfTextBox>
</div>
<div class="form-group">
<label for="age">Age:</label>
<SfTextBox ID="age" CssClass="e-small" Autocomplete="AutoComplete.Off"
ShowClearButton="true" Width="250px"></SfTextBox>
</div>
<div class="form-group">
<label for="city">City:</label>
<SfTextBox ID="city" CssClass="e-small" Autocomplete="AutoComplete.Off"
ShowClearButton="true" Width="250px"></SfTextBox>
</div>
<div class="form-group">
<label for="country">Country:</label>
<SfTextBox ID="country" CssClass="e-small" Autocomplete="AutoComplete.Off"
ShowClearButton="true" Width="250px"></SfTextBox>
</div>

```

You can create a simple Blazor component with the `BuildRenderTree` method and get the repeated properties in a `Dictionary` field and reuse it.

1. Right-click on the `~/Pages` folder in the Visual Studio and select `Add -> Class` to create a new class file (`SyncTextBox.cs`).
2. Inherit the newly created class with `ComponentBase` and override the `BuildRenderTree` method to create the component.

C#

```

using Microsoft.AspNetCore.Components;
using Microsoft.AspNetCore.Components.Rendering;
using Syncfusion.Blazor.Inputs;
using System.Collections.Generic;
namespace InputValidationApp.Pages
{
    public class SyncTextBox : ComponentBase
    {
        [Parameter]
        public string ID { get; set; }
        [Parameter]
        public string Label { get; set; }
        [Parameter]
        public Dictionary<string, object> TextAttributes { get; set; }
        protected override void BuildRenderTree(RenderTreeBuilder builder)
        {
            // create div element.
            builder.OpenElement(0, "div");
            builder.AddAttribute(1, "class", "form-group");
            // creating label element.
            builder.OpenElement(2, "label");
            builder.AddAttribute(3, "for", ID);

```

```

builder.AddContent(4, Label);
builder.CloseElement();
// create Syncfusion TextBox component.
builder.OpenComponent<SfTextBox>(5);
builder.AddAttribute(6, "ID", ID);
// Added similar attributes used in the component.
if (TextAttributes != null)
{
    builder.AddMultipleAttributes(3, TextAttributes);
}
builder.CloseComponent();
builder.CloseElement();
}
}
}

```

3. Now, render the new reusable Blazor component in the `~/Pages/Index.razor` page and run the application.

ASPX-CS

```

<SyncTextBox ID="first-name" Label="First Name:"
TextAttributes="@textAttributes"></SyncTextBox>
<SyncTextBox ID="last-name" Label="Last Name:"
TextAttributes="@textAttributes"></SyncTextBox>
<SyncTextBox ID="address" Label="Address:"
TextAttributes="@textAttributes"></SyncTextBox>
<SyncTextBox ID="age" Label="Age:"
TextAttributes="@textAttributes"></SyncTextBox>
<SyncTextBox ID="city" Label="City:"
TextAttributes="@textAttributes"></SyncTextBox>
<SyncTextBox ID="country" Label="Country:"
TextAttributes="@textAttributes"></SyncTextBox>
@code {
    public Dictionary<string, object> textAttributes { get; set; } =
        new Dictionary<string, object>() {
            { "CssClass", "e-small" },
            { "Autocomplete", AutoComplete.Off },
            { "Width", "250px" },
            { "ShowClearButton", true }
        };
}

```

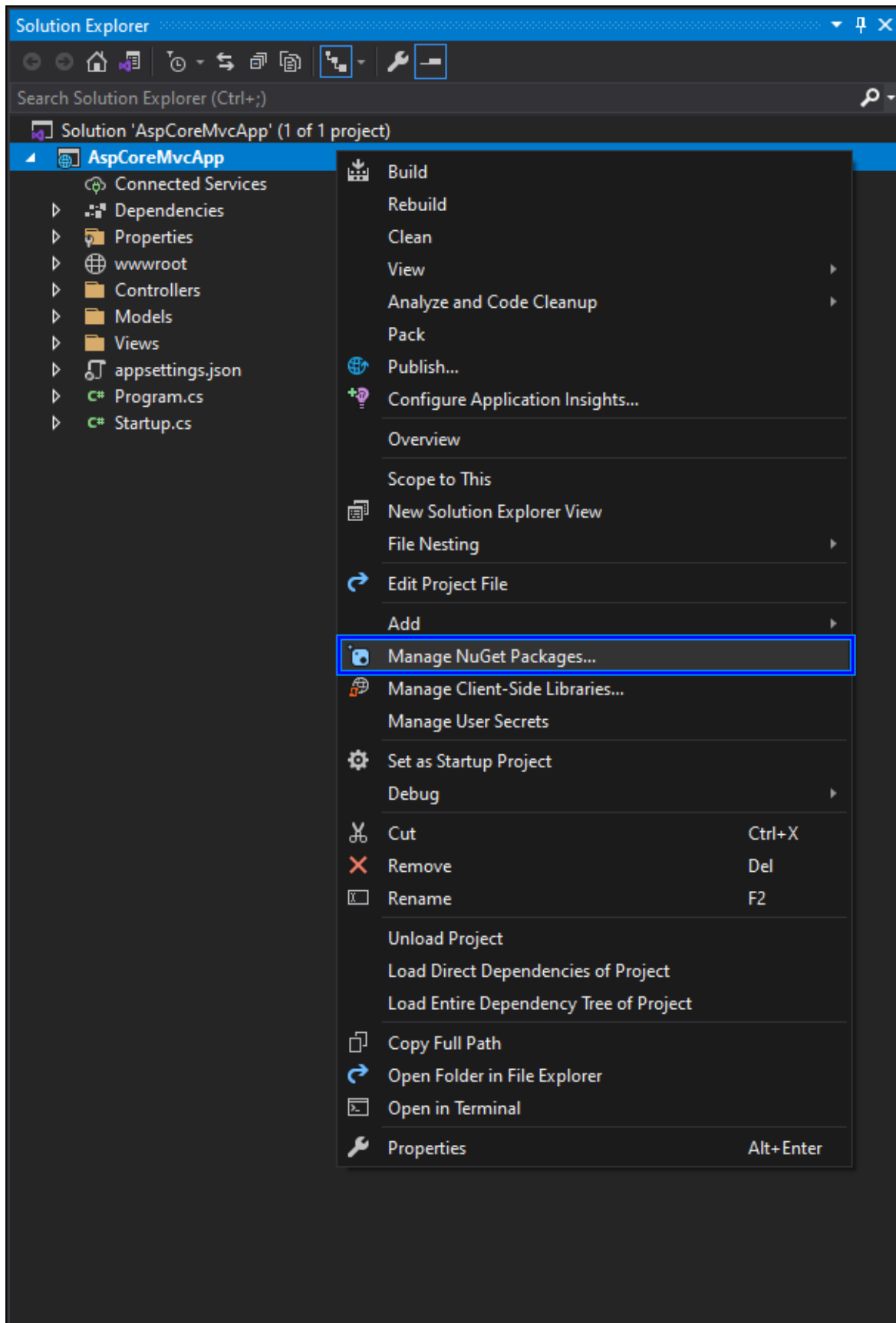
The advantages of a reusable component are:

1. Instead of changing values in each component, you can change a property value once and it will be reflected in all the components.
 2. Better code optimization against the repeated code.
-

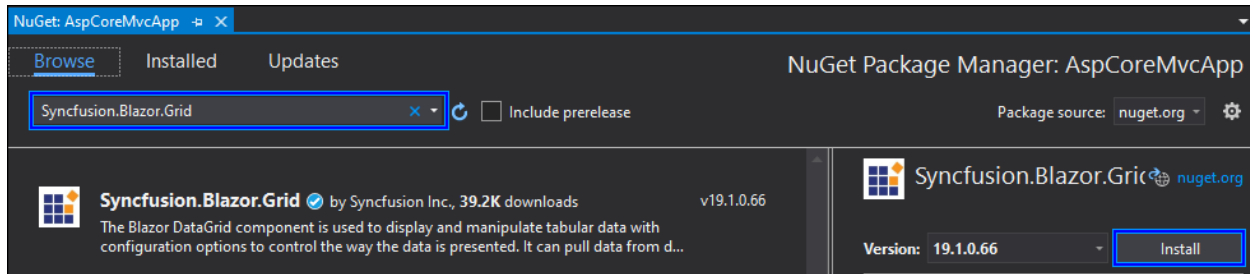
How to Add Syncfusion Blazor Component on an Existing ASP.NET Core MVC Application

This section explains how to add Syncfusion Blazor component on an existing ASP.NET Core MVC application.

1. Open your existing ASP.NET Core MVC application on Visual Studio 2019.
2. Right-click on the project and select **Manage NuGet Package**.



3. Search `Syncfusion.Blazor.Grid` and install the NuGet package.



4. Register Blazor server service and Syncfusion Blazor service in the `ConfigureServices` method on `~/Startup.cs` file.

C#

```
using Syncfusion.Blazor;
public void ConfigureServices(IServiceCollection services)
{
    ....
    ....
    services.AddServerSideBlazor();
    services.AddSyncfusionBlazor();
}
```

5. Add BlazorHub endpoint in the `Configure` method on `~/Startup.cs` file.

C#

```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    app.UseEndpoints(endpoints =>
    {
        ....
        ....
        // Map Blazor SignalR hub.
        endpoints.MapBlazorHub();
    });
}
```

6. Create `~/_Imports.razor` file in the root of your application and add the below namespaces.

ASPX-CS

```
@using System.Net.Http
@using Microsoft.AspNetCore.Authorization
@using Microsoft.AspNetCore.Components.Authorization
@using Microsoft.AspNetCore.Components.Forms
@using Microsoft.AspNetCore.Components.Routing
@using Microsoft.AspNetCore.Components.Web
@using Microsoft.AspNetCore.Components.Web.Virtualization
```



```
@using Microsoft.JSInterop
@using AspCoreMvcApp;
@using Syncfusion.Blazor
@using Syncfusion.Blazor.Grids
```

7. Add Blazor script references at the end of `tag` and Syncfusion theme reference inside the `tag` on `~/Views/Shared/_Layout.cshtml` file.

ASPX-CS

```
<head>
....
....
<link href="_content/Syncfusion.Blazor.Themes/bootstrap4.css"
rel="stylesheet" />
</head>
<body>
....
....
<script src="_framework/blazor.server.js"></script>
</body>
```

8. Create a new folder `~/Components` at the root of application. Right-click on the `~/Components` folder and add a new razor component by Add -> Razor Component.
9. Add the Syncfusion Blazor component in the created razor file.

ASPX-CS

```
<SfGrid DataSource="@Orders" AllowPaging="true">
<GridPageSettings PageSize="5"></GridPageSettings>
<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText="Order Date"
Format="d" Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) Format="C2"
Width="120"></GridColumn>
</GridColumns>
</SfGrid>
@code{
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 50).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
OrderDate = DateTime.Now.AddDays(-x),
```

```

    }).ToList();
}
public class Order
{
    public int? OrderID { get; set; }
    public string CustomerID { get; set; }
    public DateTime? OrderDate { get; set; }
    public double? Freight { get; set; }
}
}

```

10. Now, add the razor component in the `~/Views/Home/Index.cshtml` page using `component` tag helper. The `.razor` file name will be consider as a Razor component. For example, the above `SfGrid` component is added on `~/Components/MyGrid.razor` file.

ASPX-CS

```

@using AspCoreMvcApp.Components;
<component type="typeof(MyGrid)" render-mode="ServerPrerendered" />

```

11. Run the application by pressing `F5` key. Now, the Syncfusion Blazor Grid component will be rendered in the ASP.NET Core MVC application.

AspCoreMvcApp
Home
Privacy

Welcome

Learn about [building Web apps with ASP.NET Core](#).

Order ID	Customer Name	Order Date	Freight
1001	BOLID	6/2/2021	\$2.10
1002	ALFKI	6/1/2021	\$4.20
1003	ANANTR	5/31/2021	\$6.30
1004	BOLID	5/30/2021	\$8.40
1005	ANANTR	5/29/2021	\$10.50

«
<
1
2
3
4
5
6
7
8
...
>
»

1 of 15 pages (75 items)

[See Also](#)

- [Component Tag Helper in ASP.NET Core](#)
- [Integrating Blazor Components on Existing ASP.NET Core MVC apps](#)

How to Render the Blazor Server Application in IE11 Web Browser

This section explains how to render the Blazor Server application in IE11 web browser.

Blazor Client/WebAssembly App

The Microsoft Internet Explorer doesn't support with **WebAssembly**. So, Blazor Client/WebAssembly application does not support with Internet Explorer web browser.

Blazor Server App

The Microsoft Internet Explorer supports **Blazor Server** app with additional polyfills in .NET Core 3.1 apps. However, it is not support with .NET 5.0 or later versions. Refer to [Blazor updated browser support](#) for more information.

Find the following steps to add the polyfills in the Blazor server application.

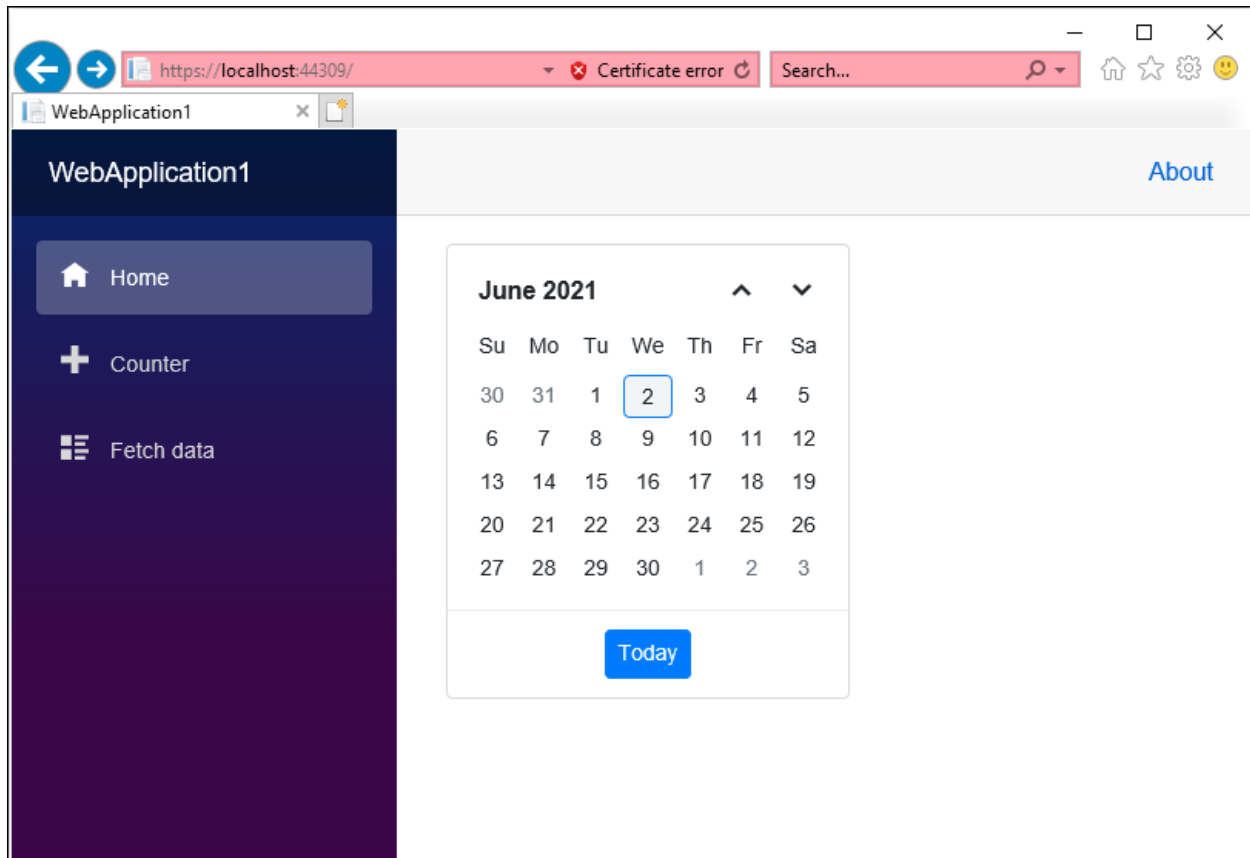
1. Create a Blazor server application using [Blazor Server Getting Started](#) documentation.
2. Add the polyfill script reference in the `~/Pages/_Host.cshtml` page.

HTML

```
<head>
.....
.....
<script
src="https://github.com/Daddoon/Blazor.Polyfill/releases/download/3.0.8/blaz
or.polyfill.min.js"></script>
</head>
```

This [polyfill](#) is required to configure in Blazor server application for IE 11 support for .NET Core 3.1 app.

3. Run the application in the IE 11 web browser and the Syncfusion Blazor Component is now rendered on IE 11.



See Also

- [Blazor Updated Browser Support](#)
- [Blazor Browser Support on .NET 5.0](#)

How to upgrade Syncfusion Blazor Components to the latest version

To upgrade Syncfusion Blazor Components to the latest version, you need to ensure the following:

Compatible .NET version

Syncfusion Blazor components in the latest version '{:nuget-version:}' are compatible with the latest version of .NET Core 5.0 and .NET Core 3.1. Also, refer to [version compatibility](#) documentation for more information about version compatibility of Syncfusion Blazor components and .NET Core SDK.

Client resource file references

Ensure your CSS files have been properly configured in your application.

- Add the following style file reference in the `~/Pages/_Host.cshtml` for Blazor Server app or add it in the `~/wwwroot/index.html` for Blazor WebAssembly app.

If you are using `Syncfusion.Blazor` NuGet package in your application, then use the below reference link.

HTML

```
<link href="_content/Syncfusion.Blazor/styles/bootstrap4.css"
rel="stylesheet" />
```

If you are using [individual NuGet packages](#) in your application, then use the below reference link.

HTML

```
<link href="_content/Syncfusion.Blazor.Themes/bootstrap4.css"
rel="stylesheet" />
```

For production purpose and minimal requirement, Syncfusion provides an option to generate custom styles of selective components by using the ThemeStudio web application. Refer to this [link](#) for more details on ThemeStudio.

Breaking changes

Some changes have been modified in our Blazor samples for each release. So, we suggest you to ensure the breaking changes. Refer to this [release notes](#) for our Blazor components.

Cache problem

Before restoring the NuGet packages, clean the older versions of Syncfusion Blazor NuGet packages.

The following steps explain how to clean the cache:

1. Delete the Syncfusion Blazor NuGet packages from the installed location {System-driver}\Users\{user-name}\.nuget\packages\}. In Windows, the installed location of Syncfusion Blazor NuGet packages can be found using %userprofile%\nuget\packages\}.
2. Update the latest version of Syncfusion Blazor NuGet packages.

Linker.xml configuration

In Blazor WebAssembly application, ensure if you have configured **Linker.xml** file in your Syncfusion Blazor application. Missing this configuration may prevent the rendering of Syncfusion Blazor components in the application.

Refer to this [KB article](#) for more information on Linker.xml file usage.

Linker.xml configuration is applicable only for the Blazor WebAssembly application.

See Also

- [Version Compatibility](#)

How to upgrade Syncfusion Blazor Components to 18.1.0.36 version

To upgrade Syncfusion Blazor Components to 18.1.0.36 version, you need to check the following:

NuGet changes

Previous versions of the Syncfusion Blazor NuGet package name is **Syncfusion.EJ2.Blazor**. From version 18.1.0.36, it will be **Syncfusion.Blazor**. Other Blazor packages will also have the same name changes.

For example, refer to the following table for changes in the Blazor packages name.

Before 18.1.0.36 version	From 18.1.0.36 version
-----	-----

| Syncfusion.EJ2.Blazor | Syncfusion.Blazor |

| Syncfusion.EJ2.WordEditor.Blazor | Syncfusion.WordEditor.Blazor |

| Syncfusion.EJ2.Blazor.PdfViewerServer.Windows | Syncfusion.Blazor.PdfViewerServer.Windows |

| Syncfusion.EJ2.Blazor.PdfViewerServer.Linux | Syncfusion.Blazor.PdfViewerServer.Linux |

| Syncfusion.EJ2.Blazor.PdfViewerServer.OSX | Syncfusion.Blazor.PdfViewerServer.OSX |

Namespace changes

The previous version of Syncfusion Blazor contains the namespace `Syncfusion.EJ2.Blazor` followed by component package names such as Buttons, Charts, Grids, and Inputs. After 18.1.0.36 version, this has been modified to `Syncfusion.Blazor` followed by its package name. Check the following table for examples.

| Before 18.1.0.36 version | From 18.1.0.36 version |

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

| Syncfusion.EJ2.Blazor | Syncfusion.Blazor |

| Syncfusion.EJ2.Blazor.Buttons | Syncfusion.Blazor.Buttons |

| Syncfusion.EJ2.Blazor.Calendars | Syncfusion.Blazor.Calendars |

| Syncfusion.EJ2.Blazor.Charts | Syncfusion.Blazor.Charts |

| Syncfusion.EJ2.Blazor.Grids | Syncfusion.Blazor.Grids |

Component name changes

In the previous version, the component names are prefixed with Ejs (example: EjsGrid, EjsChart) which has been changed to the prefix Sf (example: SfGrid, SfChart). Check the following examples for your reference.

<!-- markdownlint-disable MD033 -->

Before v18.1.0.36	From v18.1.0.36
{% highlight razor %}Button{% endhighlight %}	{% highlight razor %}Button{% endhighlight %}
{% highlight razor %}{% endhighlight %}	{% highlight razor %}{% endhighlight %}

Resource changes

Till the previous version, you will be loading scripts `ej2.min.js` and `ejs-interop.min.js` manually in the application, which is not required from 18.1.0.36 version. The script will be loaded from NuGet through static web assets for the components loaded on the page. We have also provided styles as static web assets to load in the application.

<!-- markdownlint-disable MD033 -->

....

{% endhighlight %}

Still, you can load the resource from CRG by disabling default script loading from static web assets by passing arguments to `AddSyncfusionBlazor` service in `~/Startup.cs` or `~/Program.cs`.

DataManager changes

In the previous version, Query's initialization was made as `new ej.data.Query()`. From version 18.0.1.36, it has been changed to `new sf.data.Query()`. Check the following table for examples.

<!-- markdownlint-disable MD033 -->

Before v18.1.0.36	From v18.1.0.36
<code>{% highlight razor %}@using Syncfusion.EJ2.Blazor.Data@{ var chartQuery = \$"new ej.data.Query().where('EmployeeID', 'equal', {employee.EmployeeID}, false)"; } { endhighlight %}</code>	<code>{% highlight razor %}@using Syncfusion.Blazor.Data@{ var chartQuery = \$"new sf.data.Query().where('EmployeeID', 'equal', {employee.EmployeeID}, false)"; } { endhighlight %}</code>

To downgrade the project from 18.1.0.36 version, you need to do the reverse process of the previous steps.

How to Create a Blazor WebAssembly Application with Prerendering

This section explains how to enable prerendering to a Blazor WebAssembly application.

Prerequisites

[.NET 5.0 SDK](#) or later.

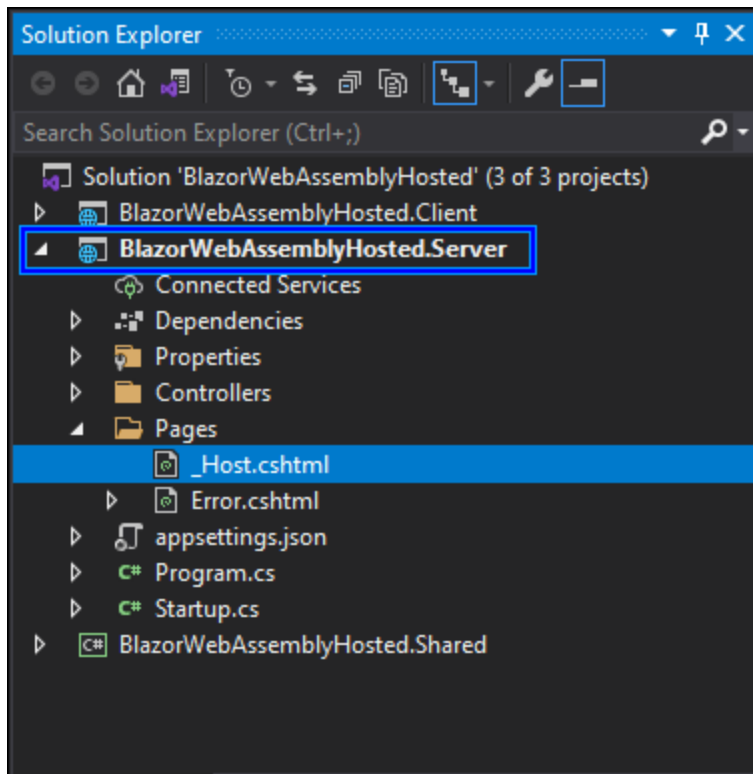
Create a New Project for Blazor WebAssembly ASP.NET Core Hosted Application

1. Create a new [Blazor WebAssembly ASP.NET Core Hosted application](#).
2. Delete `~/wwwroot/index.html` file in the Client project.
3. Remove the below line from Client project's `~/Program.cs` file.

C#

```
// builder.RootComponents.Add<App>("#app");
```

4. Add `~/Pages/_Host.cshtml` file in the Server project.



5. Copy and paste the below code content in the `~/Server/Pages/_Host.cshtml` file.

ASPX-CS

```
@page "/"
@namespace BlazorWebAssemblyHosted.Client.Pages
@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers
@{
    Layout = null;
}
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<meta name="viewport" content="width=device-width, initial-scale=1.0" />
<title>BlazorWebAssemblyHosted</title>
<base href="~/>
<link rel="stylesheet" href="css/bootstrap/bootstrap.min.css" />
<link href="css/bootstrap/bootstrap.min.css" rel="stylesheet" />
<link href="css/app.css" rel="stylesheet" />
<link href="BlazorWebAssemblyHosted.Client.styles.css" rel="stylesheet" />
<link href="_content/Syncfusion.Blazor.Themes/bootstrap4.css"
rel="stylesheet" />
</head>
<body>
<component type="typeof(App)" render-mode="WebAssemblyPrerendered" />
<script src="_framework/blazor.webassembly.js"></script>
</body>
</html>
```


6. Open `Startup.cs` file in the Server project and change endpoint of `MapFallbackToFile` configuration from `index.html` to `/_Host` on `Configure` method.

C#

```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    ....
    ....
    app.UseEndpoints(endpoints =>
    {
        ....
        ....
        endpoints.MapFallbackToFile("/_Host"); // Previously it was mapped into
        "index.html".
    });
}
```

7. Add Syncfusion Blazor service in the `~/Server/Startup.cs` file.

C#

```
using Syncfusion.Blazor;
public void ConfigureServices(IServiceCollection services)
{
    ....
    ....
    services.AddSyncfusionBlazor();
}
```

8. If you don't inject and use `HttpClient` DI on your index page, you can run the application and the component will render in the web browser with prerendering mode.

The created [Blazor WebAssembly ASP.NET Core Hosted application](#) has injected the `HttpClient` DI and fetch the data from server for SfGrid component data source. So, refer to the next topic to resolve the `HttpClient` error on prerendering mode.

Resolving HttpClient Errors on WebAssembly Prerendering

When the index page has injected with the `HttpClient` and tried to prerender on the server, the client will not establish its connection on that time. So, it will throws the runtime exceptions.

InvalidOperationException: An invalid request URI was provided. The request URI must either be an absolute URI or BaseAddress must be set.

The Syncfusion Blazor service has registered the `HttpClient` service itself by default. When you run the `WebAssemblyPrerendered` mode application, the it tried to get the WebAPI with its absolute URI or `BaseAddress`.

If you configure with absolute URI in the `~/Client/Pages/Index.razor` file, you will face another runtime error.

C#

```
protected override async Task OnInitializedAsync()  
{  
    forecasts = await  
    Http.GetFromJsonAsync<WeatherForecast[]>("http://localhost:44376/WeatherFore  
cast");  
}
```

SocketException: An existing connection was forcibly closed by the remote host.

IOException: Unable to read data from the transport connection: An existing connection was forcibly closed by the remote host

HttpRequestException: An error occurred while sending the request.

We are trying to use HTTP from Server to get the fetch data. But, it is also not possible in the prerender mode because of the client is not yet established.

Refer to the below steps to resolve these issues and make the app running with HttpClient in the prerendering mode.

1. Create a public interface in the `~/Shared/WeatherForecast.cs` file on the Shared project to abstract the API call.

C#

```
using System.Threading.Tasks;  
public interface IWeatherForecastService  
{  
    Task<WeatherForecast[]> GetForecastAsync();  
}
```

2. Create a class file `~/Client/Shared/WeatherForecastService.cs` and inherit the class with the new interface `IWeatherForecastService`. Here, the override method `GetForecastAsync` will fetch the data using HTTP Get action.

C#

```
using System.Net.Http;  
using System.Net.Http.Json;  
using System.Threading.Tasks;  
using BlazorWebAssemblyHosted.Shared;  
namespace BlazorWebAssemblyHosted.Client.Shared  
{  
    public class WeatherForecastService : IWeatherForecastService  
    {  
        private readonly HttpClient httpClient;  
        public WeatherForecastService(HttpClient http)  
        {  
            httpClient = http;  
        }  
    }  
}
```

```

}
public async Task<WeatherForecast[]> GetForecastAsync()
{
return await httpClient.GetFromJsonAsync<WeatherForecast[]>
("WeatherForecast");
}
}
}

```

3. create a new class file with same class name on the Server project and inherit with the interface `IWeatherForecastService`. Here, the existing API `~/Server/Controller/WeatherForecastController.cs` data creation process moved into the override method `GetForecastAsync`.

C#

```

using System;
using System.Linq;
using System.Threading.Tasks;
using BlazorWebAssemblyHosted.Shared;
namespace BlazorWebAssemblyHosted.Server.Shared
{
public class WeatherForecastService : IWeatherForecastService
{
private static readonly string[] Summaries = new[]
{
"Freezing", "Bracing", "Chilly", "Cool", "Mild", "Warm", "Balmy", "Hot",
"Sweltering", "Scorching"
};
public async Task<WeatherForecast[]> GetForecastAsync()
{
var rng = new Random();
return Enumerable.Range(1, 5).Select(index => new WeatherForecast
{
Date = DateTime.Now.AddDays(index),
TemperatureC = rng.Next(-20, 55),
Summary = Summaries[rng.Next(Summaries.Length)]
}).ToArray();
}
}
}

```

4. Now, the API controller will have the below changes on `~/Server/Controller/WeatherForecastController.cs` file.

C#

```

using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using System.Collections.Generic;
using BlazorWebAssemblyHosted.Shared;
namespace BlazorWebAssemblyHosted.Server.Controllers

```

```
{
    [ApiController]
    [Route("[controller]")]
    public class WeatherForecastController : ControllerBase
    {
        private readonly IWeatherForecastService weatherForecastService;
        public WeatherForecastController(IWeatherForecastService weatherService)
        {
            weatherForecastService = weatherService;
        }
        [HttpGet]
        public async Task<IEnumerable<WeatherForecast>> Get()
        {
            return await weatherForecastService.GetForecastAsync();
        }
    }
}
```

5. Register the services in both Client and Server project

Client/Program.cs

C#

```
using BlazorWebAssemblyHosted.Shared;
using BlazorWebAssemblyHosted.Client.Shared;
public static async Task Main(string[] args)
{
    ....
    ....
    builder.Services.AddSyncfusionBlazor();
    builder.Services.AddScoped<IWeatherForecastService,
    WeatherForecastService>();
    await builder.Build().RunAsync();
}
```

Server/Startup.cs

C#

```
using BlazorWebAssemblyHosted.Shared;
using BlazorWebAssemblyHosted.Server.Shared;
public void ConfigureServices(IServiceCollection services)
{
    ....
    ....
    services.AddSyncfusionBlazor();
    services.AddScoped<IWeatherForecastService, WeatherForecastService>();
}
```

6. Now, change the DI injection from `HttpClient` to `IWeatherForecastService` on the `~/Client/Pages/Index.razor` file.

ASPX-CS

```
@using BlazorWebAssemblyHosted.Shared
@Inject IWeatherForecastService WeatherForecastService
....
....
@code {
private WeatherForecast[] forecasts;
protected override async Task OnInitializedAsync()
{
forecasts = await WeatherForecastService.GetForecastAsync();
}
}
```

7. Run the application by pressing **F5** key. The Server prerendering will get the data from its local service and when it will render on the Client, the HTTP Get request sent to get the data.

See Also

- [Prerender on ASP.NET Core Razor Component](#)
- [Stateful Reconnection After Prerendering](#)

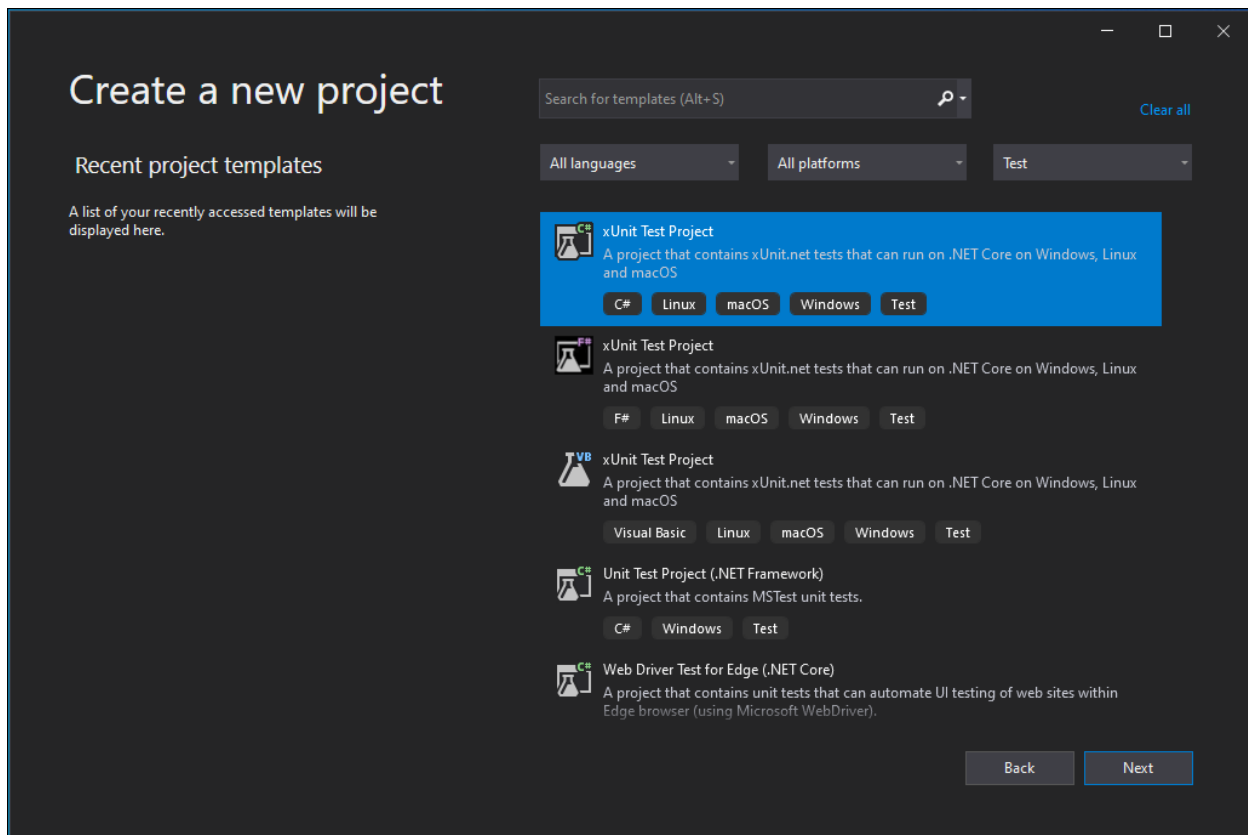
How to Configure Syncfusion Blazor Component in bUnit Testing

This section explains how to configure Syncfusion Blazor component in bUnit testing.

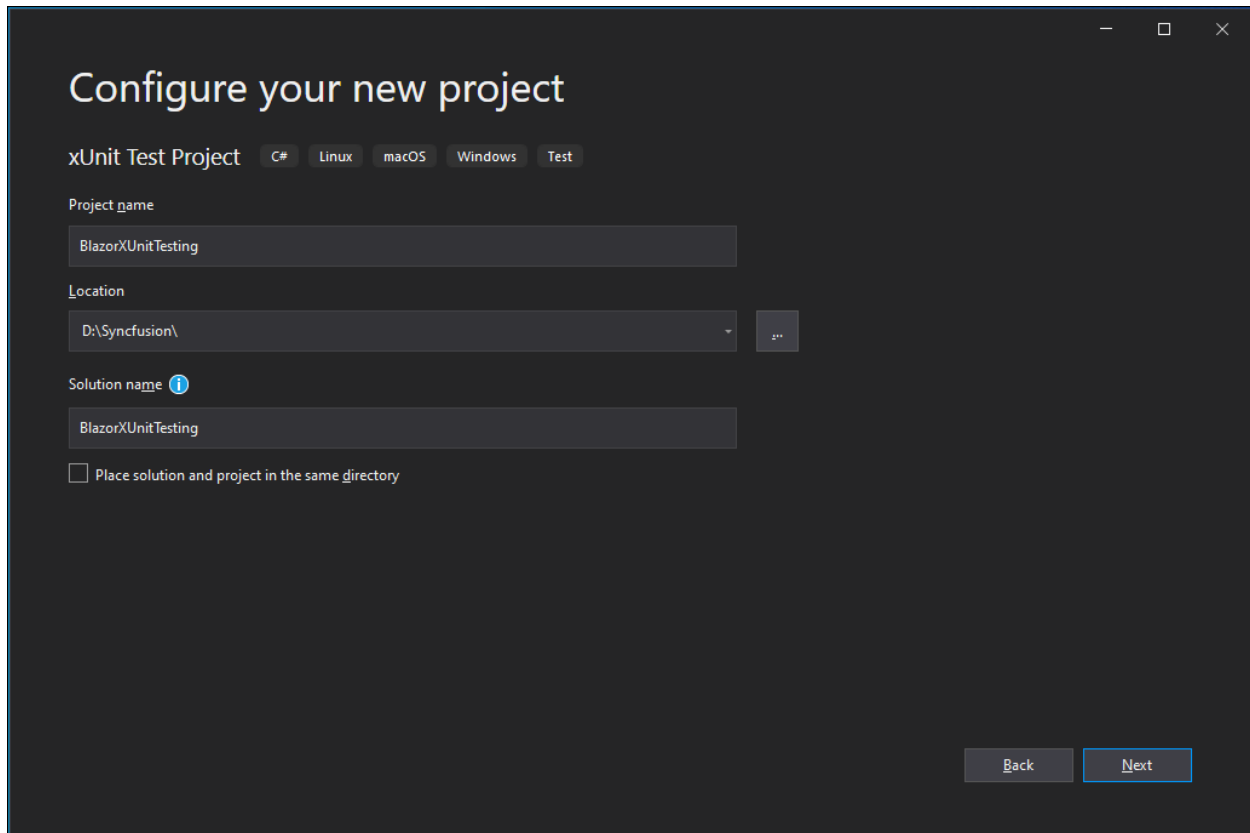
Configure bUnit with xUnit Test Project

Create xUnit Test Project

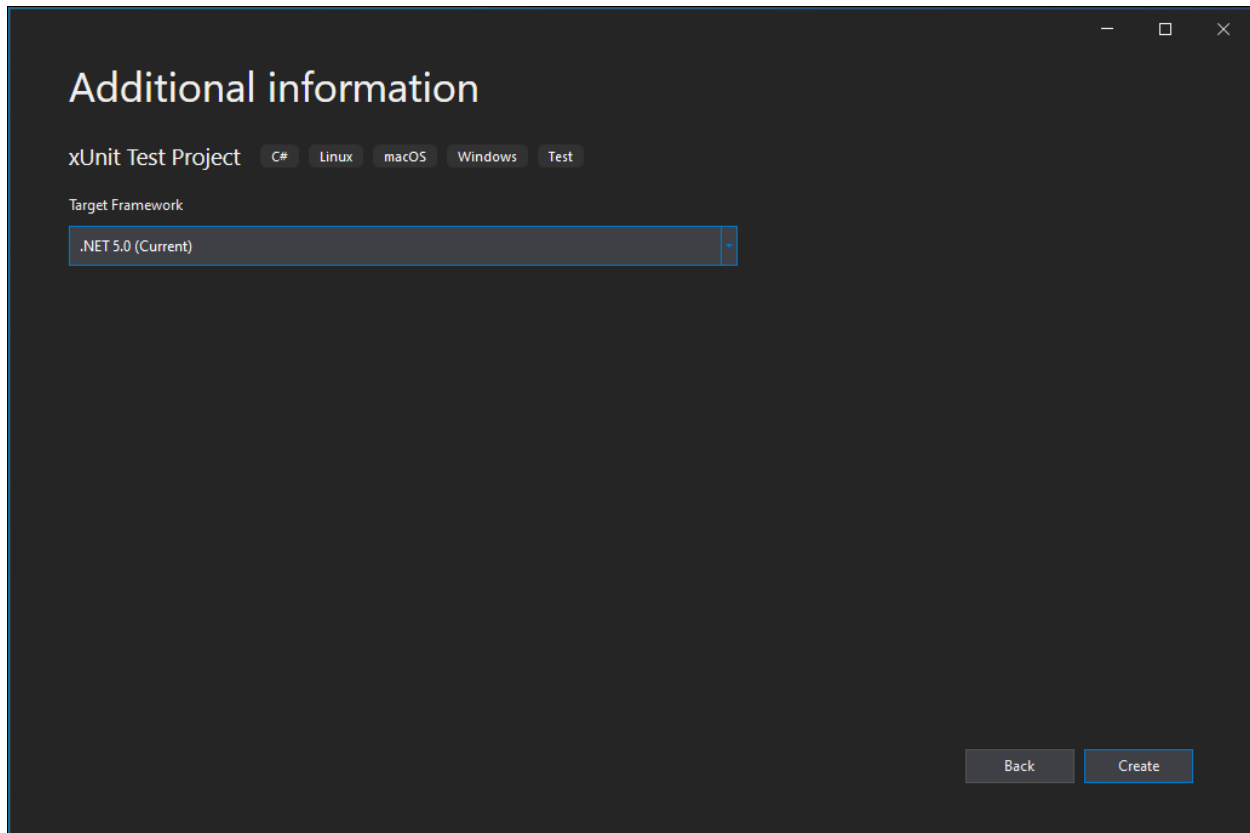
1. Open Visual Studio 2019 and create a new **xUnit Test Project**.



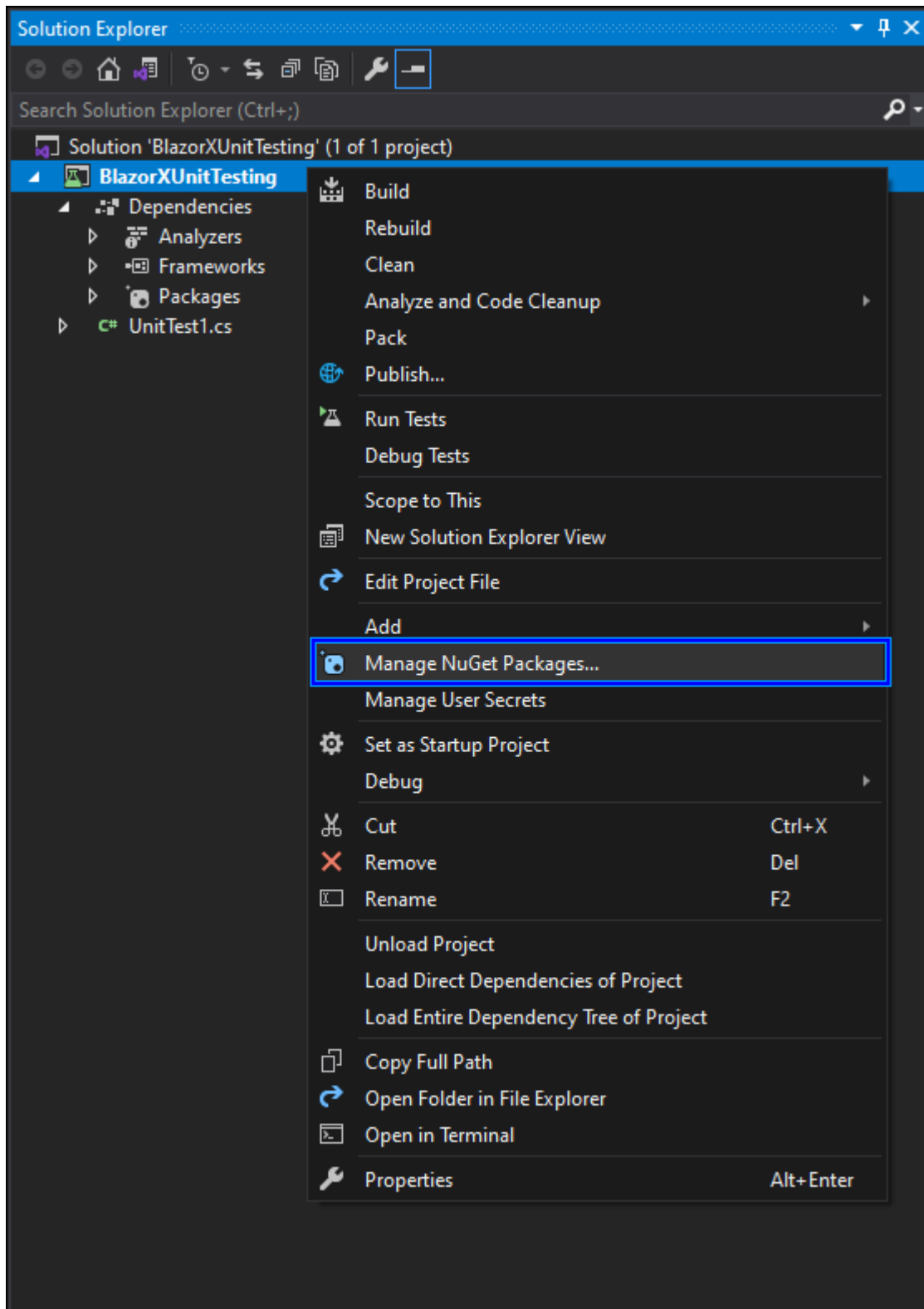
2. Specify the project name and click the **Next** button.



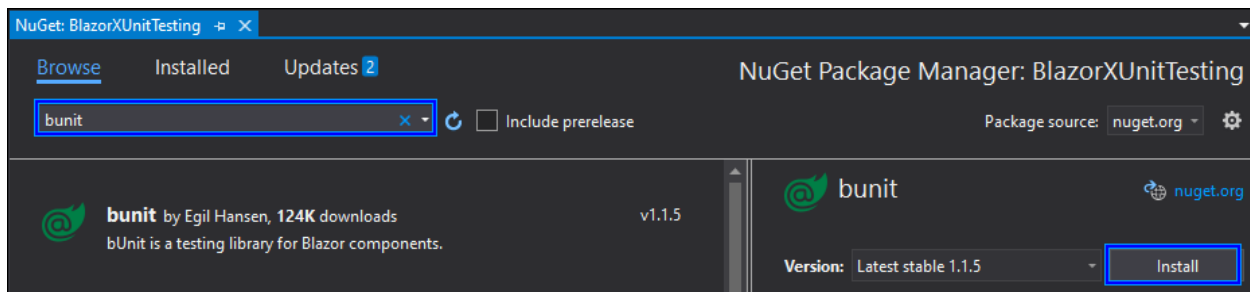
3. Select specific **Target Framework** and click the **Create** button.



4. Right-click on the project in the Solution Explorer and select **Manage NuGet Package**.

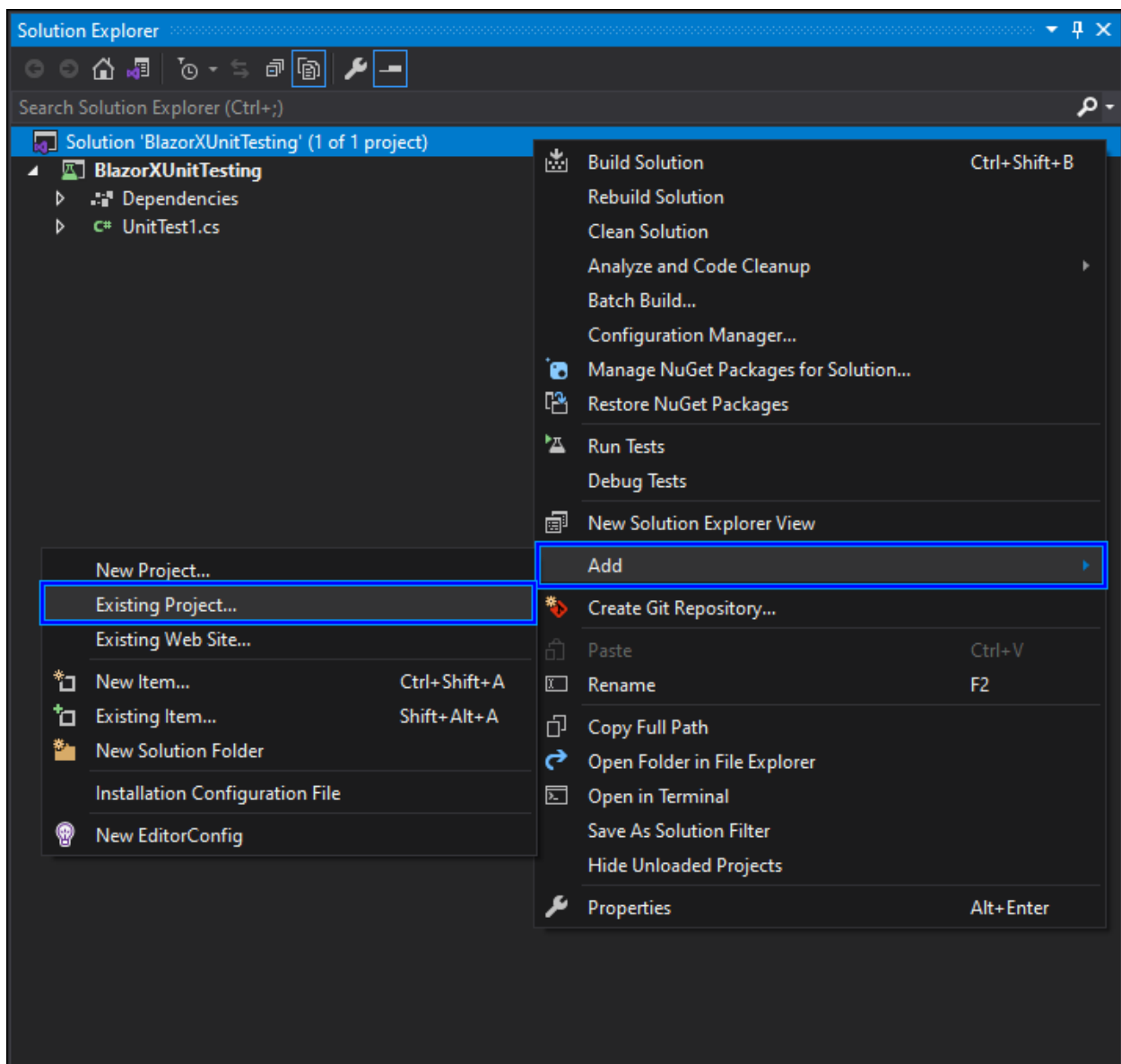


5. Search `bunit` and install both NuGet packages in the test project.



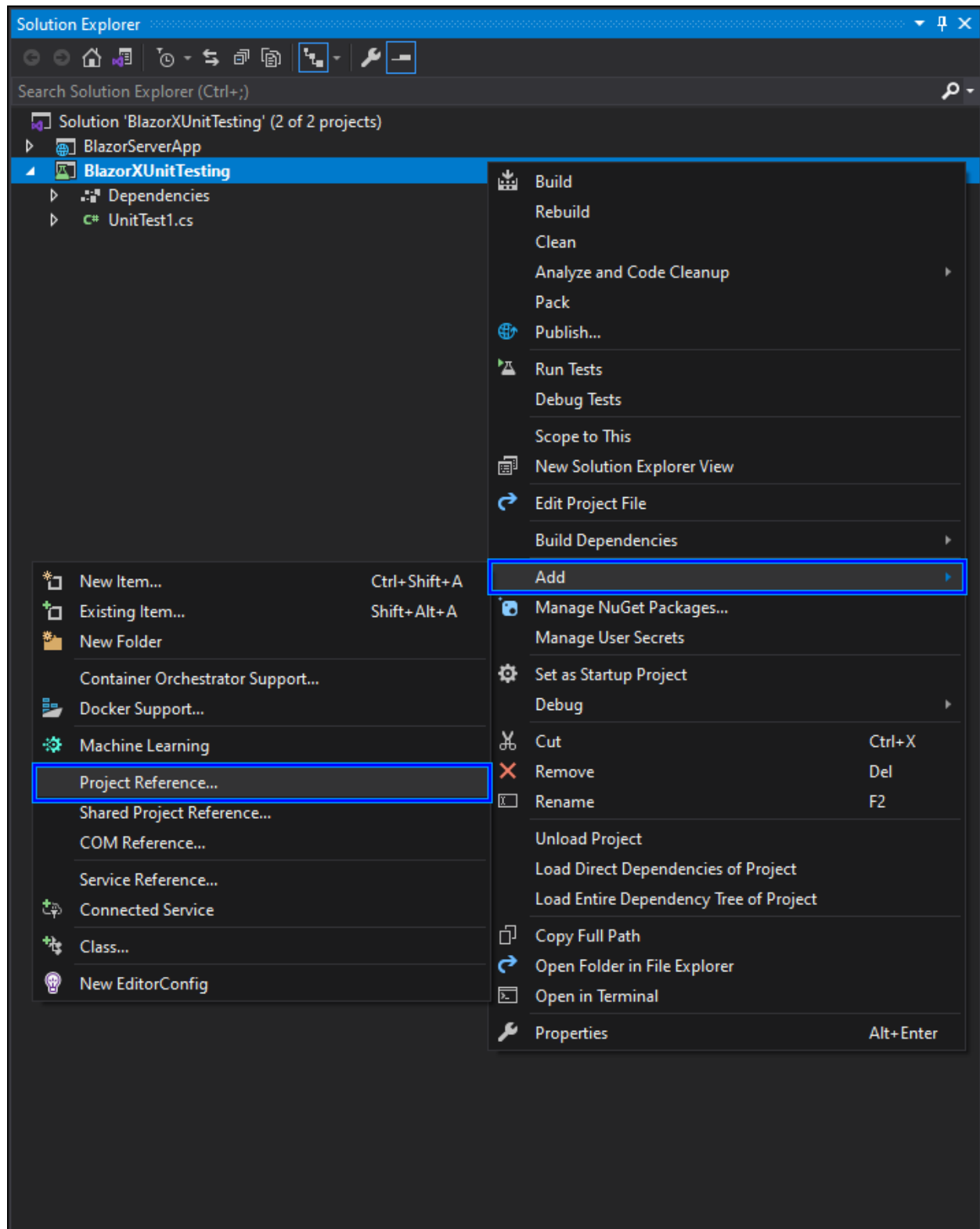
Add Existing Blazor App and Configure it on xUnit Project

1. Right-click on the Solution and select **Add -> Existing Project**. Browse and add your existing project from the local machine.



Refer to [Blazor Server Getting Started](#) documentation, if you don't have any existing application.

- Now, right-click on the xUnit project and select **Add -> Project Reference** and select the added project reference.



3. Add the below Syncfusion Button sample in `~/Pages/Index.razor` file on the Blazor project for testing purpose. You can test your Blazor component from your application instead of the below example component.

ASPX-CS

```
@using Syncfusion.Blazor.Buttons
<SfButton @onclick="OnButtonClick">My Button</SfButton>
<span class="alert alert-info">Count: @clickCount</span>
@code {
private int clickCount = 0;
[Parameter]
public int Step { get; set; } = 1;
private void OnButtonClick()
{
clickCount += Step;
}
}
```

4. Add the below bUnit test cases in the `~/UnitTest1.cs` file on xUnit project.

C#

```
using Xunit;
using Bunit;
using BlazorServerApp.Pages;
using Syncfusion.Blazor;
using Syncfusion.Blazor.Buttons;
using Microsoft.Extensions.DependencyInjection;
namespace BlazorXUnitTesting
{
public class UnitTest1
{
[Fact]
public void TestIndex()
{
using var testContext = new TestContext();
// Add Syncfusion Blazor service and Ignore script isolation.
testContext.Services.AddSyncfusionBlazor(options => {
options.IgnoreScriptIsolation = true;
});
testContext.Services.AddOptions();
// Rendering application Index component (~/Pages/Index.razor).
var indexComponent = testContext.RenderComponent<Index>();
// Find Syncfusion Button component.
var sfButton = indexComponent.FindComponent<SfButton>();
// Find span element.
var span = indexComponent.Find("span.alert.alert-info");
// Assert
// Testing span element markup.
span.MarkupMatches("<span class=\"alert alert-info\">Count: 0</span>");
// Click Syncfusion Button component.
sfButton.Find(".e-btn").Click();
// Testing span element markup again.
```

```
span.MarkupMatches("<span class=\"alert alert-info\">Count: 1</span>");  
}  
}  
}
```

From the above code snippet:

- Created a new `TestContext` and added Syncfusion Blazor Service.

C#

```
using var testContext = new TestContext();  
// Add Syncfusion Blazor service and Ignore script isolation.  
testContext.Services.AddSyncfusionBlazor(options => {  
options.IgnoreScriptIsolation = true;  
});  
testContext.Services.AddOptions();
```

- Rendered the Blazor application's `Index` component which we added in the 3rd step.

C#

```
// Rendering application Index component (~/Pages/Index.razor).  
var indexComponent = testContext.RenderComponent<Index>();
```

- Find Syncfusion Button component and span element from the rendered `Index` component.

C#

```
// Find Syncfusion Button component.  
var sfButton = indexComponent.FindComponent<SfButton>();  
// Find span element.  
var span = indexComponent.Find("span.alert.alert-info");
```

- Test the span element's markup at initial state.

C#

```
// Testing span element markup.  
span.MarkupMatches("<span class=\"alert alert-info\">Count: 0</span>");
```

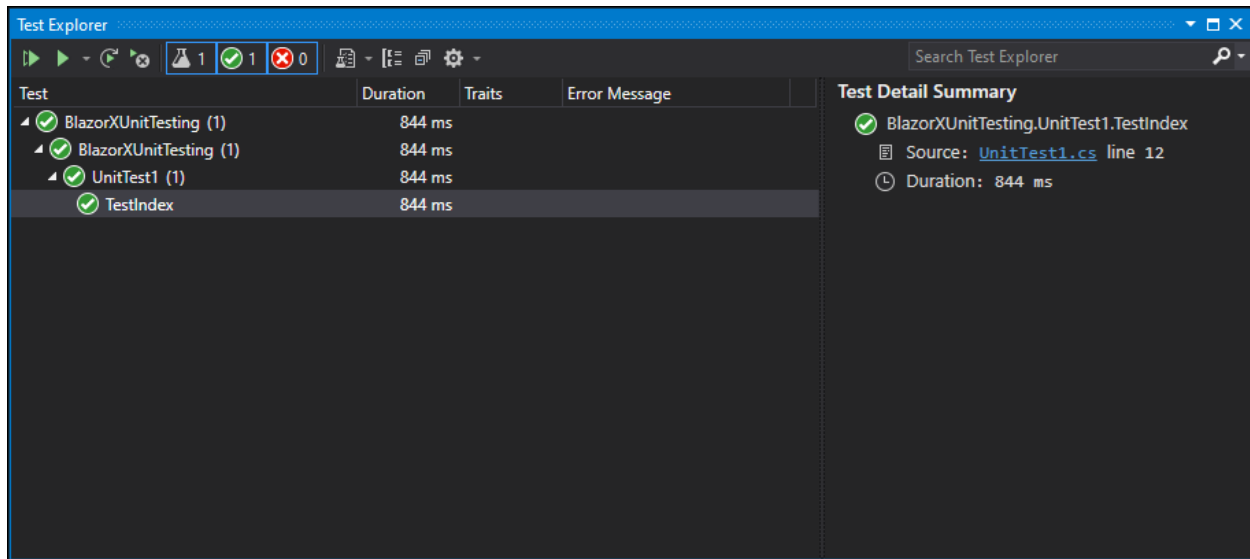
- Find the button element from Syncfusion Button component and trigger the click action. Test the span element's markup state after the button click.

C#

```
// Click Syncfusion Button component.  
sfButton.Find(".e-btn").Click();
```

```
// Testing span element markup again.  
span.MarkupMatches("<span class=\"alert alert-info\">Count: 1</span>");
```

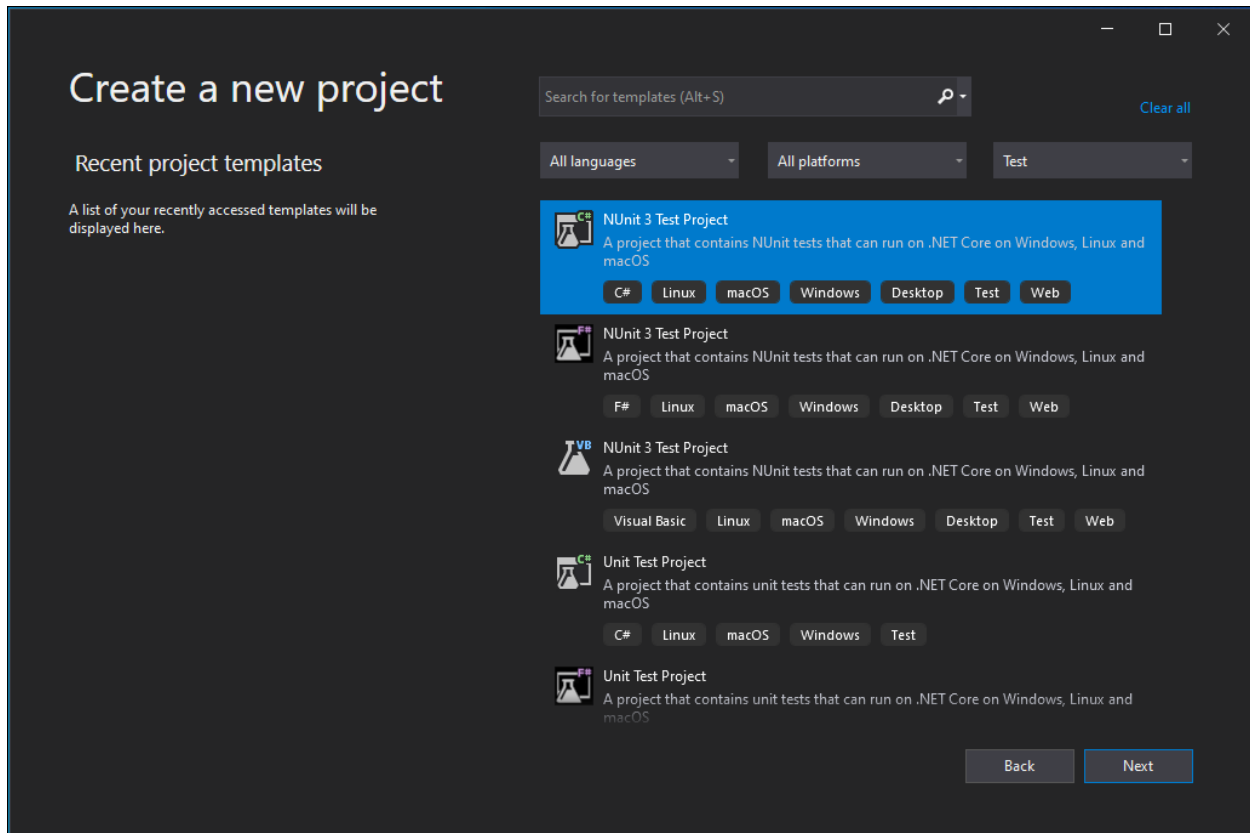
5. Right-click on the xUnit project and select **Run Tests**. The test cases will be run and reports the output.



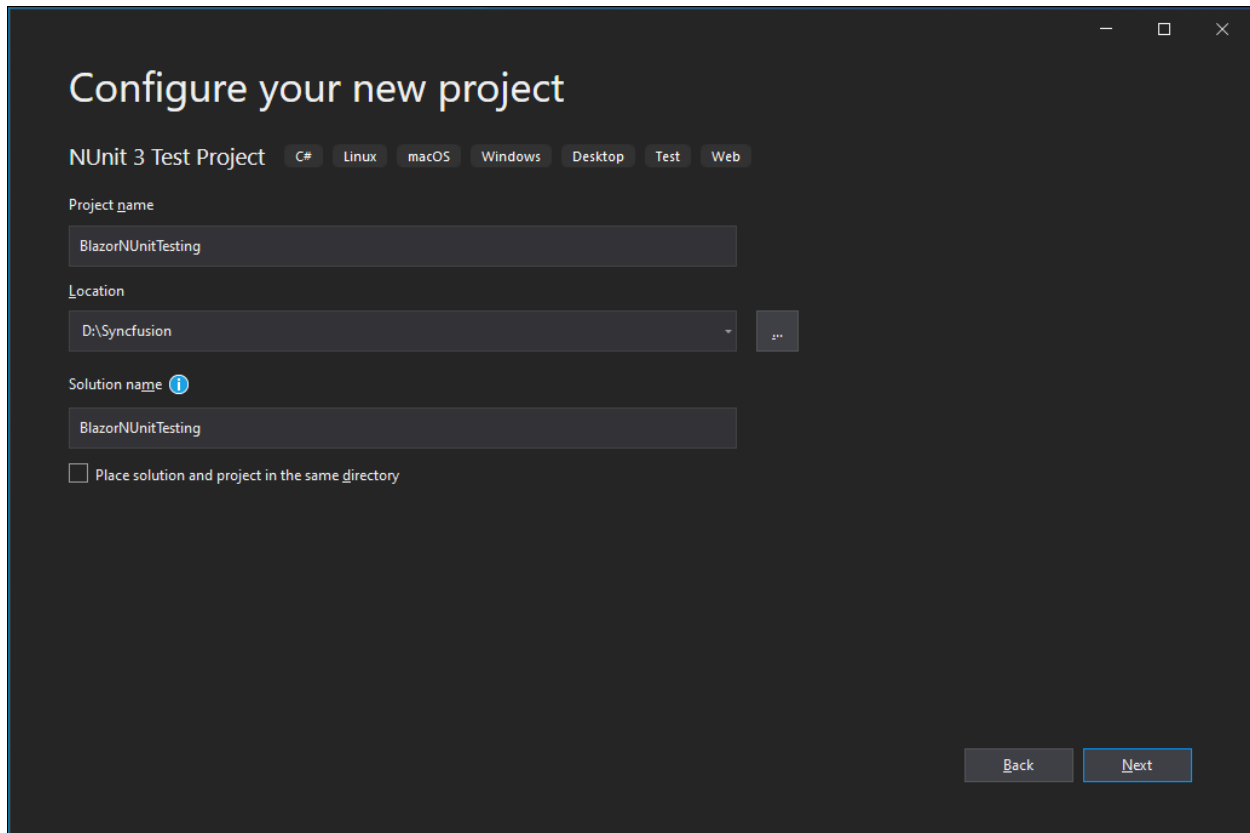
Configure bUnit with NUnit Test Project

Create NUnit Test Project

1. Open Visual Studio 2019 and create a new **NUnit 3 Text Project**.



2. Specify the project name and click the **Next** button.



Configure your new project

NUnit 3 Test Project C# Linux macOS Windows Desktop Test Web

Project name

BlazorUnitTesting

Location

D:\Syncfusion

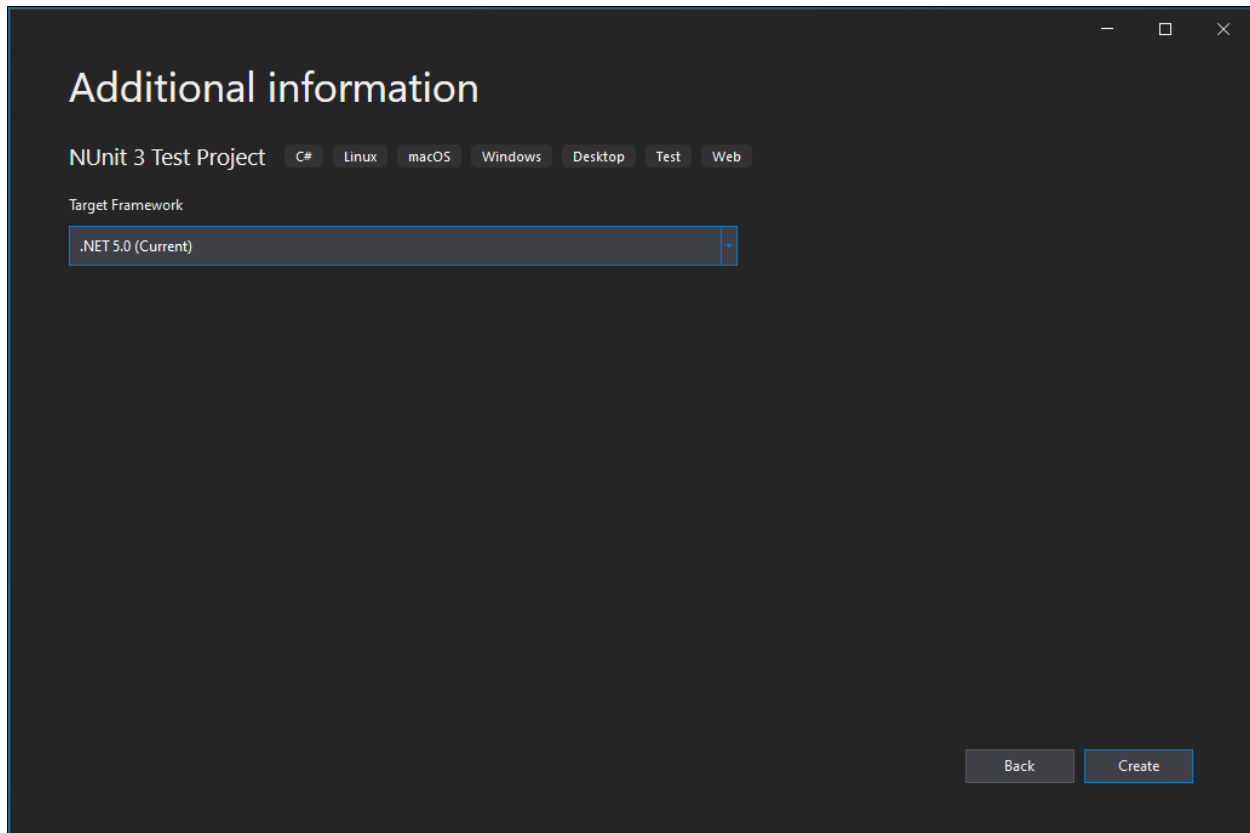
Solution name ⓘ

BlazorUnitTesting

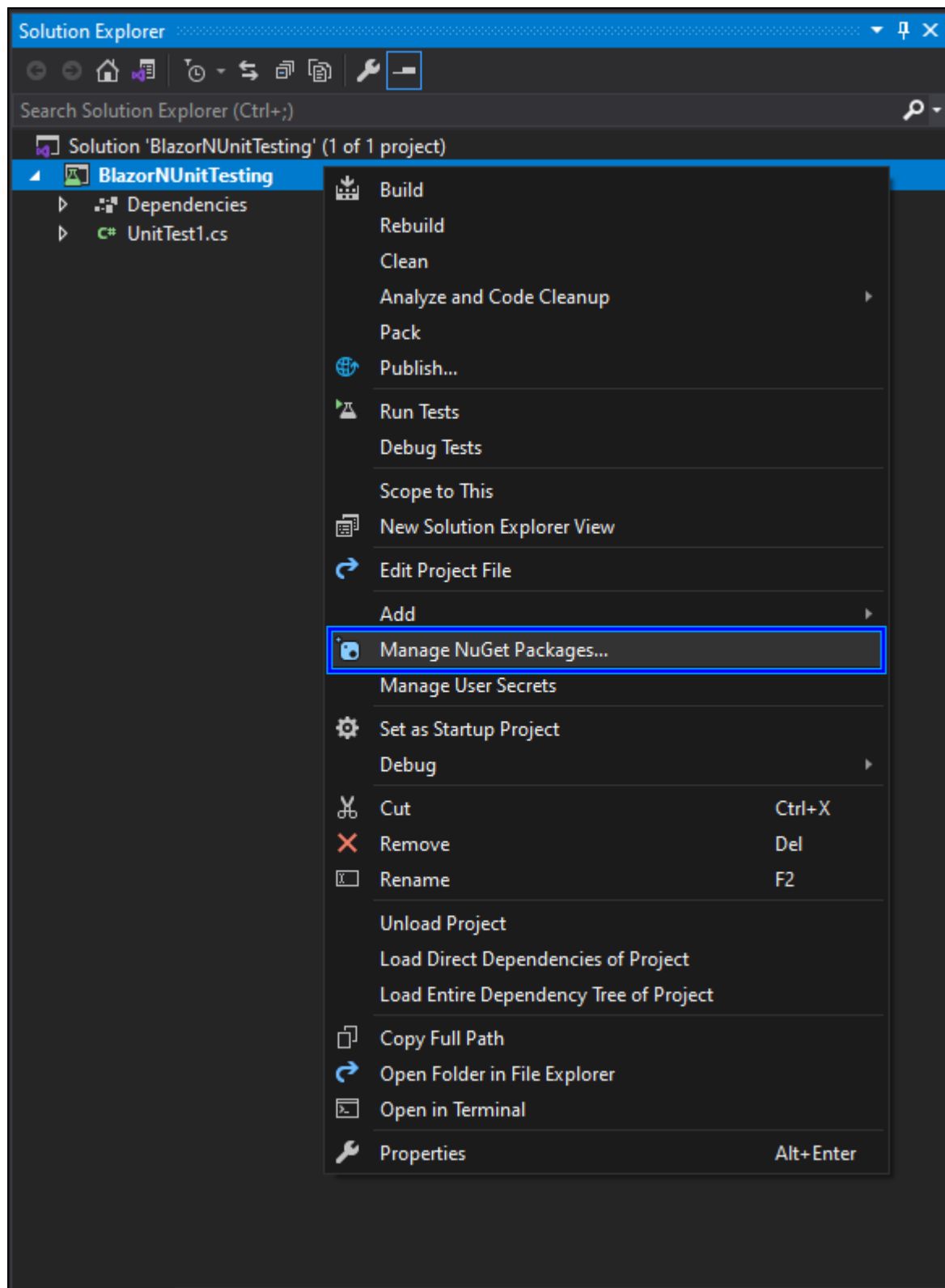
☐ Place solution and project in the same directory

Back Next

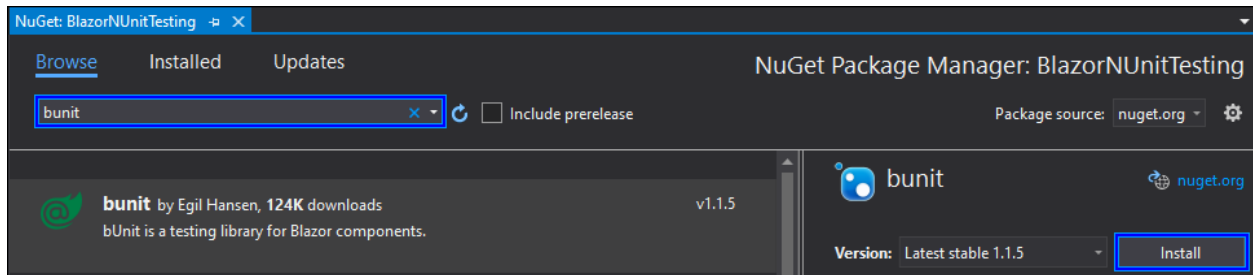
3. Select specific Target Framework and click the Create button.



4. Right-click on the project in the Solution Explorer and select **Manage NuGet Package**.

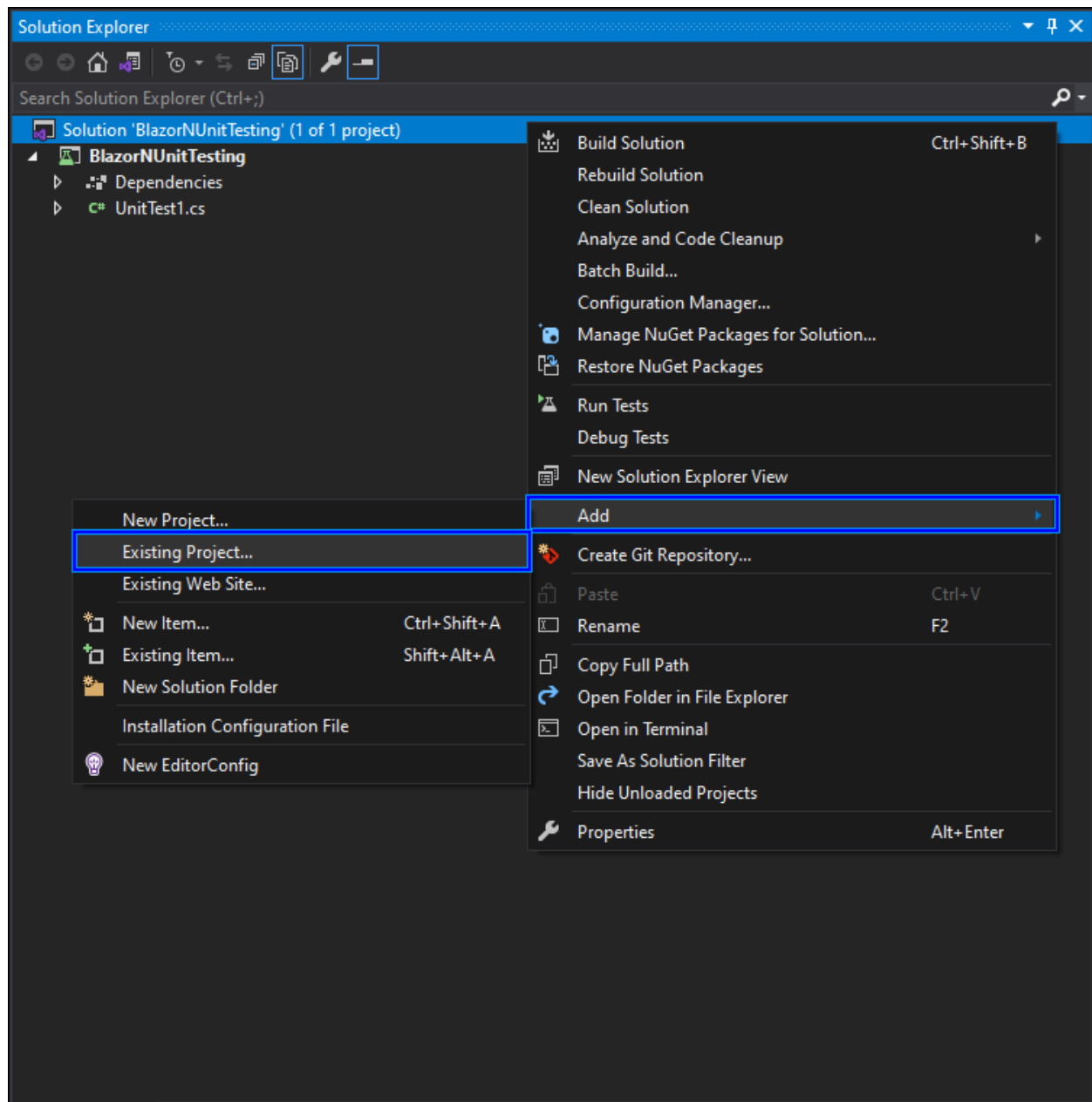


5. Search `bunit` and install both NuGet packages in the test project.



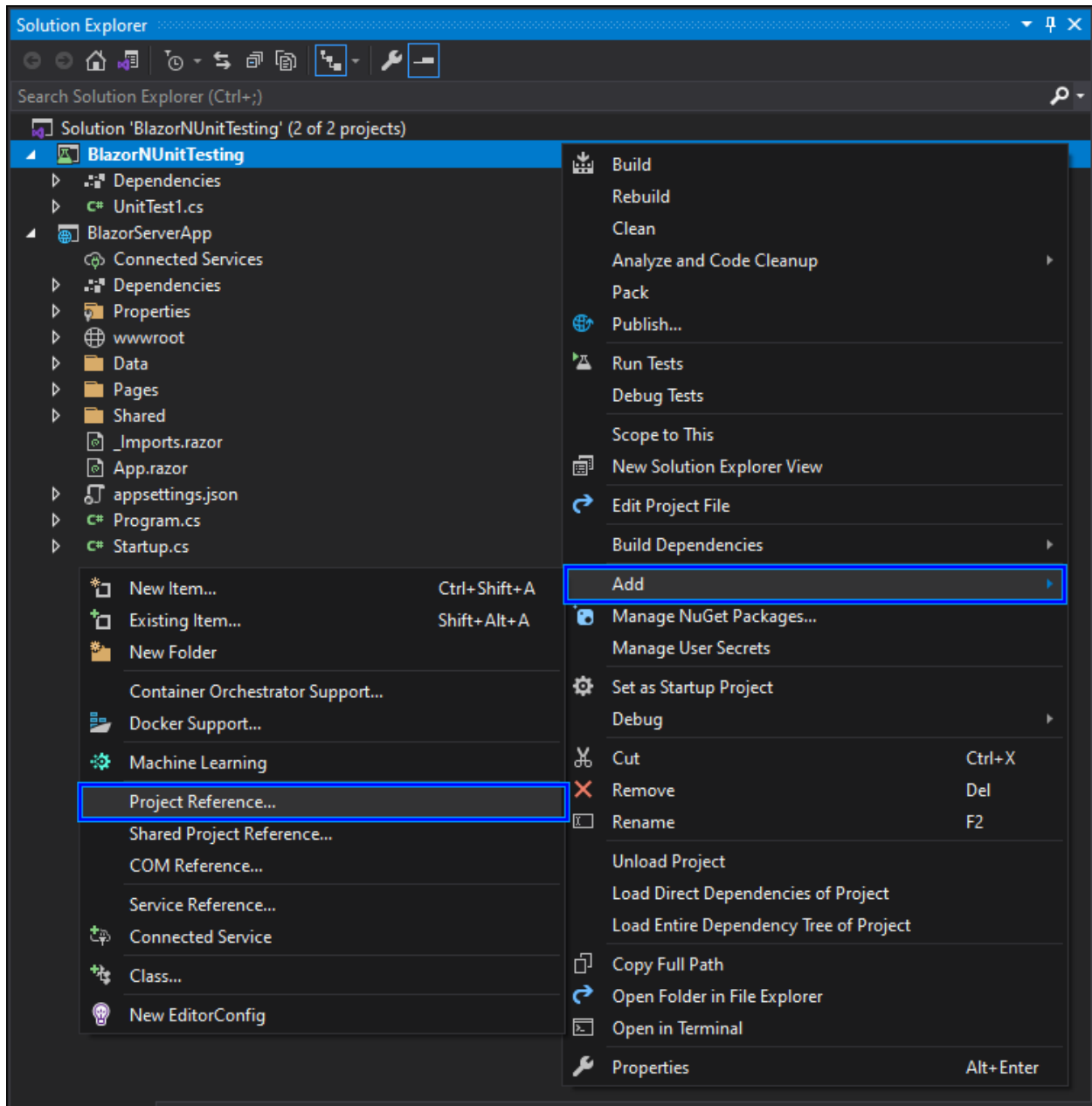
Add Existing Blazor App and Configure it on NUnit Project

1. Right-click on the Solution and select **Add -> Existing Project**. Browse and add your existing project from the local machine.



Refer to [Blazor Server Getting Started](#) documentation, if you don't have any existing application.

2. Now, right-click on the NUnit project and select **Add -> Project Reference** and select the added project reference.



3. Add the below Syncfusion Button sample in `~/Pages/Index.razor` file on the Blazor project for testing purpose. You can test your Blazor component from your application instead of the below example component.

ASPX-CS

```
@using Syncfusion.Blazor.Buttons
<SfButton @onclick="OnButtonClick">My Button</SfButton>
<span class="alert alert-info">Count: @clickCount</span>
@code {
    private int clickCount = 0;
    [Parameter]
    public int Step { get; set; } = 1;
```

```
private void OnButtonClick()
{
    clickCount += Step;
}
}
```

4. Add the below bUnit test cases in the `~/UnitTest1.cs` file on NUnit project.

C#

```
using Bunit;
using NUnit.Framework;
using BlazorServerApp.Pages;
using Syncfusion.Blazor;
using Syncfusion.Blazor.Buttons;
using Microsoft.Extensions.DependencyInjection;
namespace BlazorNUnitTesting
{
    public class Tests
    {
        [Test]
        public void TestIndex()
        {
            // Arrange
            using var testContext = new Bunit.TestContext();
            // Add Syncfusion Blazor service and Ignore script Isolation.
            testContext.Services.AddSyncfusionBlazor(options => {
                options.IgnoreScriptIsolation = true;
            });
            testContext.Services.AddOptions();
            // Rendering application Index component (~/Pages/Index.razor).
            var indexComponent = testContext.RenderComponent<Index>();
            // Find Syncfusion Button component.
            var sfButton = indexComponent.FindComponent<SfButton>();
            // Find span element.
            var span = indexComponent.Find("span.alert.alert-info");
            // Assert
            // Testing span element markup.
            span.MarkupMatches("<span class=\"alert alert-info\">Count: 0</span>");
            // Click Syncfusion Button component.
            sfButton.Find(".e-btn").Click();
            // Testing span element markup again.
            span.MarkupMatches("<span class=\"alert alert-info\">Count: 1</span>");
        }
    }
}
```

From the above code snippet:

- Created a new `TestContext` and added Syncfusion Blazor Service.

C#

```
using var testContext = new Bunit.TestContext();  
// Add Syncfusion Blazor service and Ignore script isolation.  
testContext.Services.AddSyncfusionBlazor(options => {  
    options.IgnoreScriptIsolation = true;  
});  
testContext.Services.AddOptions();
```

- Rendered the Blazor application's **Index** component which we added in the 3rd step.

C#

```
// Rendering application Index component (~/Pages/Index.razor).  
var indexComponent = testContext.RenderComponent<Index>();
```

- Find Syncfusion Button component and span element from the rendered **Index** component.

C#

```
// Find Syncfusion Button component.  
var sfButton = indexComponent.FindComponent<SfButton>();  
// Find span element.  
var span = indexComponent.Find("span.alert.alert-info");
```

- Test the span element's markup at initial state.

C#

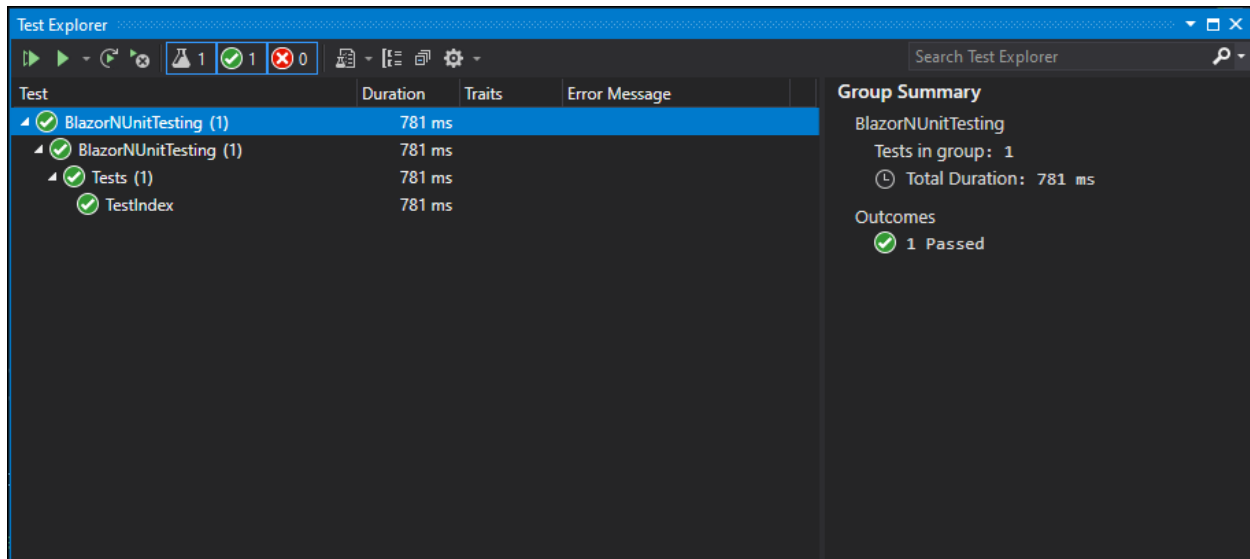
```
// Testing span element markup.  
span.MarkupMatches("<span class=\"alert alert-info\">Count: 0</span>");
```

- Find the button element from Syncfusion Button component and trigger the click action. Test the span element's markup state after the button click.

C#

```
// Click Syncfusion Button component.  
sfButton.Find(".e-btn").Click();  
// Testing span element markup again.  
span.MarkupMatches("<span class=\"alert alert-info\">Count: 1</span>");
```

5. Right-click on the NUnit project and select **Run Tests**. The test cases will be run and reports the output.



Passing Parameters to the Blazor Component Testing

You can set the Blazor component parameter using `SetParametersAndRender` method.

C#

```
[Fact]
public void TestParameter()
{
    using var testContext = new TestContext();
    // Add Syncfusion Blazor service and Ignore script isolation.
    testContext.Services.AddSyncfusionBlazor(options => {
        options.IgnoreScriptIsolation = true;
    });
    testContext.Services.AddOptions();
    // Rendering application Index component (~/Pages/Index.razor).
    var indexComponent = testContext.RenderComponent<Index>();
    // Set Index component parameter Step value.
    indexComponent.SetParametersAndRender(parameters => parameters.Add(p =>
        p.Step, 5));
    // Find Syncfusion Button component.
    var sfButton = indexComponent.FindComponent<SfButton>();
    // Find span element.
    var span = indexComponent.Find("span.alert.alert-info");
    // Assert
    // Testing span element markup initial state.
    span.MarkupMatches("<span class=\"alert alert-info\">Count: 0</span>");
    // Click Syncfusion Button component.
    sfButton.Find(".e-btn").Click();
    // Testing span element markup again.
    span.MarkupMatches("<span class=\"alert alert-info\">Count: 5</span>");
}
```

See Also

- [Create a new bUnit Test Project](#)
- [Test components in ASP.NET Core Blazor](#)

How to Use Syncfusion Blazor ReportViewer

This section contains the references of how to use Syncfusion ReportViewer in Blazor application.

- [How to Use Syncfusion ReportViewer in Blazor WebAssembly Application](#)
- [How to Use Syncfusion ReportViewer in Blazor Server Application](#)
- [Blazor Reporting Components](#)

How to troubleshoot server and client exceptions in Blazor

<!-- markdownlint-disable MD036 -->

Runtime exceptions

- **InvalidOperationException: Cannot provide a value for property 'SyncfusionService' on type 'Syncfusion.Blazor.Namespace.Component'**

You may see the below runtime exception while running the application for the first time.

InvalidOperationException: Cannot provide a value for property 'SyncfusionService' on type 'Syncfusion.Blazor.Namespace.Component'. There is no registered service of type 'Syncfusion.Blazor.SyncfusionBlazorService'.

Cause:

This exception thrown because you missed registering the `SyncfusionBlazorService` in `Startup.cs`.

Solution:

You can register the `SyncfusionBlazorService` in `Startup.cs` file to resolve the exception.

C#

```
using Syncfusion.Blazor;  
public class Startup  
{  
    ....  
    ....  
    public void ConfigureServices(IServiceCollection services)  
    {  
        ....  
        ....  
        services.AddSyncfusionBlazor();  
    }  
    ....  
    ....  
}
```

- **System.NullReferenceException: Object reference not set to an instance of an object**

You may see the below exception while running the application in your production/server machine.

System.NullReferenceException: Object reference not set to an instance of an object.
 at `Syncfusion.Blazor.SyncfusionBlazorService.GetContext()`

 at `Syncfusion.Blazor.BaseComponent.OnInitializedAsync()`

```
<br /> at Syncfusion.Blazor.Charts.SfChart.OnInitializedAsync()
```

```
<br /> at Microsoft.AspNetCore.Components.ComponentBase.RunInitAndSetParametersAsync()
```

Cause:

The production/hosting server machine may have an outdated configuration like .NET Core SDK and .NET Core runtime hosting bundle.

Solution:

Update your dotnet SDK/hosting bundle with the latest version in your production/hosting server machine.

Install the latest dotnet SDK/hosting bundle from [here](#) in your hosting machine to resolve this.

- **The type name 'Shared' does not exist in the type 'SyncfusionBlazor'**

You may see the below exception while running the Syncfusion blazor application.

```
The type name 'Shared' does not exist in the type 'SyncfusionBlazor' SyncfusionBlazor
\SyncfusionBlazor\SyncfusionBlazor\obj\Debug\netcoreapp3.1\Razor\Shared\MainLayout.razo
r.g.cs
```

Cause:

This issue occurred, if you are creating the application name as **SyncfusionBlazor**. The name **SyncfusionBlazor** is already registered in the Syncfusion service handling. So your application name and the internally used class name gets conflict.

Solution:

Try to use a different application name, when you create the blazor application with Syncfusion Blazor components.

- **System.IO.FileLoadException: Could not load file or assembly 'System.Text.Json**

You may get the below exception while running the Syncfusion blazor application.

```
System.IO.FileLoadException: Could not load file or assembly 'System.Text.Json, Version=4.0.1.1,
Culture=neutral PublicKeyToken=cc7b13ffcd2ddd51'. The located assembly's manifest definition does
not match the assembly reference.
```

Cause:

You may have .NET Core SDK older than version 3.1.2, but Syncfusion Blazor components need the latest version of .NET Core SDK as stated in the [pre-requisite](#) documentation.

Solution:

Check installed .NET Core SDK version and update to the latest version. You can also find/download the details of the latest .NET Core SDK version [here](#).

Compile-time errors

- **error CS0121: The call is ambiguous between the following methods or properties: 'Syncfusion.Blazor.SyncfusionBlazor.AddSyncfusionBlazor(Microsoft.Extensions.DependencyInjection.IServiceCollection, bool)' and 'Syncfusion.Blazor.SyncfusionBlazor.AddSyncfusionBlazor(Microsoft.Extensions.DependencyInjection.IServiceCollection, bool)'**

You may see the below compile-time exception while running the application.

The call is ambiguous between the following methods or properties
'Syncfusion.Blazor.SyncfusionBlazor.AddSyncfusionBlazor(Microsoft.Extensions.DependencyInjection.IServiceCollection, bool)'

Cause:

1. You may used `SfPdfViewer` or `SfDocumentEditor` components along with other Syncfusion Blazor components in your application.
2. You may installed both [Syncfusion.Blazor](#) and [individual NuGet packages](#) in the same application.

Solution

1. Starts with Volume 4, 2020 (v18.4.0.30) release, The `SfPdfViewer` and `SfDocumentEditor` components changed its dependency structure.
2. We suggest you to use the [individual NuGet packages](#) to resolve this issue.
 - **The type or namespace name 'EJ2' does not exist in the namespace 'Syncfusion' (are you missing an assembly reference?)**

You may see the below compile-time exception while running the application.

The type or namespace name 'EJ2' does not exist in the namespace 'Syncfusion' (are you missing an assembly reference?)

Cause:

You may use deprecated namespace in the application with Syncfusion Blazor NuGet package version greater than or equal to 18.1.0.36.

Solution

The Syncfusion Blazor library has changed its namespace, NuGet package name, component name from v18.1.0.36. Refer this [documentation](#) to migrate your application with correct namespace. Example: `Syncfusion.EJ2.Blazor` is now modified as `Syncfusion.Blazor`.

- **Found markup element with unexpected name 'Ejs'. If this is intended to be a component, add a @using directive for its namespace**

You may see the below compile-time exception while running the application.

Found markup element with unexpected name 'Ejs'. If this is intended to be a component, add a @using directive for its namespace

Cause:

You may use the deprecated component name in the application with the Syncfusion Blazor NuGet package version greater than or equal to 18.1.0.36.

Solution

The Syncfusion Blazor library has changed its namespace, NuGet package name and component name from v18.1.0.36. Refer this [documentation](#) to migrate your application with correct component name. Example: `EjsGrid` is now modified as `SfGrid`.

Browser console errors

- **net::ERR_ABORTED 404 Error While Using Syncfusion Blazor Static CSS**

You may see the below exception in the web browser dev tool console.

```
GET ./content/Syncfusion.Blazor/styles/bootstrap4.css net::ERRABORTED 404
```

Cause:

If you upgraded Syncfusion Blazor packages from previous versions to 18.4.0.* version or later, this issue occurred.

Solution

The Syncfusion Blazor library provides [individual NuGet packages](#) from the 18.4.0.30 version. If you are using individual NuGet Package in your application, you have to modify the below static web assets (styles) reference from `Syncfusion.Blazor` to `Syncfusion.Blazor.Themes` in the application to resolve this issue.

HTML

```
<head>
...
...
<link href="_content/Syncfusion.Blazor.Themes/bootstrap4.css"
rel="stylesheet" />
</head>
```

Do not use both `Syncfusion.Blazor` and individual NuGet packages in the same application. It will throw ambiguous errors while compiling the project.

- **An unhandled exception has occurred. See browser dev tools for details**

You may see the below exception at the bottom of the page.

```
An unhandled exception has occurred. See browser dev tools for details
```

Cause:

There may be script errors thrown in the JS Interop or client script source.

Solution:

Open the browser dev tool by pressing the `F12` key and navigate to the `Console` tab.

Check the console error with the following topics to resolve the issue or report us the console error through our [support ticket](#).

- **Unhandled exception rendering component: Could not find 'loadScripts' in 'window.sfBlazor'.**

You may see the below exception in the web browser dev tool console.

Could not find 'loadScripts' in 'window.sfBlazor'.

Cause:

This script error may throw because of browser cache or outdated script reference in the application.

Solution:

We recommend you to clear the browser cache to resolve the above script error in v18.2.0.44 or later.

- **net::ERR_ABORTED 404 Error While Using Syncfusion Blazor Static files in modified base path or hosted as sub path app.**

You may face the below exception when deploying the blazor application as Sub application.

GET ./<SUB-PATH>/<SUB-PATH>content/Syncfusion.Blazor/<Scripts and CSSs references>
net::ERR_ABORTED 404

For this, we need to configure the Base path configuration in the root application's `startup.cs` and `_Host.cshtml`.

Cause:

For sub-path hosting we need to configure the base path related configuration. If we missed to configure it leads to this errors.

Solution:

We need to configure the base path in our application when we are hosting the app as Sub-URL like below.

```
| In _Host.cshtml File | In Startup.cs File |
| ----- | ----- |
| <base href="/myblazorapp/" /> | app.UsePathBase("/myblazorapp");|
```

The trailing slash is must for '_Host.cshtml' base path configuration.

For further details, please refer this [MSDN documentation](#) for your reference.

See also

- [How to solve AOT publish failure when using Syncfusion.Blazor](#)

Installation

Web Installer

Downloading Syncfusion Blazor web installer

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

- Trial Installer
- Licensed Installer

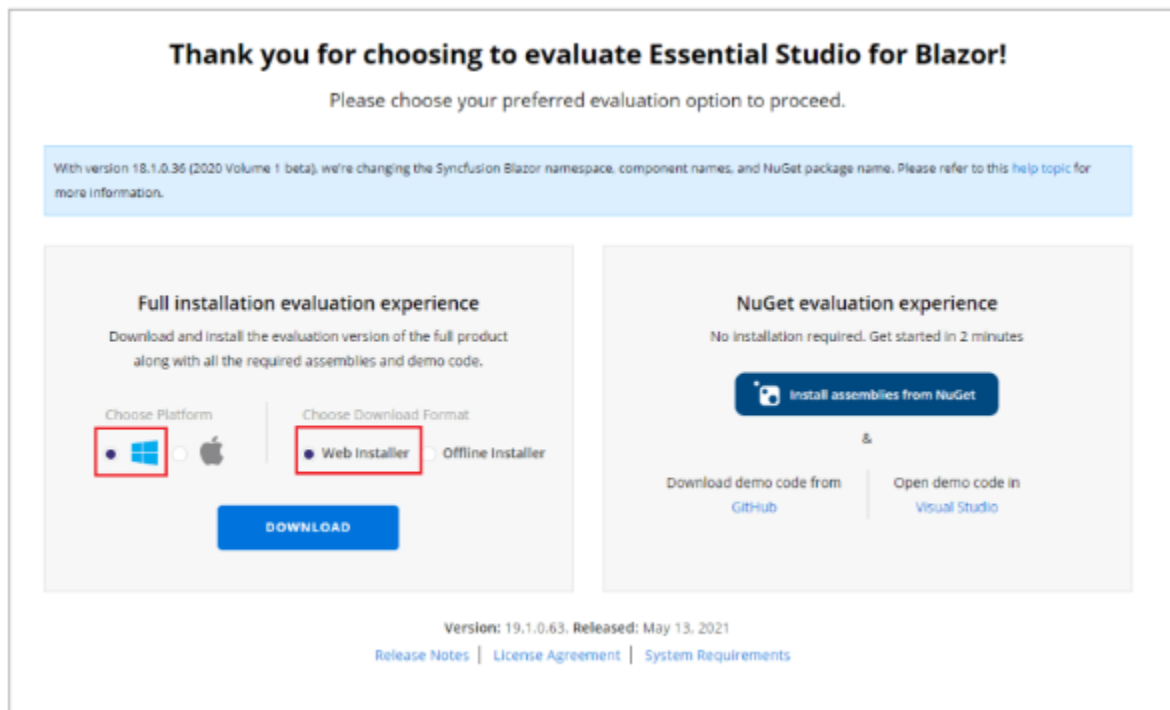
Download Free Trial Version

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

- Download Free Trial Setup
- Start Trials if using components through [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 Blazor platform.
2. After completing the required form or logging in with your registered Syncfusion account, you can download the Blazor trial installer from the confirmation page. (as shown in below screenshot.)



3. After downloading, the Syncfusion Blazor web installer can be unlocked using Syncfusion registered login credential.

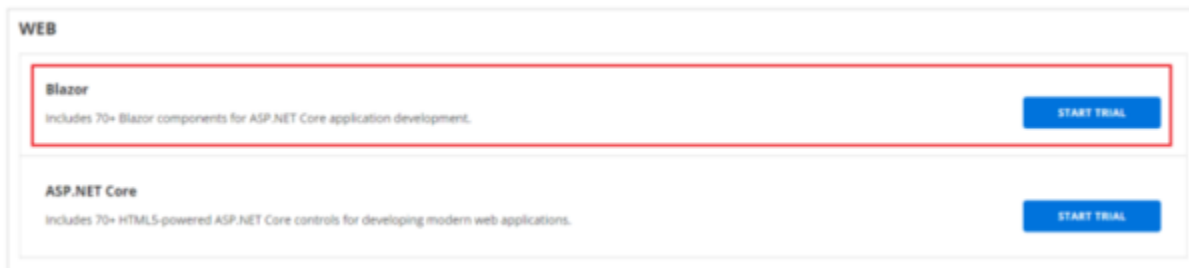
- Before the trial license expires, you can download the blazor installer at any time from your registered account's [Trials & Downloads](#) page (as shown in below screenshot.)
- Click the Download (element 1 in the screenshot below) button to download the blazor web installer.



Start Trials if using components through [NuGet.org](#)

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

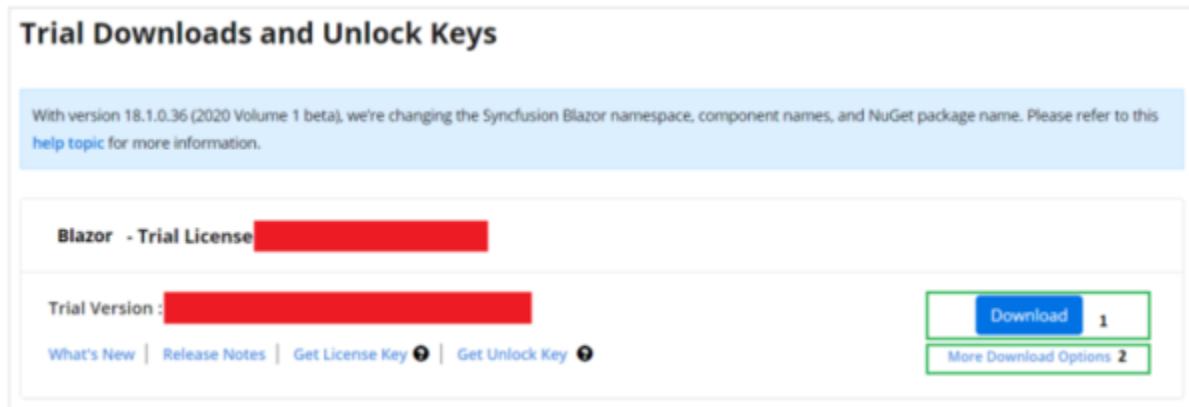
- You can start your 30-day free trial for Blazor from the [Start Trial](#) page from your account.



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

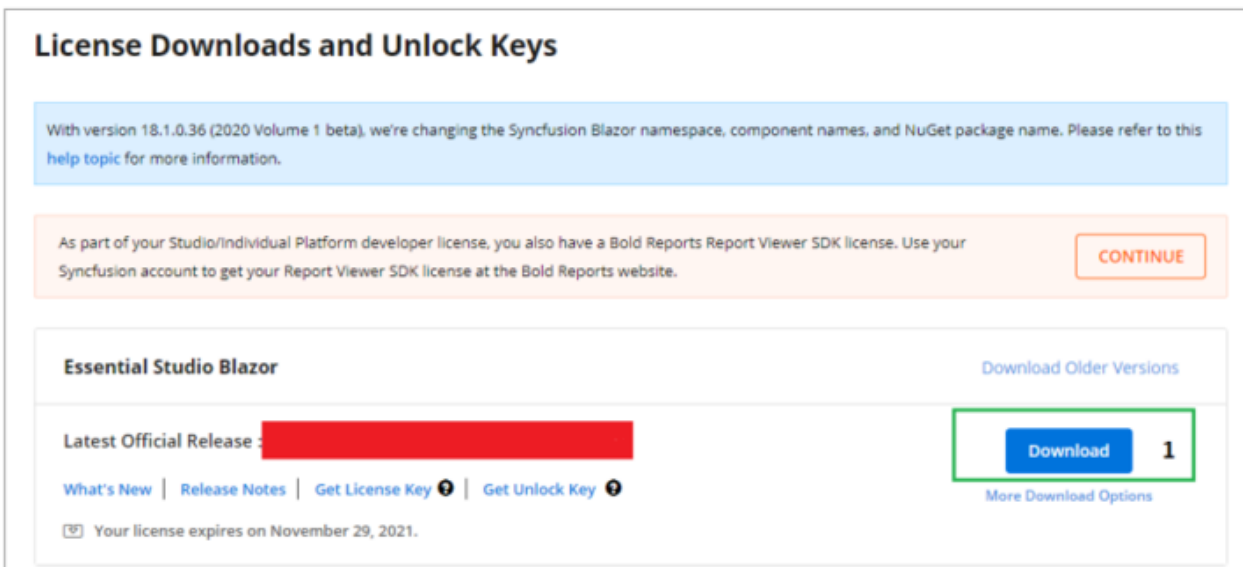
- After you've started the trial, go to the [Trials & Downloads](#) page to get the latest version trial installer. You can generate the [unlock key](#) and [license key](#) here at any time before the trial period expires. (as shown in below screenshot.)



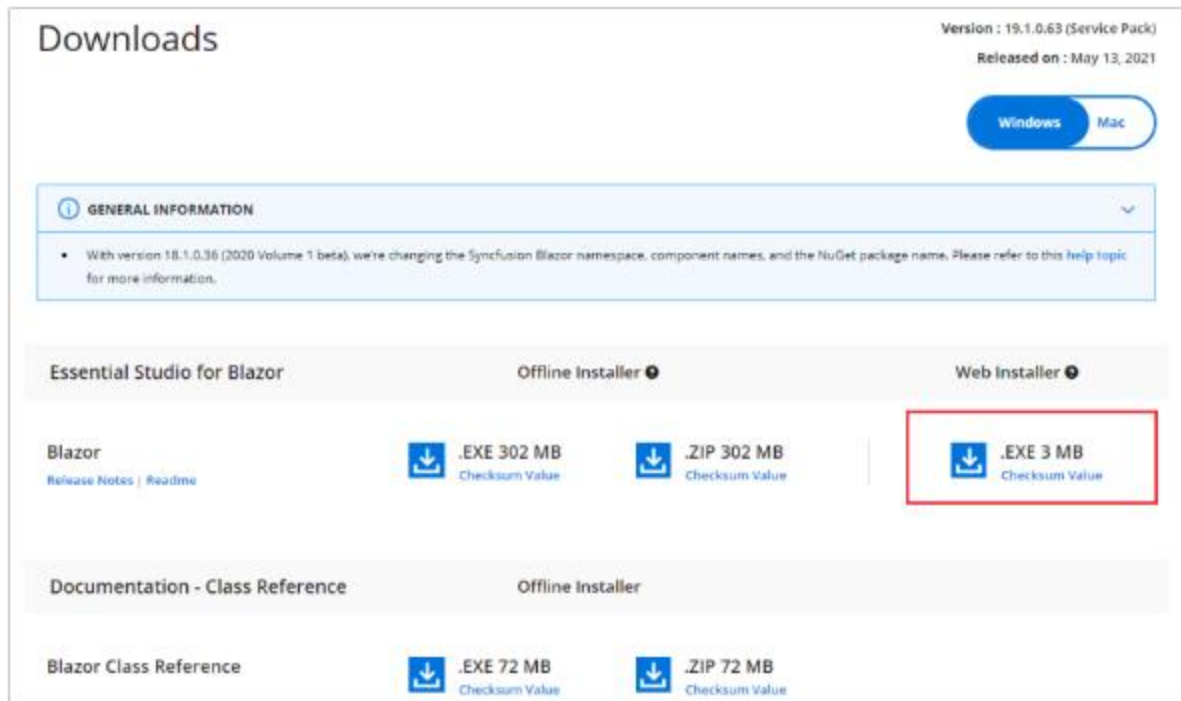
5. You can find your current active trial products on the [Trials & Downloads](#) page.

Download the Licensed 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 blazor web installer.



4. Before the license expires, you can download the installer at any time from your registered account's [License & Downloads](#) page (See the screenshot below.)



5. After downloading, the Syncfusion Blazor web installer can be unlocked using Syncfusion registered login credential.

Note For Syncfusion trial and licensed products, there is no separate web installer. Based on your account license, Syncfusion trial or licensed products will be installed via web installer.

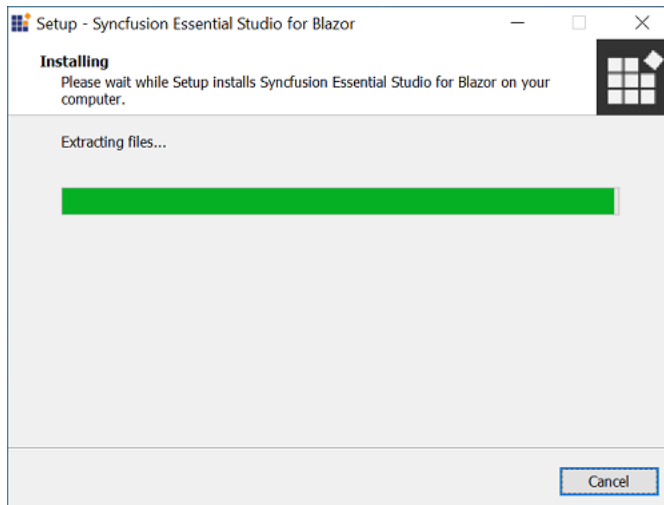
You can also refer to the [web installer](#) links for step-by-step installation guidelines.

Installing Syncfusion Blazor web installer

Installation

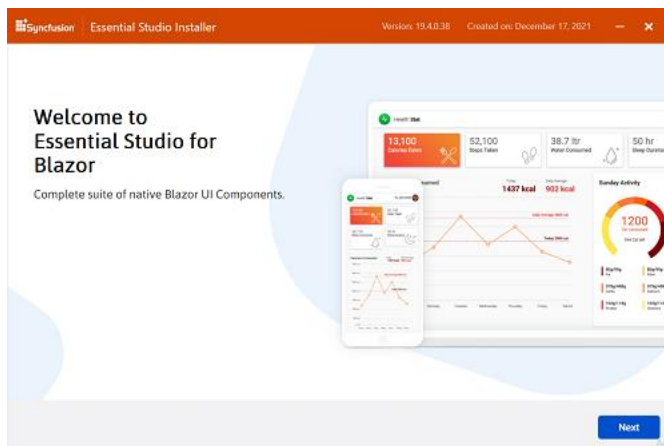
The steps below show how to install Essential Studio Blazor Web Installer.

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



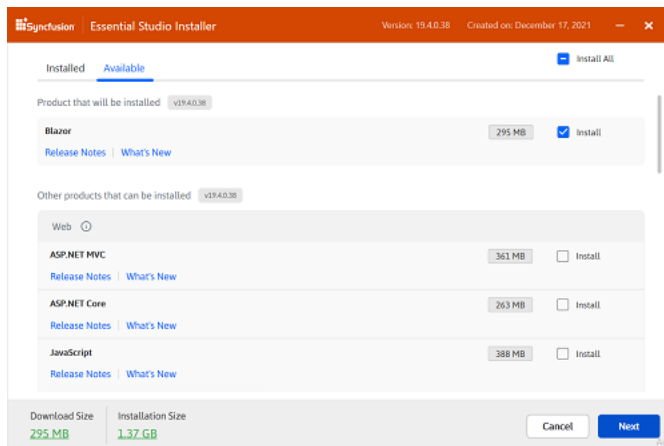
Note The installer wizard extracts the syncfusionessentialblazorwebinstaller_{version}.exe dialog, which displays the package's unzip operation.

2. The Syncfusion Blazor Web Installer's **welcome wizard** will be displayed. Click the Next button.



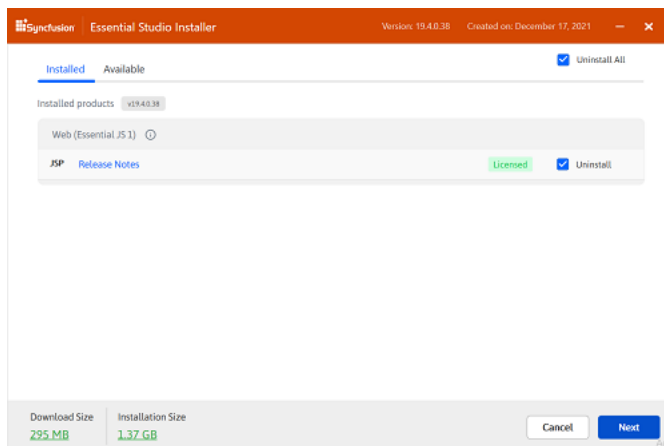
3. The **Platform Selection Wizard** will appear. From the **Available** tab, select the products to be installed. Select the **Install All** checkbox to install all products.

Available



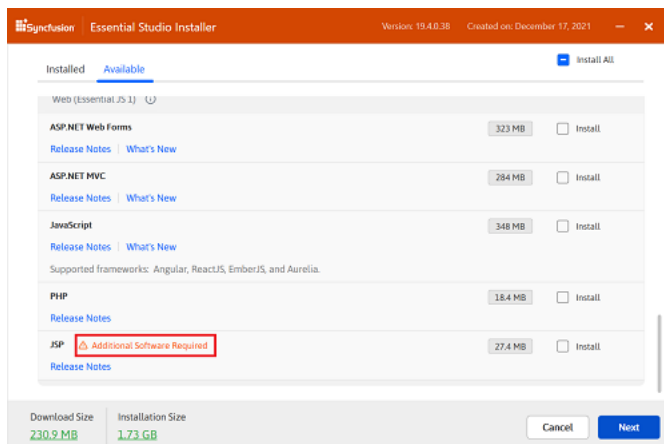
If you have multiple products installed in the same version, they will be listed under the **Installed** tab. You can also select which products to uninstall from the same version. Click the Next button.

Installed

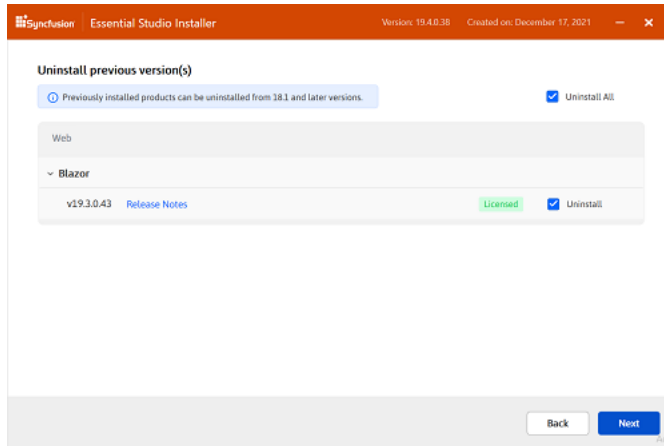


IMPORTANT If the required software for the selected product isn't already installed, the **Additional Software Required** alert will appear. You can, however, continue the installation and install the necessary software later.

Required Software

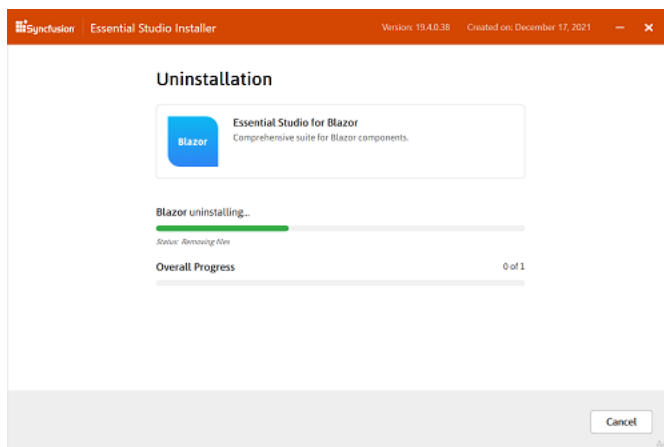


4. If previous version(s) for the selected products are installed, the **Uninstall previous version wizard** will be displayed. You can see the list of previously installed versions for the products you've chosen here. To remove all versions, check the **Uninstall All** checkbox. Click the Next button.

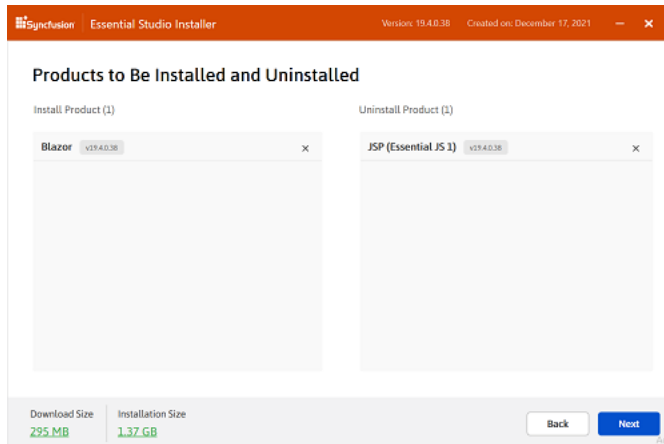


Note From the 2021 Volume 1 release, Syncfusion has provided option to uninstall the previous versions from 18.1 while installing the new version.

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

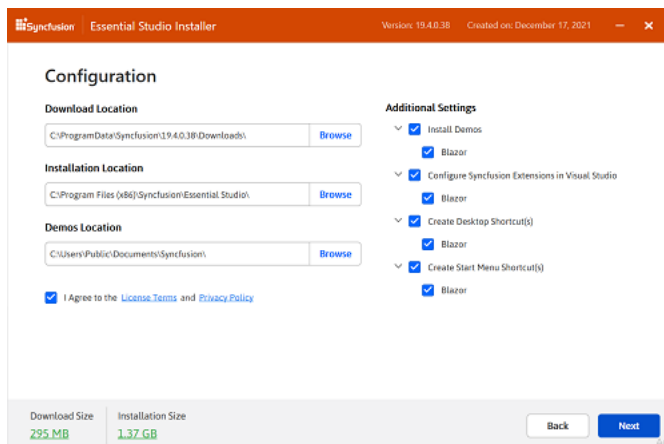


6. The **Confirmation Wizard** will appear with the list of products to be installed/uninstalled. You can view and modify the list of products that will be installed and uninstalled from this page.



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

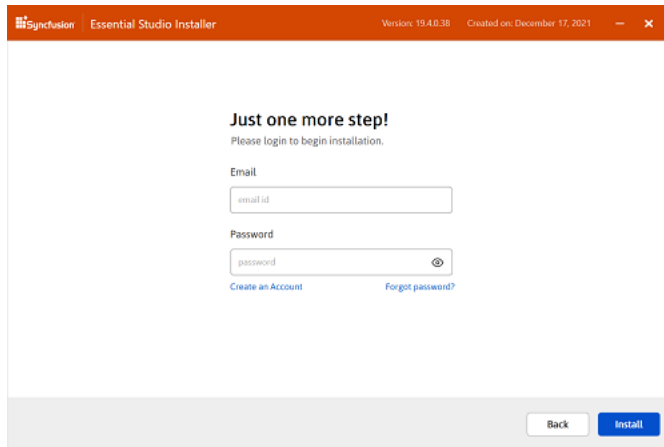
- The **Configuration Wizard** will appear. You can change the Download, Install, and Demos locations from here. You can also change the Additional settings on a product-by-product basis. Click Next to install with the default settings.



Additional settings

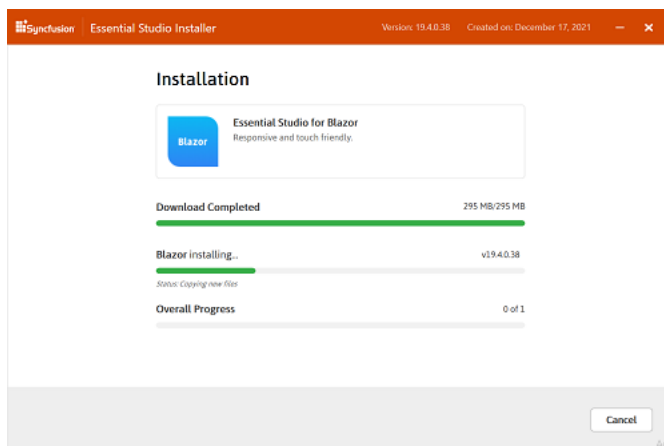
- Select the **Install Demos** check box to install Syncfusion samples, or leave the check box unchecked, if you do not want to install Syncfusion samples.
 - Select the **Configure Syncfusion Extensions controls in Visual Studio** checkbox to configure the Syncfusion Extensions in Visual Studio or clear 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.
- After reading the License Terms and Conditions, check the **I agree to the License Terms and Privacy Policy** check box. Click the Next button.
 - 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.



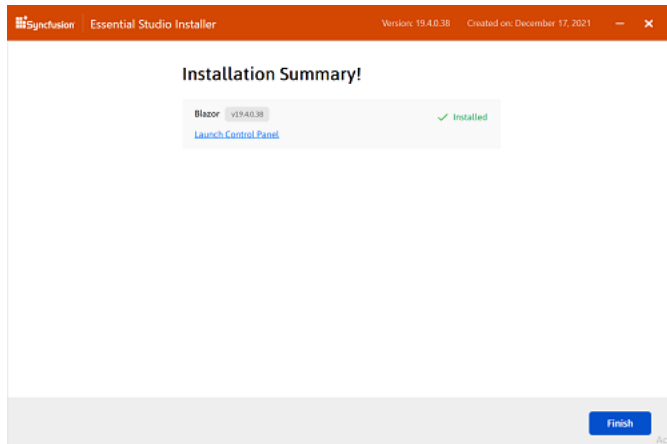
IMPORTANT The products you have chosen will be installed based on your Syncfusion License (Trial or Licensed).

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

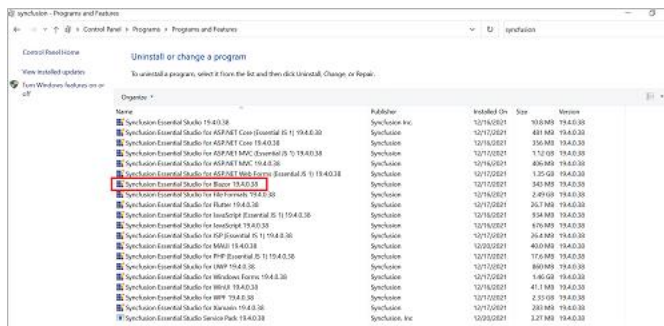


To open the Syncfusion Control Panel, click **Launch Control Panel**.

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.



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

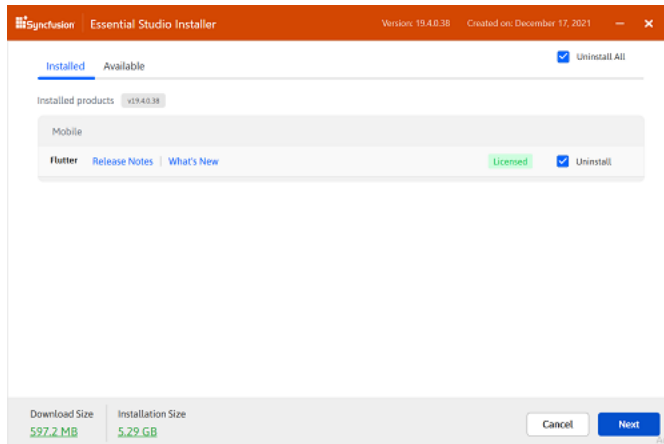
Syncfusion Blazor installer can be uninstalled in two ways.

- Uninstall the Blazor using the Syncfusion Blazor web installer
- Uninstall the Blazor from Windows Control Panel

Follow either one of the option below to uninstall Syncfusion Essential Studio Blazor installer,

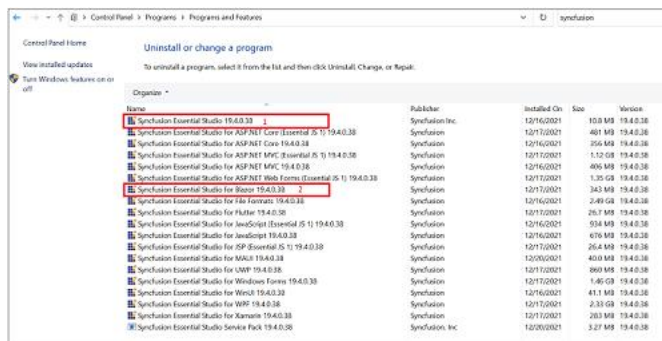
Option 1: *Uninstall the Blazor using the Syncfusion Blazor web installer*

Syncfusion provides the option to uninstall products of the same version directly from the Web Installer application. Open the Syncfusion Essential Studio Blazor Online Installer file from downloaded location by double-clicking it. Select the products to be uninstalled from the list, and Web Installer will uninstall them one by one.



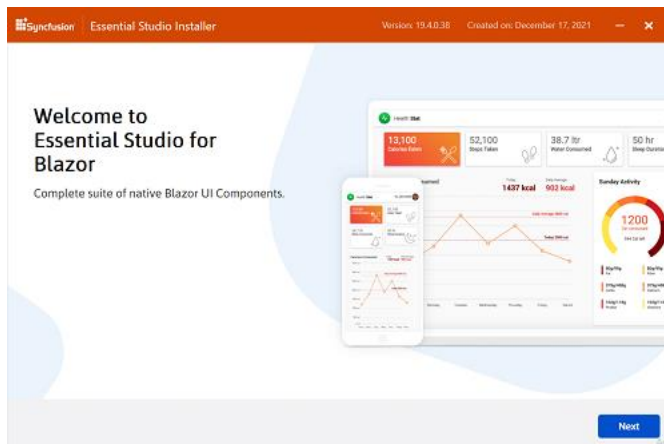
Option 2: Uninstall the Blazor from Windows Control Panel

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 Blazor alone by selecting the **Syncfusion Essential Studio for Blazor {version}** entry (element 2 in the below screenshot) from the Windows control panel.



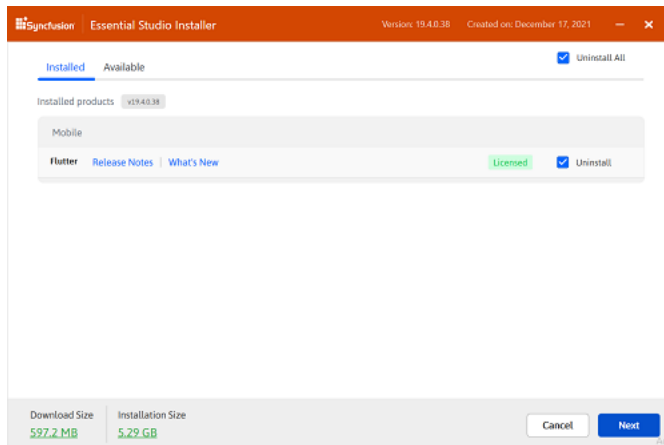
Note If the **Syncfusion Essential Studio for Blazor {version}** entry is selected from the Windows control panel, the Syncfusion Essential Studio Blazor alone will be removed and the below default MSI uninstallation window will be displayed.

1. The Syncfusion Blazor Web Installer's **welcome wizard** will be displayed. Click the Next button



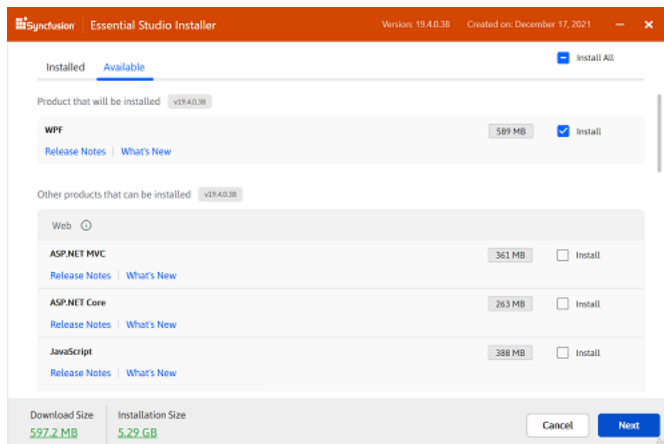
2. The **Platform Selection Wizard** will appear. From the **Installed** tab, select the products to be uninstalled. To select all products, check the **Uninstall All** checkbox. Click the Next button.

Installed

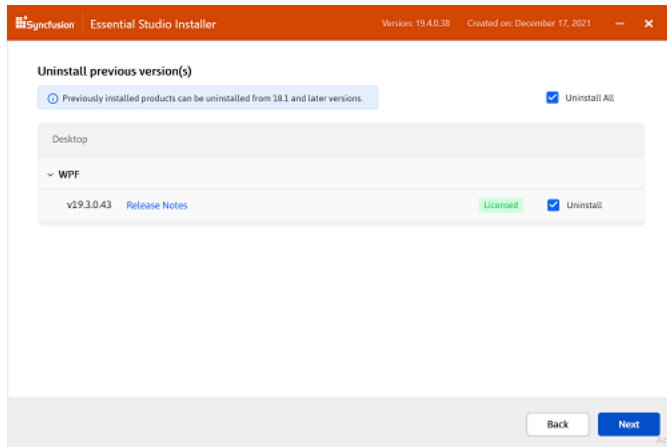


You can also select the products to be installed from the **Available** tab. Click the Next button.

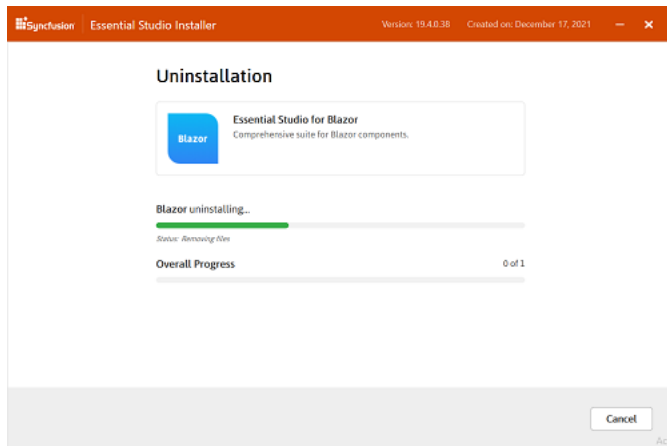
Available



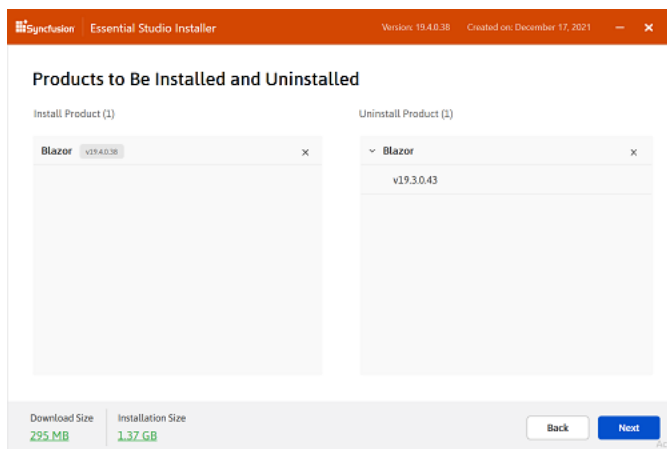
3. If any other products selected for installation, **Uninstall previous version wizard** will be displayed with previous version(s) installed for the selected products. Here you can view the list of installed previous versions for the selected products. Select **Uninstall All** checkbox to select all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



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

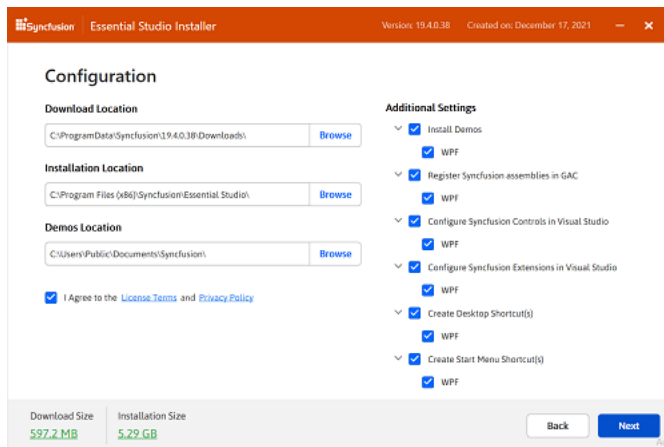


5. The **Confirmation Wizard** will appear with the list of products to be installed/uninstalled. Here you can view and modify the list of products that will be installed/uninstalled.

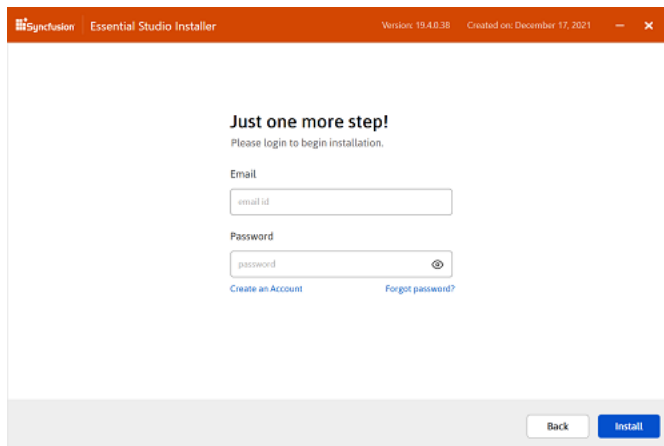


Note By clicking the **Download Size** and **Installation Size** links, you can determine the approximate size of the download and installation.

6. 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.

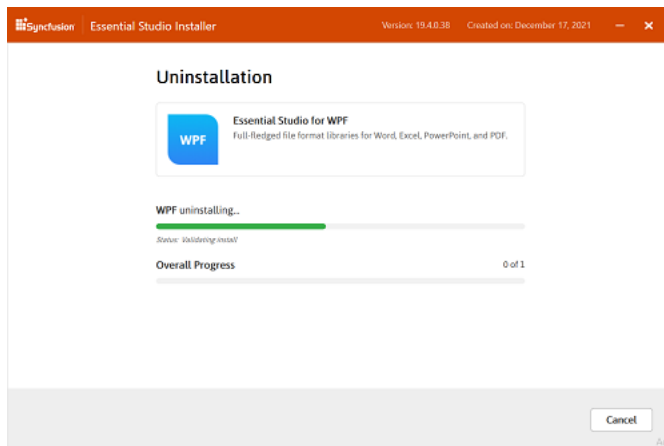


7. After reading the License Terms and Conditions, check the **I agree to the License Terms and Privacy Policy** check box. Click the Next button.
8. The **login wizard** will appear. You must enter your Syncfusion email address and password. If you do not already have a Syncfusion account, you can create 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.

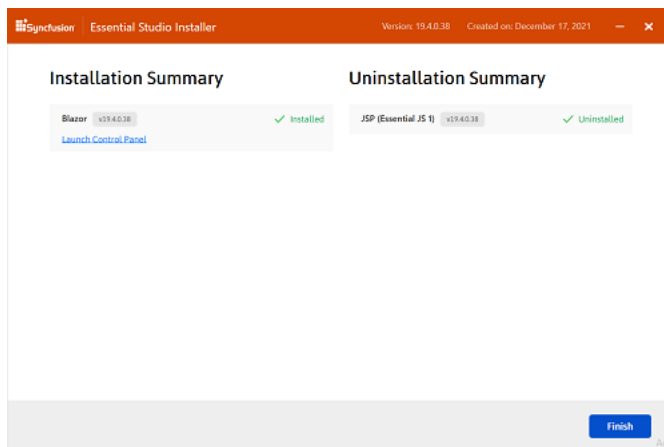


IMPORTANT The products you have chosen will be installed based on your Syncfusion License (Trial or Licensed).

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



10. 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**.

Offline Installer

Downloading Syncfusion Blazor offline installer

The Syncfusion Blazor offline installer can be downloaded from the [Syncfusion](https://www.syncfusion.com/) website. You can either download the licensed installer or try our trial installer depending on your license.

- Trial Installer
- Licensed Installer

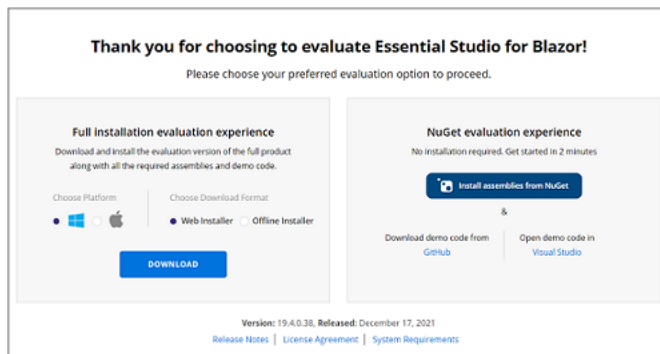
Download the Trial Version

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

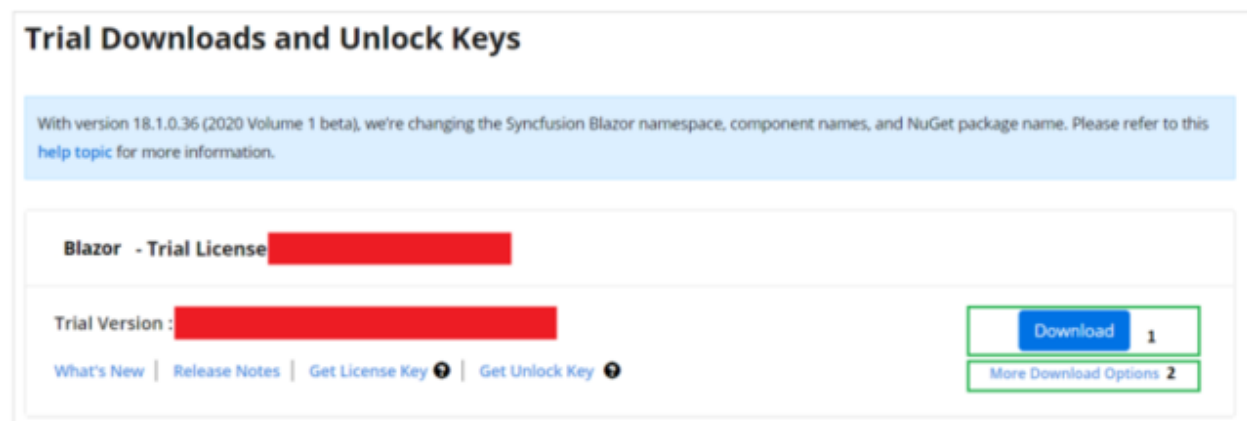
- Download Free Trial Setup
- Start Trials if using components through [NuGet.org](https://www.nuget.org/)

Download Free Trial Setup

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



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

The screenshot shows the 'Downloads' section of the Syncfusion website. At the top right, it indicates 'Version : 19.4.0.38' and 'Released on : Dec 17, 2021'. There are tabs for 'Windows' and 'Mac'. The page is organized into sections for different products:

- Enterprise Studio**: Offers 'Offline Installer' and 'Web Installer'.
 - Enterprise Edition - Binaries & Samples: .ZIP 4349 MB (Checksum Value) and .EXE 3 MB (Checksum Value).
- WEB - Essential JS 2**: Offers 'Offline Installer' and 'Web Installer'.
 - Blazor: .EXE 334 MB (Checksum Value) and .ZIP 334 MB (Checksum Value) are highlighted with a red box. Also offers .EXE 3 MB (Checksum Value) for Web Installer.
 - ASP.NET Core - EJ 2: .EXE 351 MB (Checksum Value) and .ZIP 351 MB (Checksum Value) for Offline Installer, and .EXE 3 MB (Checksum Value) for Web Installer.

Start Trials if using components through [NuGet.org](#)

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

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

The screenshot shows the 'WEB' section of the Syncfusion account page. It lists two products:

- Blazor**: Includes 70+ Blazor components for ASP.NET Core application development. A red box highlights the product name and the 'START TRIAL' button.
- ASP.NET Core**: Includes 70+ HTML5-powered ASP.NET Core controls for developing modern web applications. A 'START TRIAL' button is also visible.

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

Trial Downloads and Unlock Keys

With version 18.1.0.36 (2020 Volume 1 beta), we're changing the Syncfusion Blazor namespace, component names, and NuGet package name. Please refer to this [help topic](#) for more information.

Blazor - Trial License [Redacted]

Trial Version : [Redacted]

[What's New](#) | [Release Notes](#) | [Get License Key](#) ⓘ | [Get Unlock Key](#) ⓘ

[Download](#) **1**

[More Download Options](#) **2**

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 Blazor licensed offline installer by going to More Downloads Options (element 3 in the screenshot below).

License Downloads and Unlock Keys

With version 18.1.0.36 (2020 Volume 1 beta), we're changing the Syncfusion Blazor namespace, component names, and NuGet package name. Please refer to this [help topic](#) for more information.

As part of your Studio/Individual Platform developer license, you also have a Bold Reports Report Viewer SDK license. Use your Syncfusion account to get your Report Viewer SDK license at the Bold Reports website. [CONTINUE](#)

Essential Studio Blazor

Latest Official Release : [Redacted]

[What's New](#) | [Release Notes](#) | [Get License Key](#) ⓘ | [Get Unlock Key](#) ⓘ

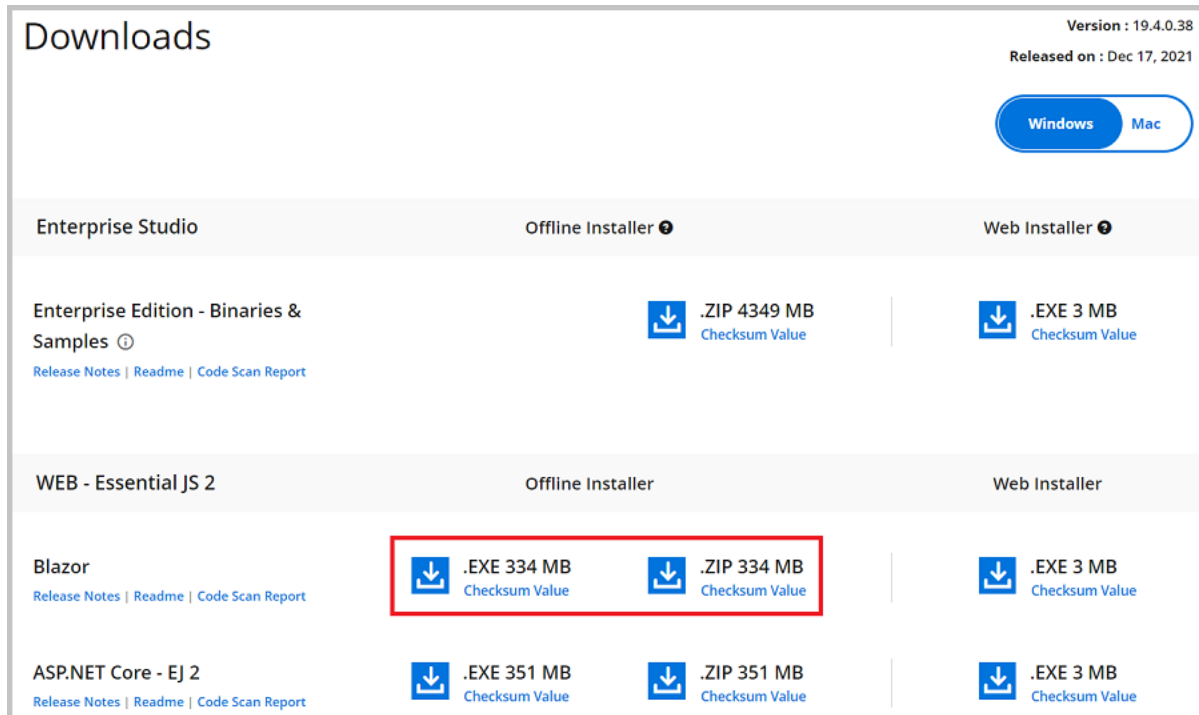
[Download Older Versions](#) **2**

[Download](#) **1**

[More Download Options](#) **3**

Ⓟ Your license expires on November 29, 2021.

4. For Windows OS, EXE and Zip formats are available for download. They are both Offline Installers.



Downloads

Version : 19.4.0.38
Released on : Dec 17, 2021

Windows Mac

Enterprise Studio Offline Installer Web Installer

Enterprise Edition - Binaries & Samples
Release Notes | Readme | Code Scan Report

WEB - Essential JS 2 Offline Installer Web Installer

Blazor
Release Notes | Readme | Code Scan Report

ASP.NET Core - EJ 2
Release Notes | Readme | Code Scan Report

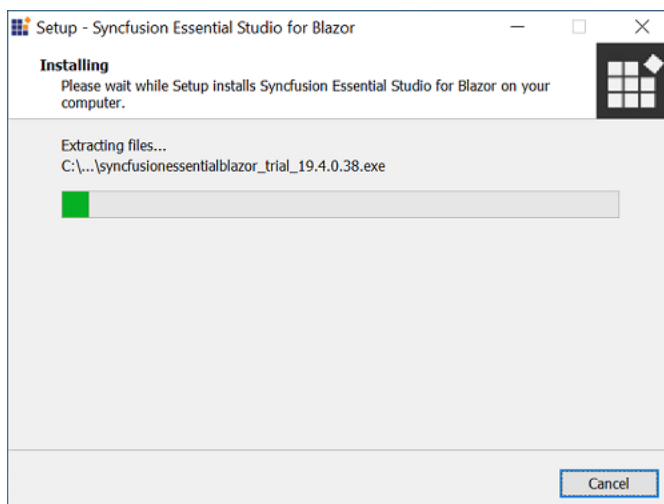
You can also refer to the [Offline installer](#) link for step-by-step installation guidelines.

Installing Syncfusion Blazor offline installer

Installing with UI

The steps below show how to install the Essential Studio Blazor installer.

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



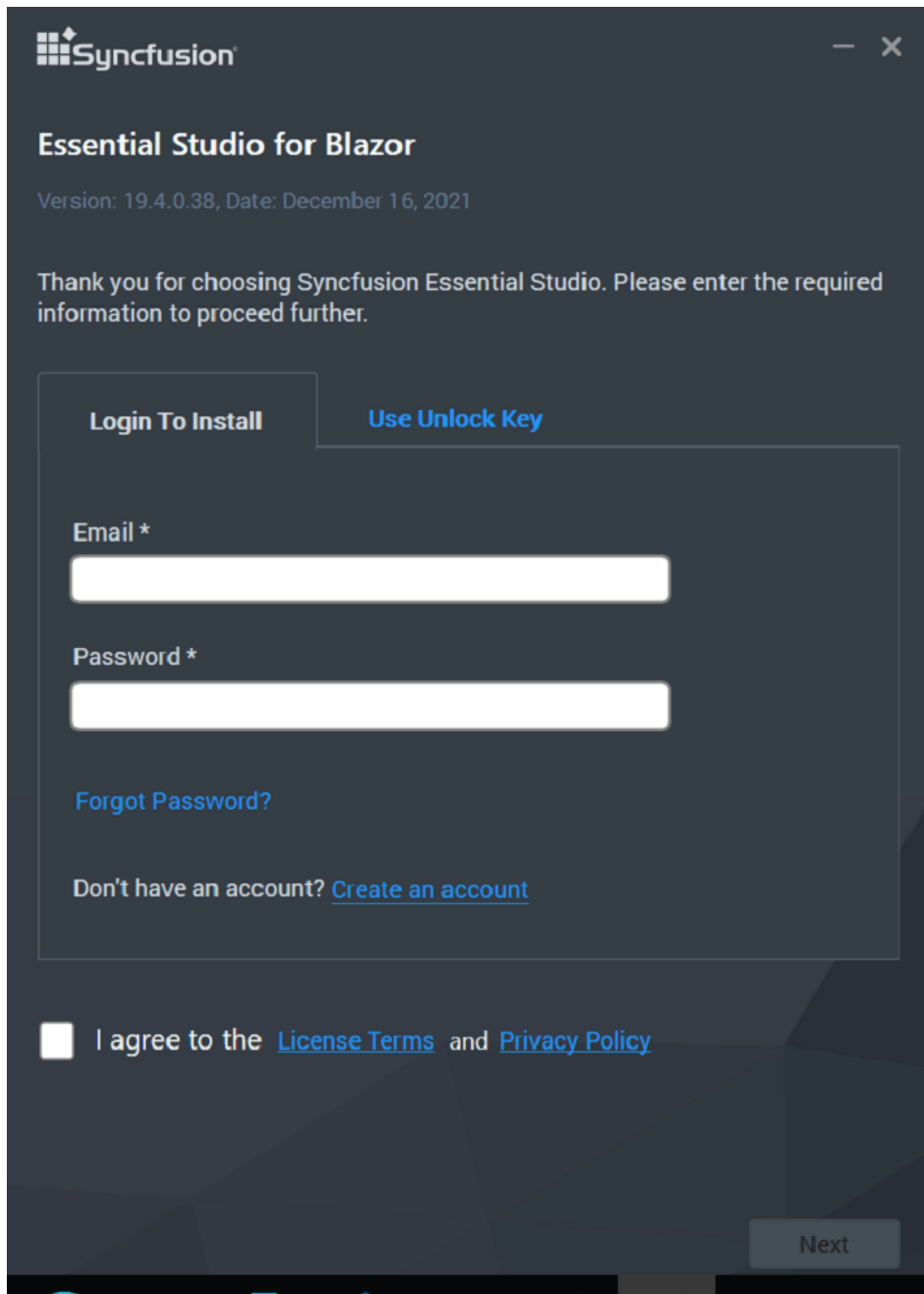
Note The Installer wizard extracts the syncfusionessentialblazor_(version).exe dialog, which displays the package's unzip operation.

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 Blazor

Version: 19.4.0.38, Date: December 16, 2021

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 Blazor 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.

— ✕

Essential Studio for Blazor

Version: 19.4.0.38, Date: December 16, 2021

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

Login To Install

Use Unlock Key

If you do not have an unlock key, [request one from Syncfusion](#).

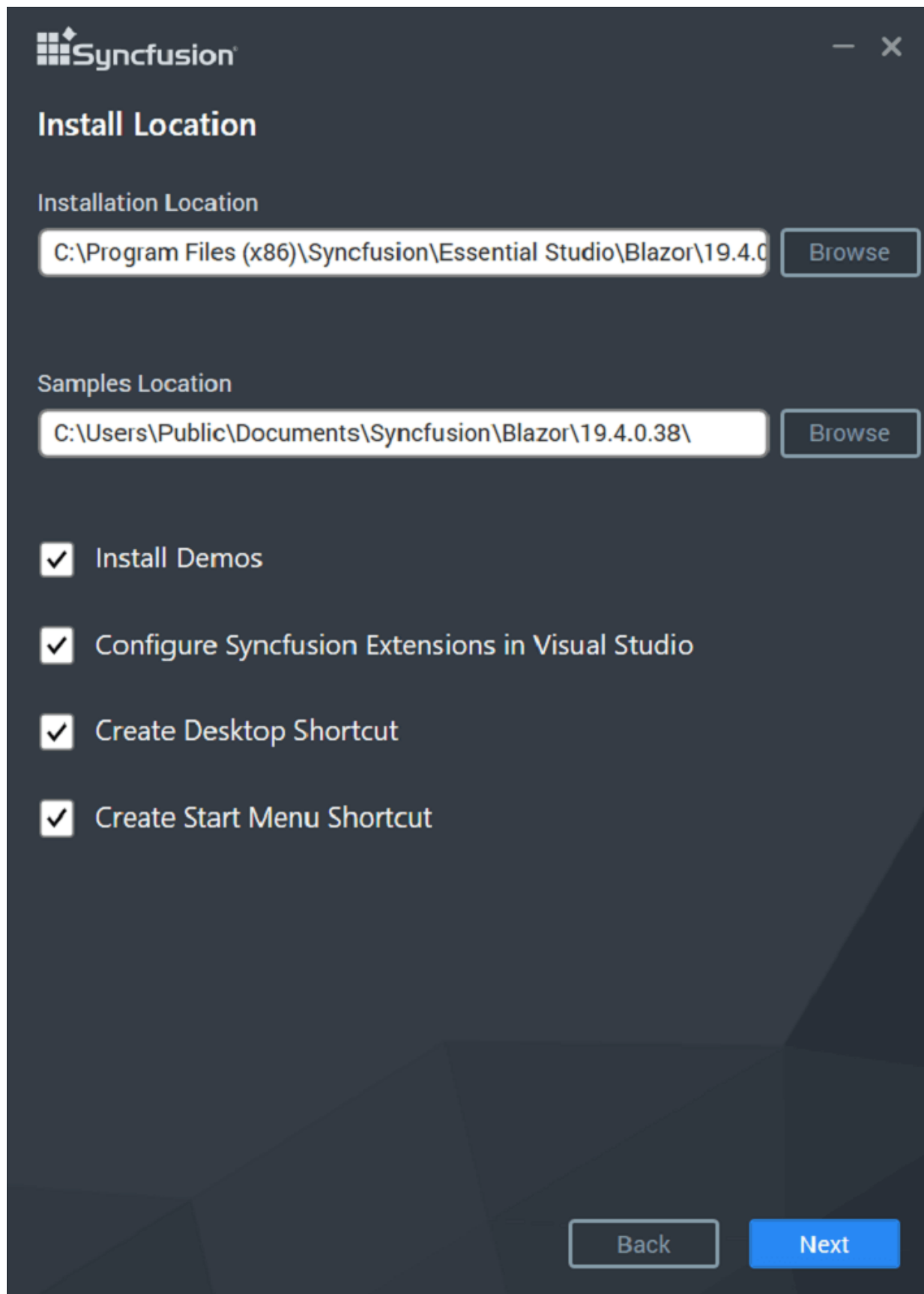
Unlock Key *

Free Evaluation Key

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

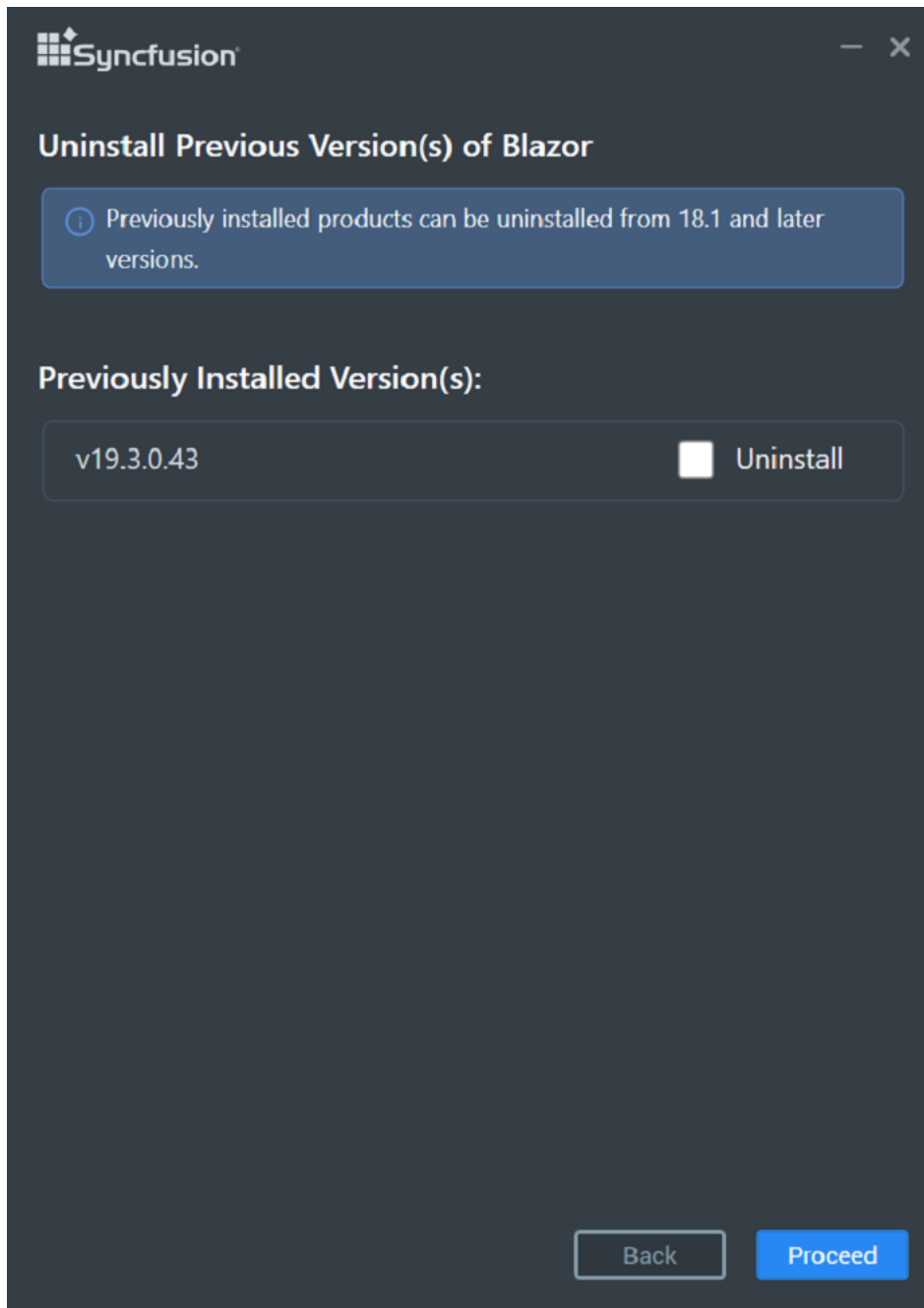
Next

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.



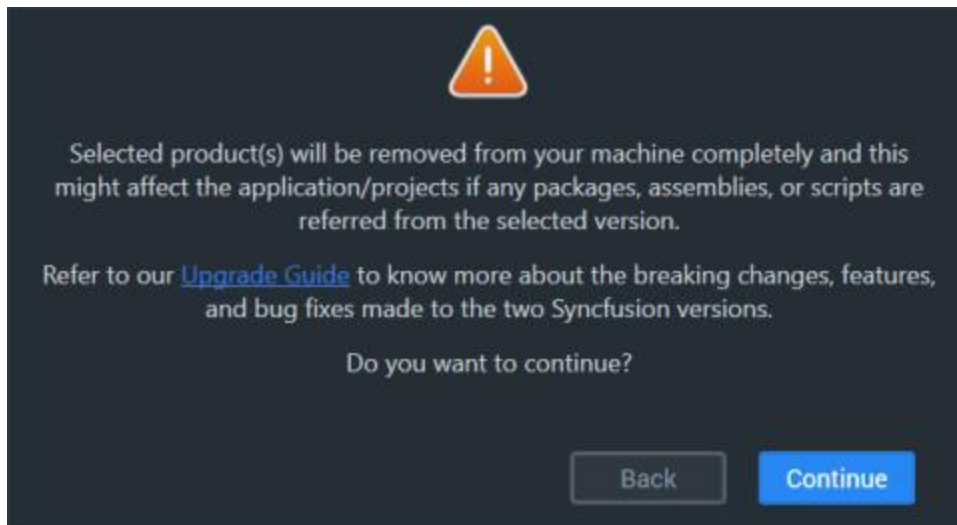
Additional Settings

- 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.

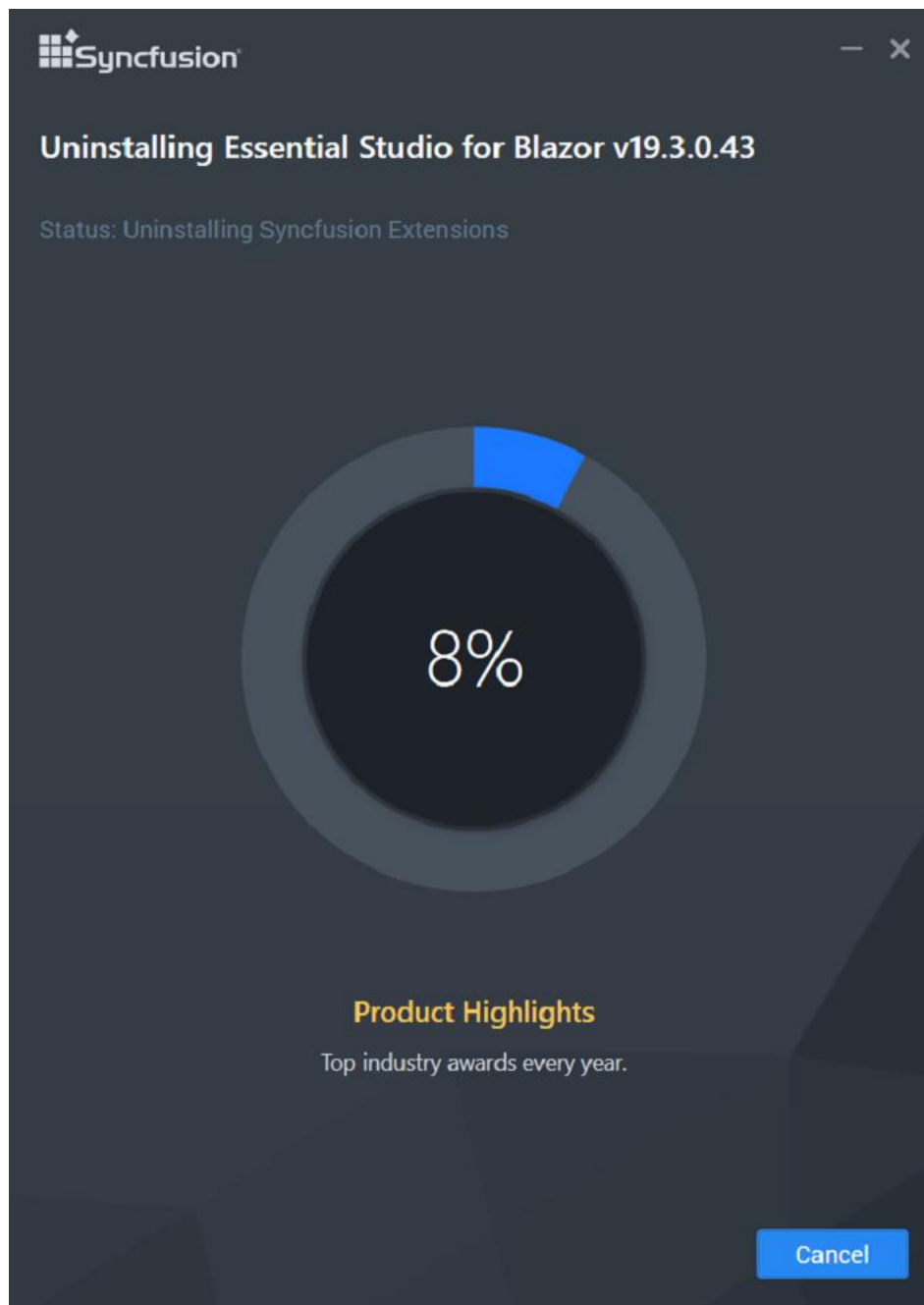


Note From the 2021 Volume 1 release, Syncfusion has added the option to uninstall previous versions from 18.1 while installing the new version. If any version is selected to uninstall, a confirmation screen will appear; if continue is selected, the Progress screen will display the uninstall and install progress, respectively. If none of the versions are chosen to be uninstalled, only the installation progress will be displayed.

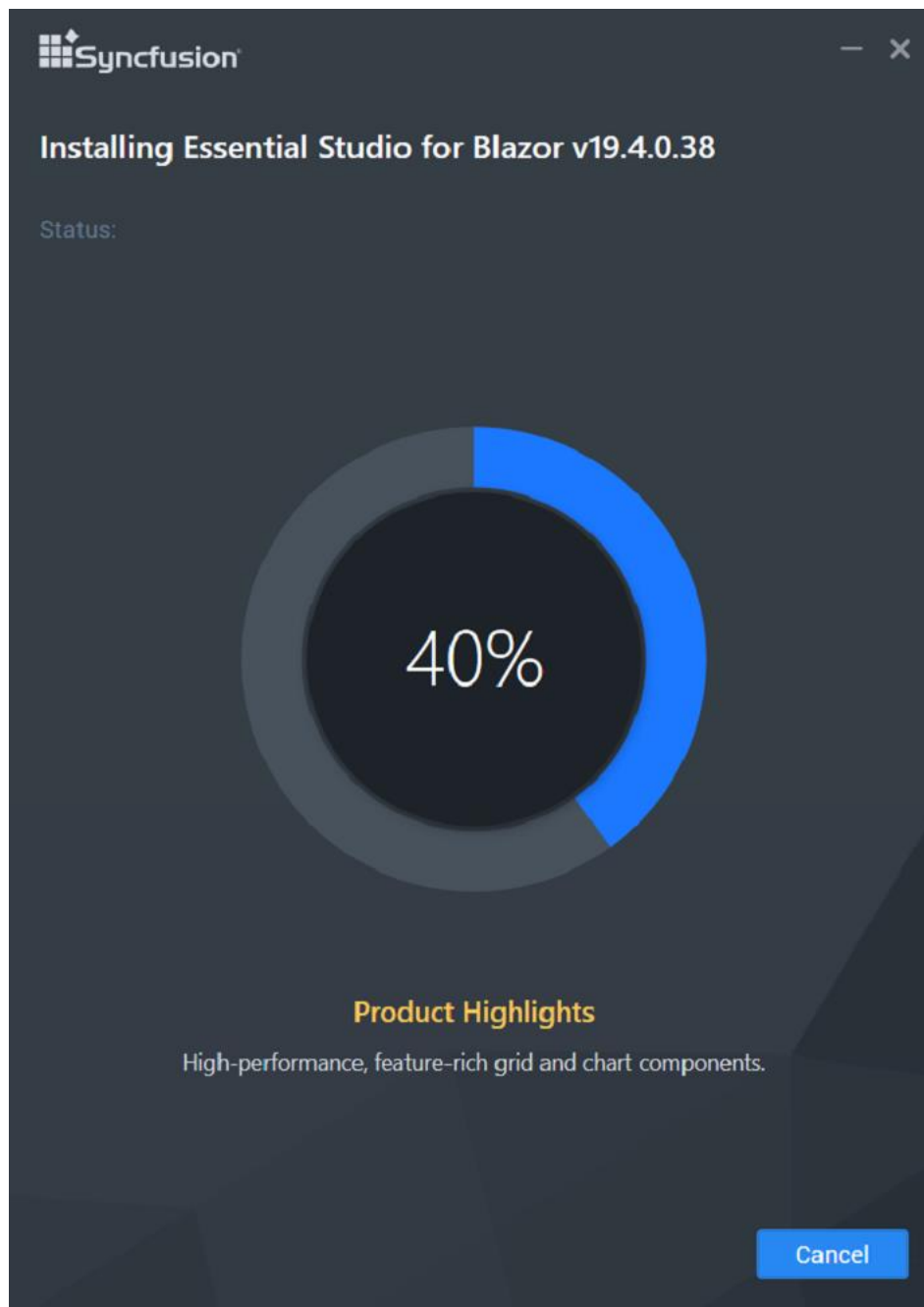
Confirmation Alert:



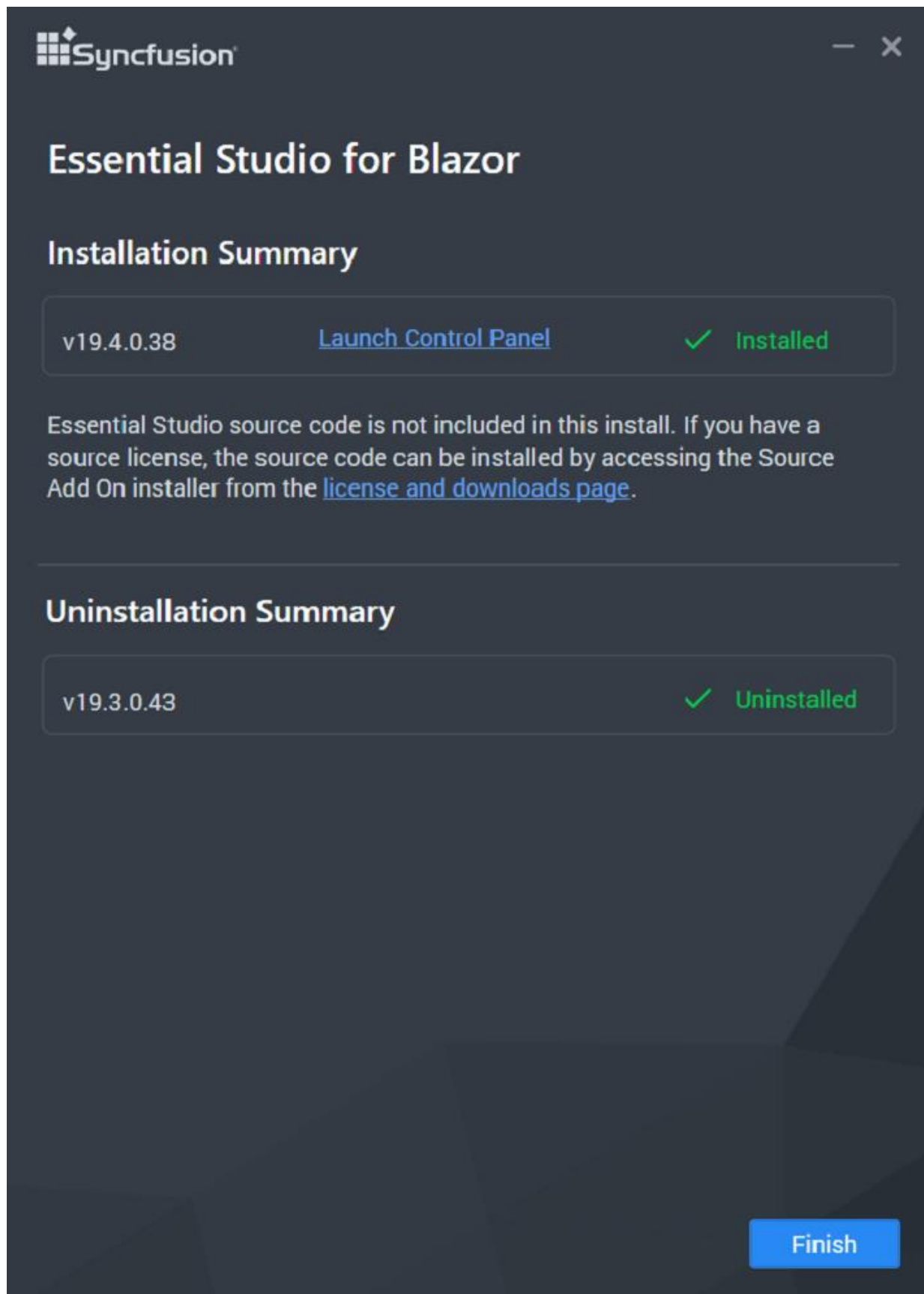
Uninstall Progress:



Install Progress



Note The Completed screen is displayed once the Blazor 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 Blazor product.

Installing in Silent Mode

The Syncfusion Essential Studio Blazor 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 Blazor installer by double-clicking it. The Installer Wizard automatically opens and extracts the package.
2. The file syncfusionessentialblazor_(version).exe file will be extracted into the Temp directory.
3. Run %temp%. The Temp folder will be opened. The syncfusionessentialblazor_(version).exe file will be located in one of the folders.
4. Copy the extracted syncfusionessentialblazor_(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}] [/CreateShortcut:{true/false}] [/ CreateStartMenuShortcut:{true/false}]

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

Example: "D:\Temp\syncfusionessentialblazorx.x.x.x.exe" /Install silent /UNLOCKKEY:"product unlock key" /log "C:\Temp\EssentialStudioProduct.log" /InstallPath:C:\Syncfusion\x.x.x.x /InstallSamples:true /CreateShortcut:true / CreateStartMenuShortcut:true

7. Essential Studio for Blazor 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 Blazor can be uninstalled silently using the Command Line.

1. Run the Syncfusion Blazor installer by double-clicking it. The Installer Wizard automatically opens and extracts the package.
2. The file syncfusionessentialblazor_(version).exe file will be extracted into the Temp directory.
3. Run %temp%. The Temp folder will be opened. The syncfusionessentialblazor_(version).exe file will be located in one of the folders.
4. Copy the extracted syncfusionessentialblazor_(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\syncfusionessentialblazor_(version).exe" /uninstall silent

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

7. Essential Studio for Blazor is uninstalled.

Mac Installer

Downloading Syncfusion Blazor Mac installer

Syncfusion provides Blazor Mac installer for both evaluation and paid customers. You can either download the licensed installer or try our trial installer depending on your license.

- Trial Installer
- Licensed Installer

Download the Trial Version

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

Thank you for choosing to evaluate Essential Studio for Blazor!

Please choose your preferred evaluation option to proceed.

Full installation evaluation experience

Download and install the evaluation version of the full product along with all the required assemblies and demo code.

Choose Platform

☐ Windows ☒ Mac

Choose Download Format

☐ Web Installer ☒ Offline Installer

DOWNLOAD

NuGet evaluation experience

No installation required. Get started in 2 minutes

Install assemblies from NuGet

&

Download demo code from [GitHub](#) | Open demo code in [Visual Studio](#)

Version: 19.4.0.38, Released: December 17, 2021

[Release Notes](#) | [License Agreement](#) | [System Requirements](#)

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 Blazor Mac 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.)

Trial Downloads and Unlock Keys

With version 18.1.0.36 (2020 Volume 1 beta), we're changing the Syncfusion Blazor namespace, component names, and NuGet package name. Please refer to this [help topic](#) for more information.

Blazor - Trial License [Redacted]

Trial Version : [Redacted]

[What's New](#) | [Release Notes](#) | [Get License Key](#) ⓘ | [Get Unlock Key](#) ⓘ

[Download](#) 1

[More Download Options](#) 2

- Click the More Download Options (element 2 in the above screenshot) button to get the Essential Studio Blazor Offline trial installer which is available in PKG format.

Downloads

Version : 19.1.0.54
Released on : Mar 30, 2021

[Windows](#) [Mac](#)

GENERAL INFORMATION

- With version 18.1.0.36 (2020 Volume 1 beta), we're changing the Syncfusion Blazor namespace, component names, and the NuGet package name. Please refer to this [help topic](#) for more information.

WEB - Essential JS 2

Offline installer ⓘ

Blazor Release Notes Readme	Download .PKG 240 MB Checksum Value
JavaScript - EJ 2 ⓘ Release Notes Readme	Download .PKG 994 MB Checksum Value

Start Trials if using components through [NuGet.org](#)

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

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

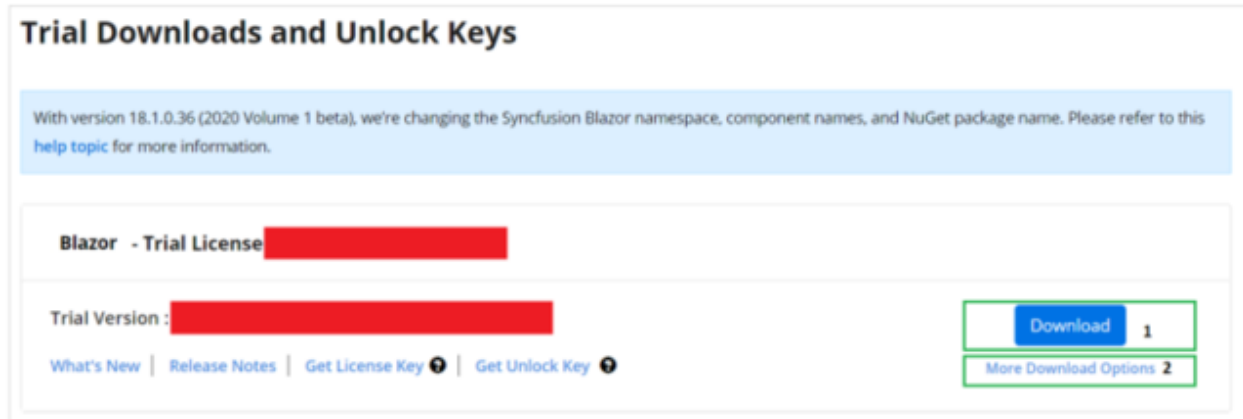
WEB

Blazor Includes 70+ Blazor components for ASP.NET Core application development.	START TRIAL
ASP.NET Core Includes 70+ HTML5-powered ASP.NET Core controls for developing modern web applications.	START TRIAL

2. To access this page, you must sign up\log in with your Syncfusion account.
3. Begin your trial by selecting the Blazor 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 Blazor Mac licensed installer by going to More Downloads Options (element 3 in the screenshot below).

License Downloads and Unlock Keys

With version 18.1.0.36 (2020 Volume 1 beta), we're changing the Syncfusion Blazor namespace, component names, and NuGet package name. Please refer to this [help topic](#) for more information.

As part of your Studio/Individual Platform developer license, you also have a Bold Reports Report Viewer SDK license. Use your Syncfusion account to get your Report Viewer SDK license at the Bold Reports website. [CONTINUE](#)

Essential Studio Blazor

Latest Official Release: [REDACTED]

[What's New](#) | [Release Notes](#) | [Get License Key](#) | [Get Unlock Key](#)

🔒 Your license expires on November 29, 2021.

[Download Older Versions](#) 2

[Download](#) 1

[More Download Options](#) 3

4. Unlock key is not required to install the Syncfusion Blazor Mac trial installer.
5. For Mac OS, PKG formats is available for download.

Downloads

Version : 19.1.0.54
Released on : Mar 30, 2021

[Windows](#) [Mac](#)

GENERAL INFORMATION

- With version 18.1.0.36 (2020 Volume 1 beta), we're changing the Syncfusion Blazor namespace, component names, and the NuGet package name. Please refer to this [help topic](#) for more information.

WEB - Essential JS 2 [Offline Installer](#)

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

[Download](#) .PKG 240 MB
[Checksum Value](#)

JavaScript - EJ 2
[Release Notes](#) | [Readme](#)

[Download](#) .PKG 994 MB
[Checksum Value](#)

You can also refer to the [Blazor Mac installation](#) link for step-by-step installation guidelines.

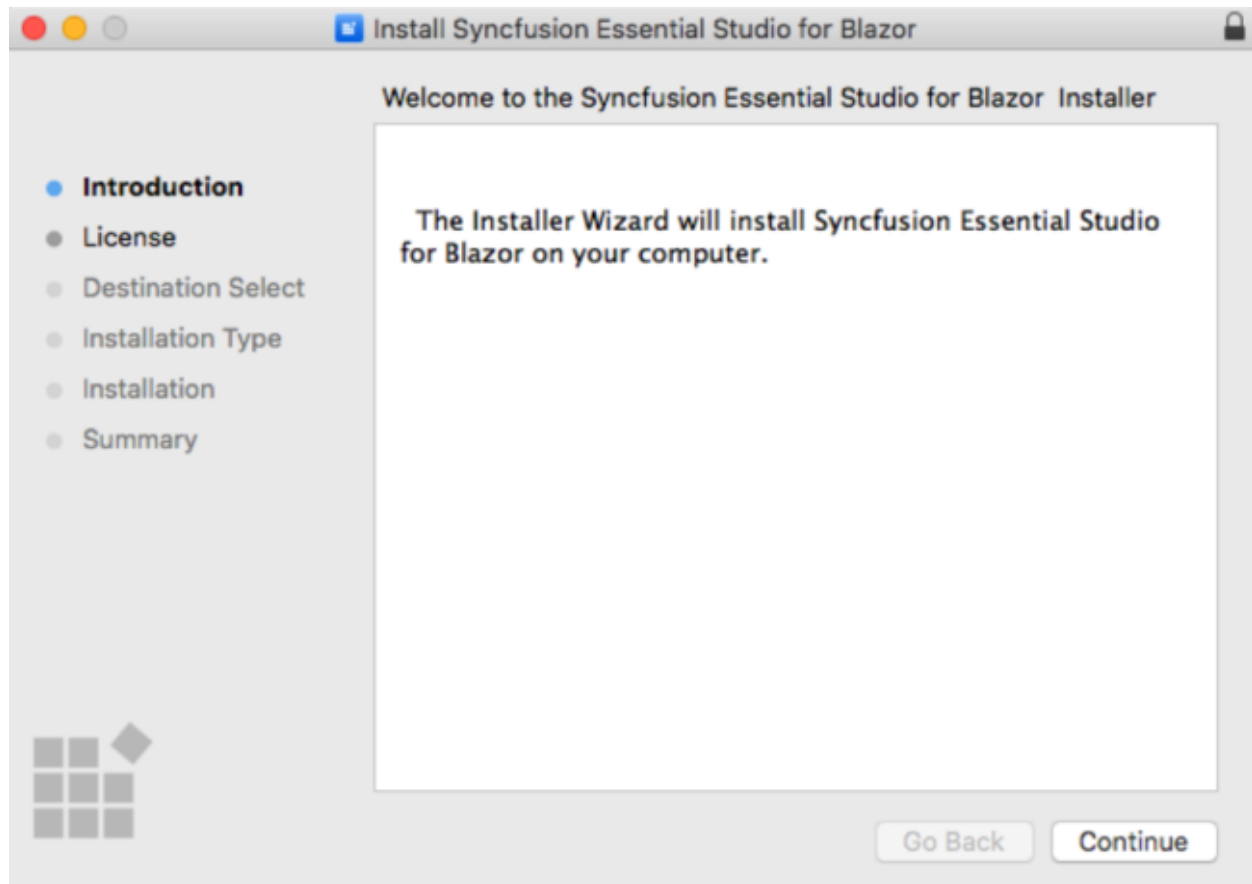
Installing Syncfusion Blazor Mac installer

The Syncfusion Blazor Mac installer allows you to create the Blazor application in Visual Studio for Mac with the Syncfusion Blazor components.

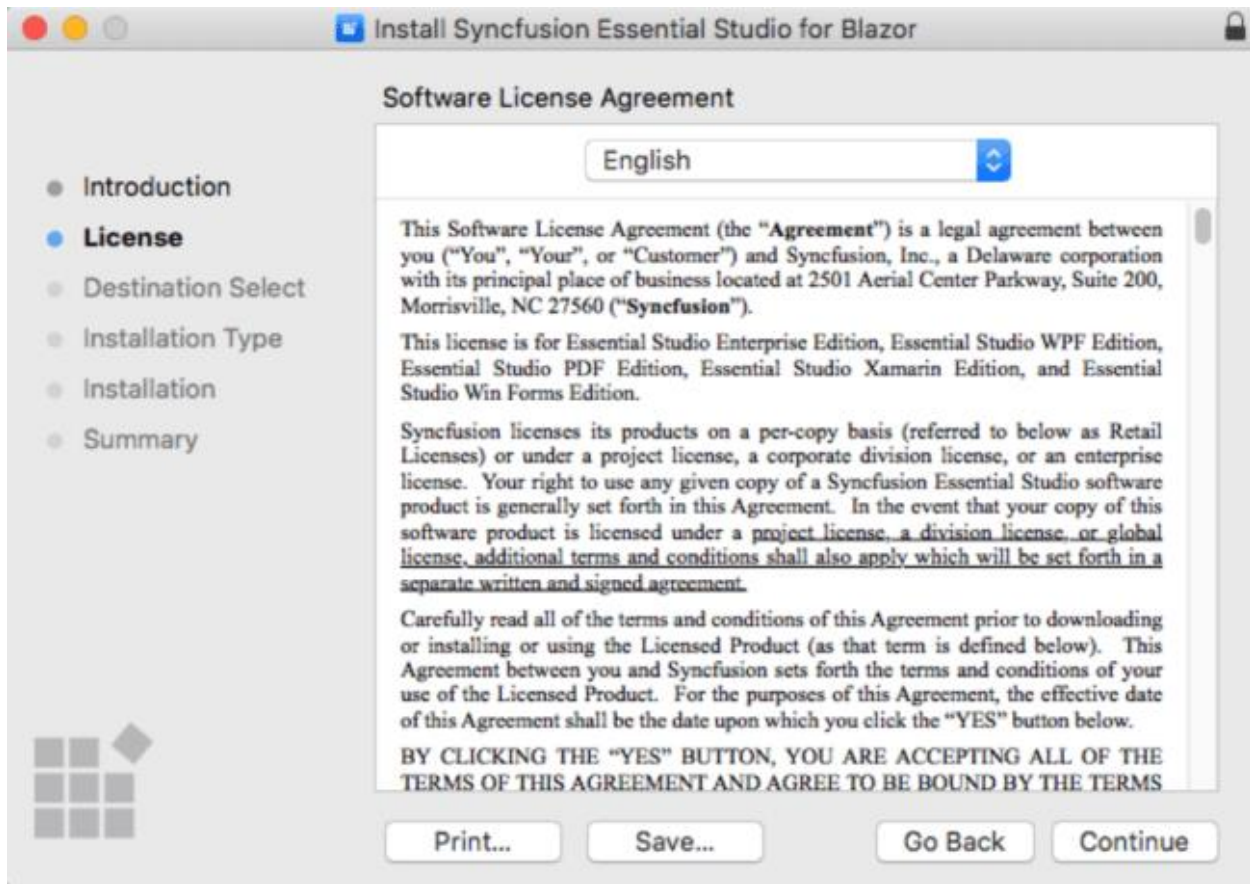
Installation

The steps below show how to install the Blazor Mac installer.

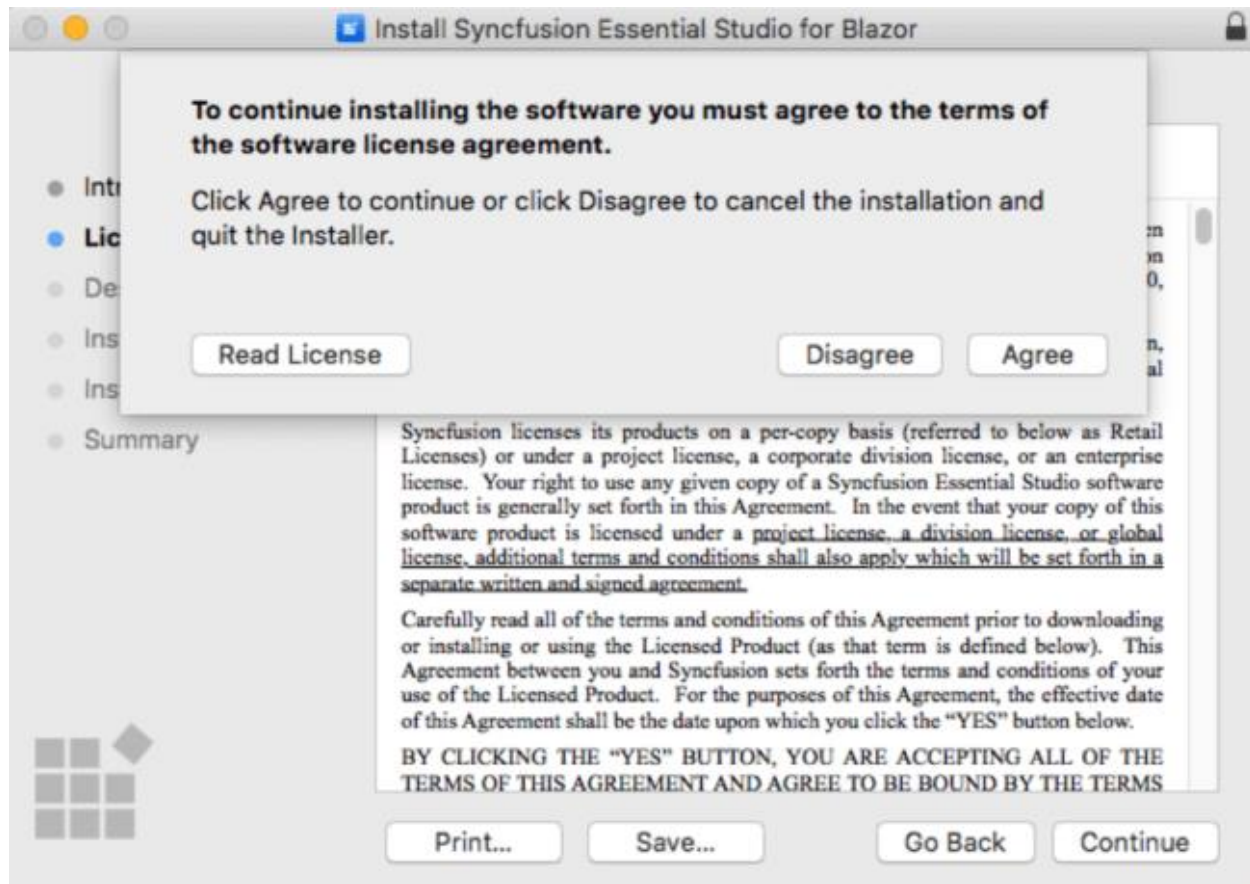
1. Open Syncfusion Blazor Mac Installer(.pkg) file. The Installer Wizard opens. Click Continue.



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

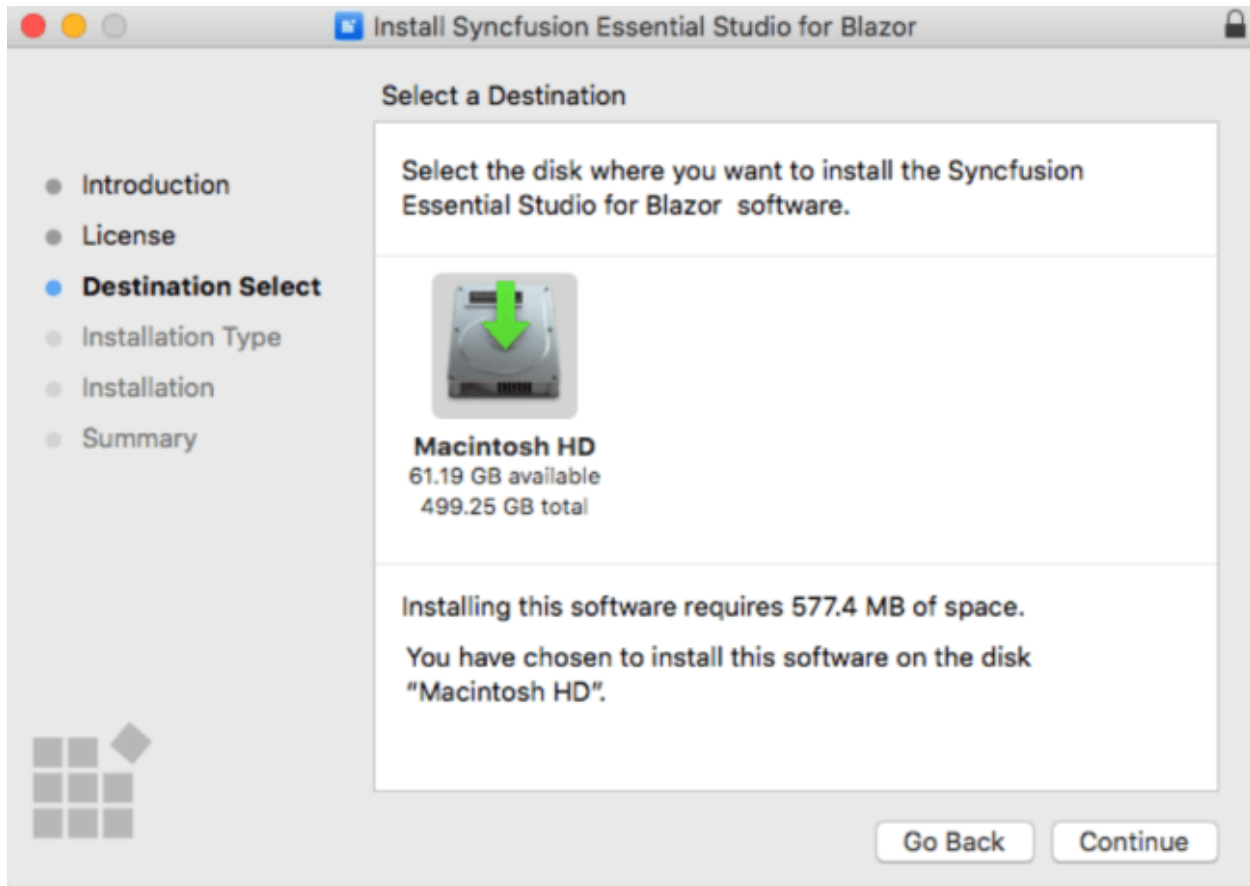


3. The License Agreement's Confirmation window will appear. If you have read the Software License Agreement, click Agree.

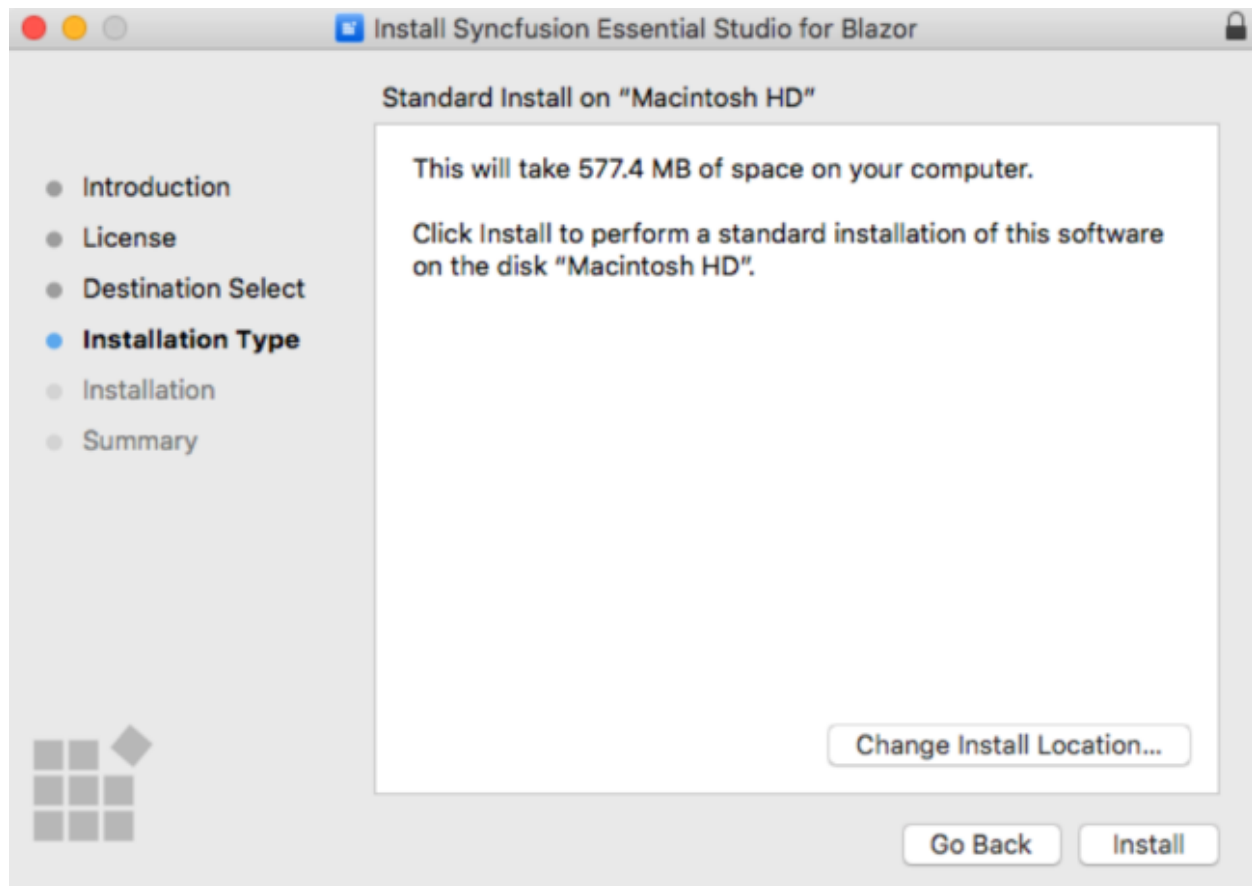


Note
 The Unlock key is not required to install the Mac installer.

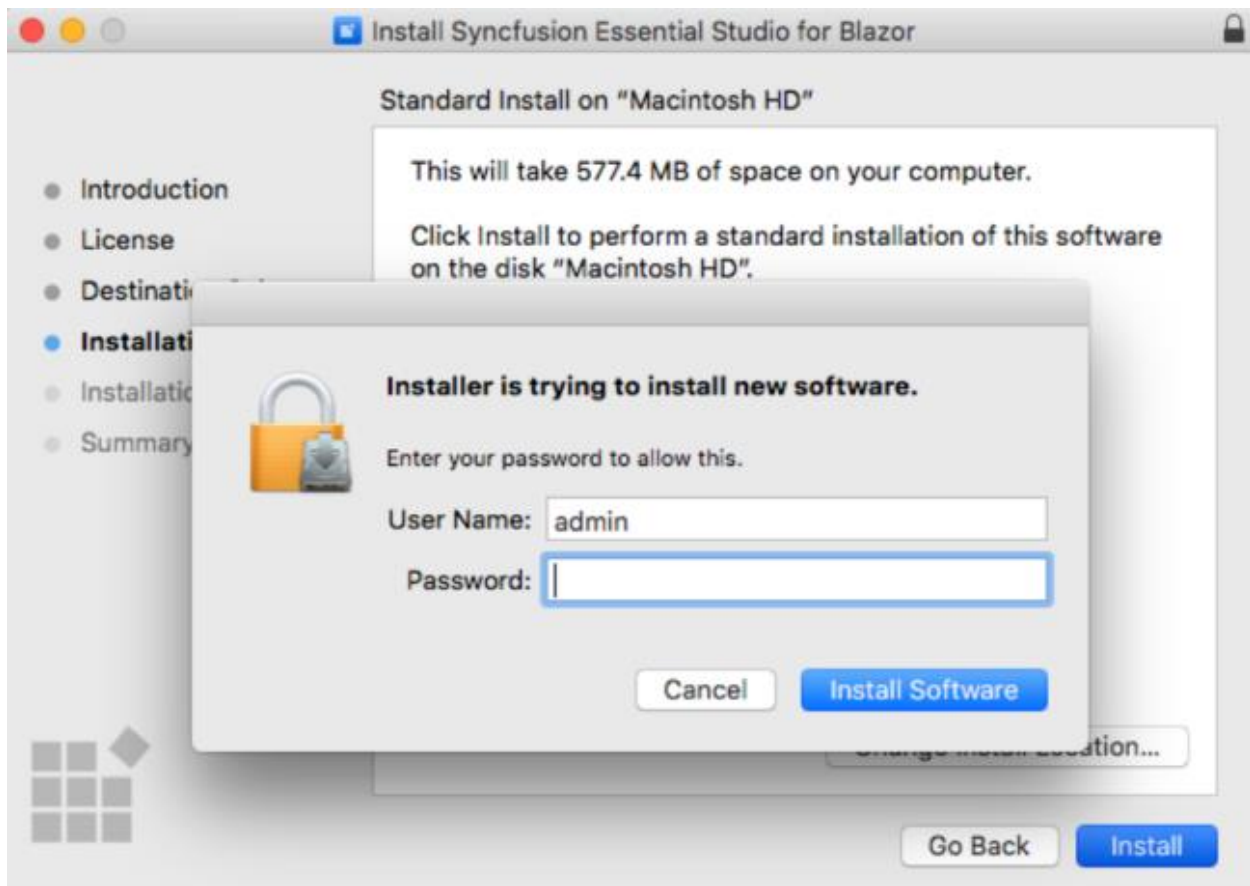
4. The Destination select wizard will appear. You can choose which disc to install the Syncfusion Essential Studio for Blazor installer on here.



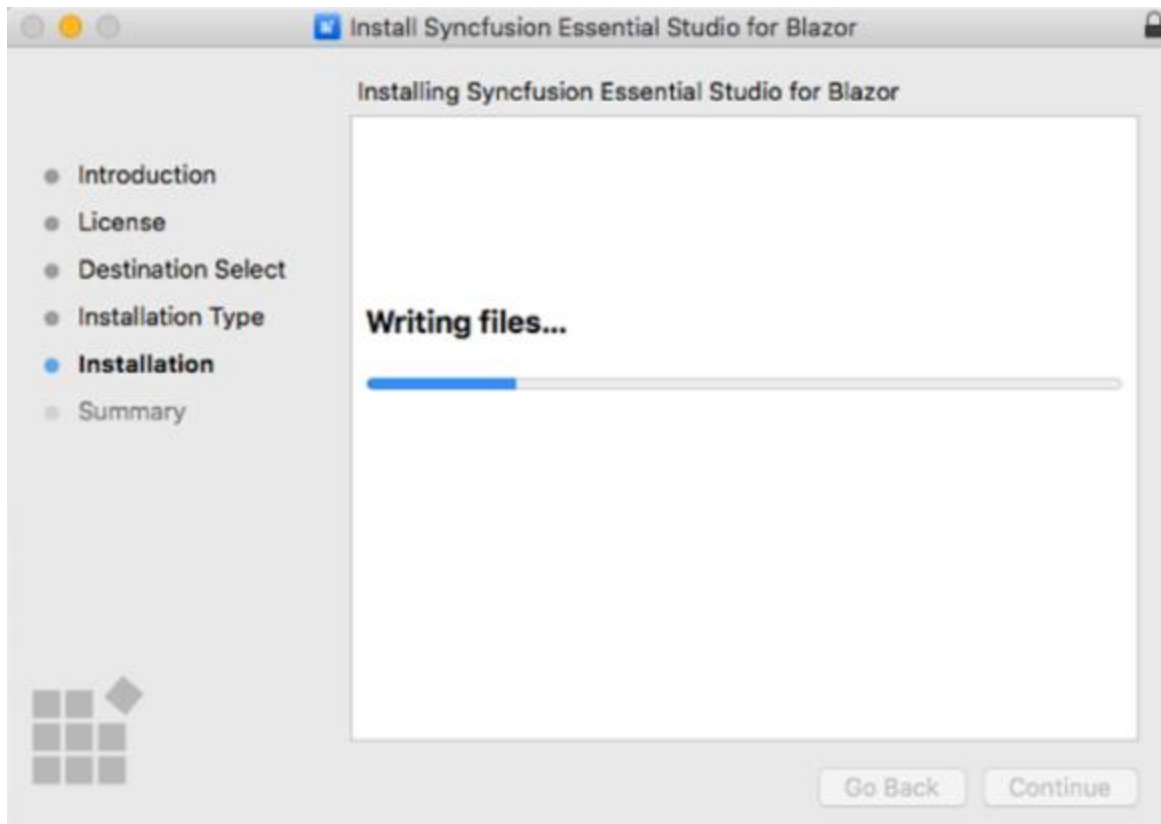
5. The Installation Type wizard will appear. Click Install to begin the standard installation of the Syncfusion Blazor Mac installer.



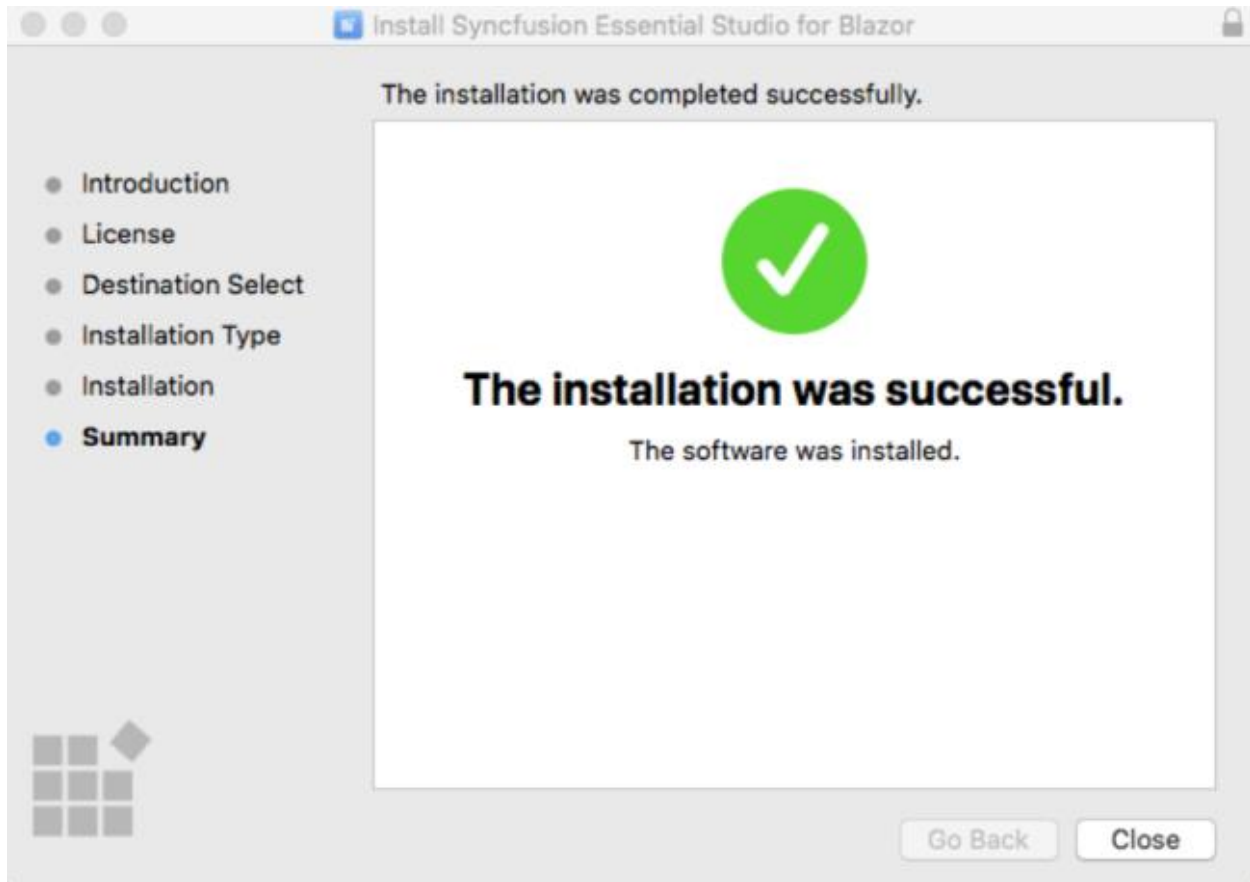
6. The Authentication window will appear. To begin the installation, enter the Mac machine's password and click Install Software.



7. The installation process will begin on your machine.

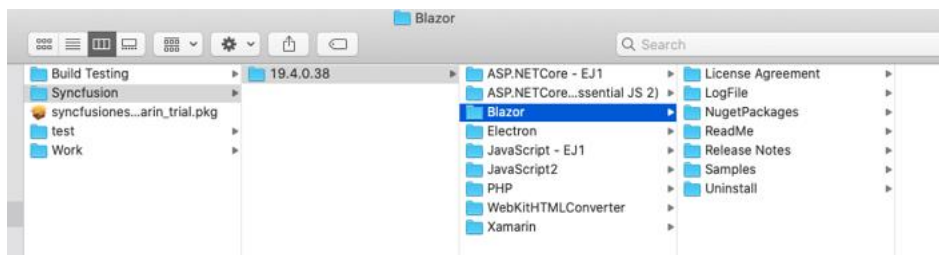


8. Once the installation is complete, the completed screen will be displayed. To exit the installation wizard, click Close.



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

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



You can also find the steps for getting started with Syncfusion Blazor components in Mac,

- Getting started with Syncfusion Blazor Components in [Blazor Server Side App](#) using Visual Studio for Mac
- Getting started with Syncfusion Blazor Components in [Web Assembly App](#) using Visual Studio for Mac

Install Syncfusion Blazor NuGet packages

Overview

NuGet is a Package management system for Visual Studio. It makes it easy to add, update and remove external libraries in our application. Syncfusion publishing all Blazor NuGet packages in nuget.org. The Syncfusion Blazor NuGet packages can be used without installing the Syncfusion installation. You can

simply exploit the Syncfusion Blazor NuGet packages in your Blazor application to develop with the Syncfusion Blazor components.

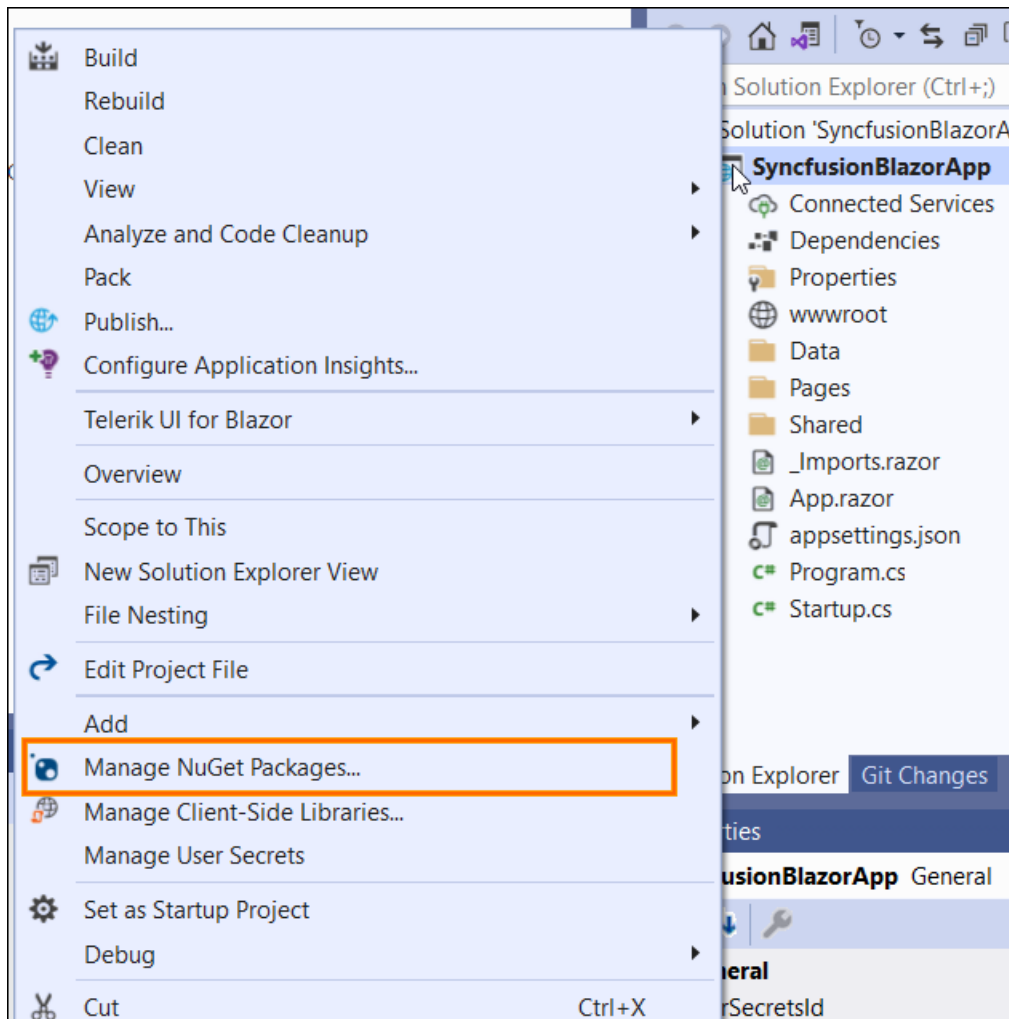
The Syncfusion Blazor NuGet package, which contains all Syncfusion Blazor components in a single package, is available beginning with v17.4.0.39 (Essential Studio 2019 Volume 4).

The Syncfusion Blazor UI components are available separately as [Individual NuGet packages](#) beginning with v18.4.0.30 (Essential Studio 2020 Volume 4). The NuGet packages are segregated based on the component usage and its namespace.

Installation using Package Manager UI

The NuGet **Package Manager UI** allows you to search, install, uninstall, and update Syncfusion Blazor NuGet packages in your applications and solutions. You can find and install the Syncfusion Blazor NuGet packages in your Visual Studio Blazor application and this process is easy with the steps below:

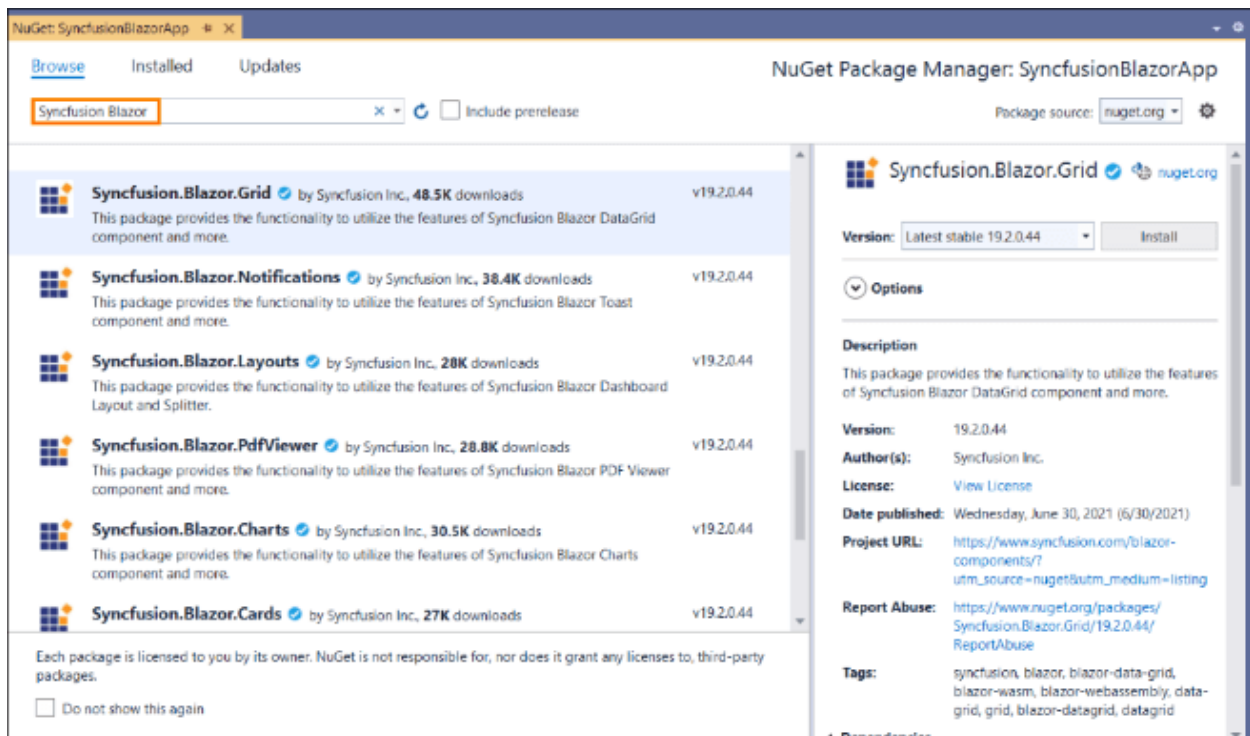
1. Right-click on the Blazor application or solution in the Solution Explorer, and choose **Manage NuGet Packages...**



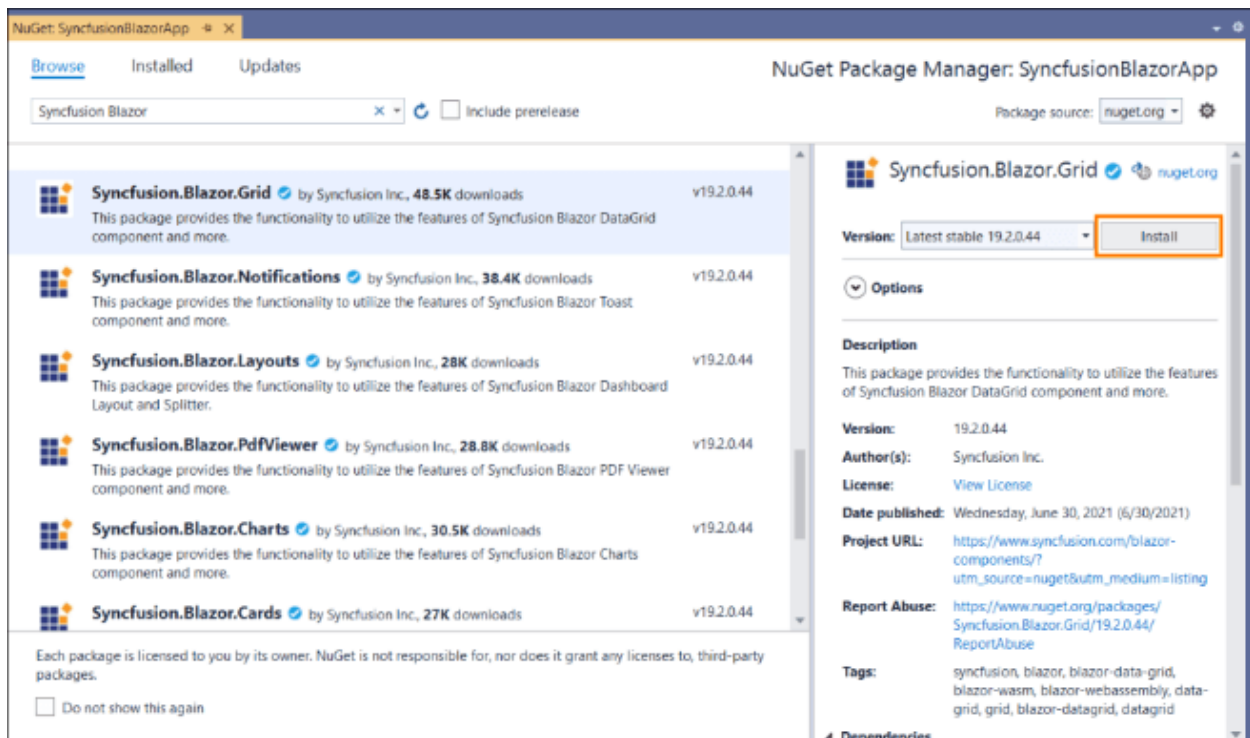
As an alternative, after opening the Blazor application in Visual Studio, go to the **Tools** menu and after hovering **NuGet Package Manager**, select **Manage NuGet Packages for Solution...**

- The Manage NuGet Packages window will open. Navigate to the **Browse** tab, then search for the Syncfusion Blazor NuGet packages using a term like **"Syncfusion Blazor"** and select the appropriate Syncfusion Blazor NuGet package for your development.

The [nuget.org](https://www.nuget.org) package source is selected by default in the Package source drop-down. If your Visual Studio does not have nuget.org configured, follow the instructions in the [Microsoft documents](#) to set up the nuget.org feed URL.



- When you select a Blazor package, the right side panel will provide more information about it.
- By default, the package selected with latest version. You can choose the required version and click the **Install** button and accept the license terms. The package will be added to your Blazor application.



- At this point, your application has all the required Syncfusion assemblies, and you will be ready to start building high-performance, responsive app with [Syncfusion Blazor components](#). Also, you can refer to the [Blazor help document](#) for development.

Installation using Dotnet (.NET) CLI

The [dotnet Command Line Interface \(CLI\)](#), allows you to add, restore, pack, publish, and manage packages without making any changes to your application files. [Dotnet add package](#) adds a package reference to the application file, then runs [dotnet restore](#) to install the package.

Follow the below instructions to use the dotnet CLI command to install the Syncfusion Blazor NuGet packages.

- Open a command prompt and navigate to the directory where your Blazor application file is located.
- To install a NuGet package, run the following command.

DOTNET ADD PACKAGE <PACKAGE NAME>CODESNIPPET

INSTALL-PACKAGE <PACKAGE NAME>CODESNIPPET

INSTALL-PACKAGE <PACKAGE NAME> - PROJECTNAME <PROJECT NAME>CODESNIPPET

INSTALL-PACKAGE SYNCFUSION.BLAZOR.GRID -VERSION 19.2.0.44CODESNIPPET

Upgrade

DOTNET ADD PACKAGE SYNCFUSION.BLAZOR.GRID -V 19.2.0.44.CODESNIPPET

UPDATE-PACKAGE <PACKAGE NAME>CODESNIPPET

UPDATE-PACKAGE <PACKAGE NAME> -PROJECTNAME <PROJECT NAME>CODESNIPPET

UPDATE-PACKAGE SYNCFUSION.BLAZOR.GRID -VERSION 19.2.0.44CODESNIPPET

Visual Studio Code Integration

Visual Studio Code Extension

Visual Studio Integration

VS 2019 Extension

DOTNET TOOL INSTALL -G SYNCFUSION.SCAFFOLDINGCODESNIPPET

DOTNET TOOL UPDATE -G SYNCFUSION.SCAFFOLDINGCODESNIPPET

SYNCFUSION SCAFFOLD {CONTROLNAME} --PROJECT "{PROJECTFILENAMEWITHPATH}" --MODEL {MODEL} -DC {DBCONTEXT} -CNAME {CONTROLLERNAME} -VNAME {VIEWNAME} [CONTROLMANTORYPARAMETER] [CONTROLMANTATORYPARAMETERVALUE]CODESNIPPET

Accordion

HTML

```
<head>
...
...
<link href="_content/Syncfusion.Blazor.Themes/bootstrap4.css"
rel="stylesheet" />
</head>
```

Warning: Syncfusion.Blazor package should not be installed along with the [individual NuGet packages](#). Hence, add the above [Syncfusion.Blazor.Themes](#) static web assets (styles) in the application.

Using Syncfusion.Blazor NuGet Package [Old standard]

Warning: If the above new standard ([individual NuGet packages](#)) is preferred, then skip this section. Using both the old and new standards in the same application will throw ambiguous compilation errors.

1. Install **Syncfusion.Blazor** NuGet package to the newly created application by using the **NuGet Package Manager**.
2. Add the client-side style resources through [CDN](#) or from [NuGet](#) package in the `element` of the `~/Pages/_Host.cshtml` page.

HTML

```
<head>
....
....
<link href="_content/Syncfusion.Blazor/styles/bootstrap4.css"
rel="stylesheet" />
@*<link href="https://cdn.syncfusion.com/blazor/{{ site.blazorversion
}}/styles/bootstrap4.css" rel="stylesheet" />*@
</head>
```

For Internet Explorer 11 kindly refer the polyfills. Refer the [documentation](#) for more information.

HTML

```
<head>
...
<link href="https://cdn.syncfusion.com/blazor/{{ site.blazorversion
}}/styles/bootstrap4.css" rel="stylesheet" />
<script
src="https://github.com/Daddoon/Blazor.Polyfill/releases/download/3.0.1/blaz
or.polyfill.min.js"></script>
...
</head>
```

Adding component package to the application

Open the `~/_Imports.razor` file and import the **Syncfusion.Blazor.Navigations** package.

ASPX-CS

```
@using Syncfusion.Blazor.Navigations
```

Add SyncfusionBlazor service in Startup file

Open the **Startup.cs** file and add services required by the Syncfusion components using the **services.AddSyncfusionBlazor()** method. Add this method in the **ConfigureServices** function as follows.

C#

```
using Syncfusion.Blazor;
namespace BlazorApplication
{
    public class Startup
    {
        ....
    }
}
```

```

.....
public void ConfigureServices(IServiceCollection services)
{
    .....
    .....
    services.AddSyncfusionBlazor();
}
}
}

```

Adding Accordion component to the application

Now, add the Syncfusion Blazor Accordion component in any web page (razor) in the **Pages** folder. For example, the Accordion component is added in the `~/Pages/Index.razor` page.

ASPX-CS

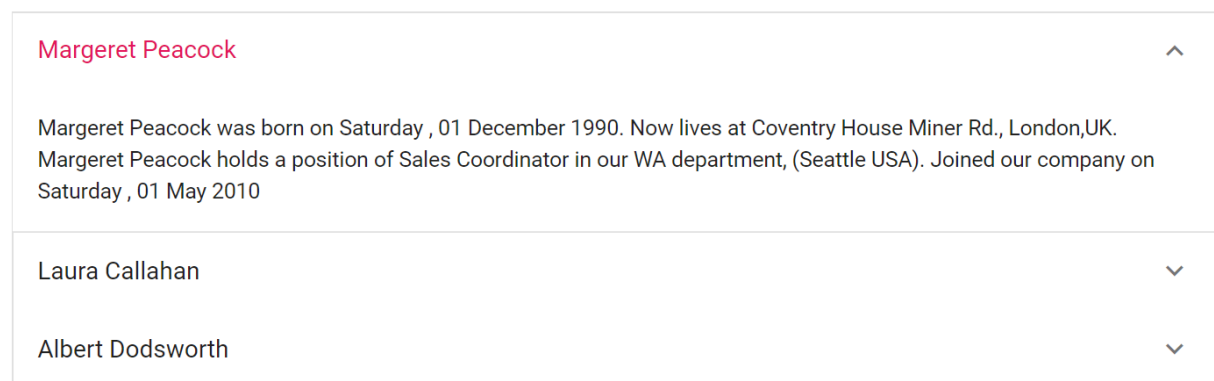
```

@using Syncfusion.Blazor.Navigations
<SfAccordion>
<AccordionItems>
<AccordionItem Header="Margeret Peacock" Content="Margeret Peacock was born
on Saturday , 01 December 1990. Now lives at Coventry House Miner Rd.,
London,UK. Margeret Peacock holds a position of Sales Coordinator in our WA
department, (Seattle USA). Joined our company on Saturday , 01 May
2010"></AccordionItem>
<AccordionItem Header="Laura Callahan" Content="Laura Callahan was born on
Tuesday , 06 November 1990. Now lives at Edgeham Hollow Winchester Way,
London,UK. Laura Callahan holds a position of Sales Coordinator in our WA
department, (Seattle USA). Joined our company on Saturday , 01 May
2010"></AccordionItem>
<AccordionItem Header="Albert Dodsworth" Content="Albert Dodsworth was born
on Thursday , 19 October 1989. Now lives at 4726 - 11th Ave. N.E.,
Seattle,USA.Albert Dodsworth holds a position of Sales Representative in our
WA department, (Seattle USA). Joined our company on Friday , 01 May
2009"></AccordionItem>
</AccordionItems>
</SfAccordion>

```

Run the application

After successful compilation of the application, simply press **F5** to run the application.

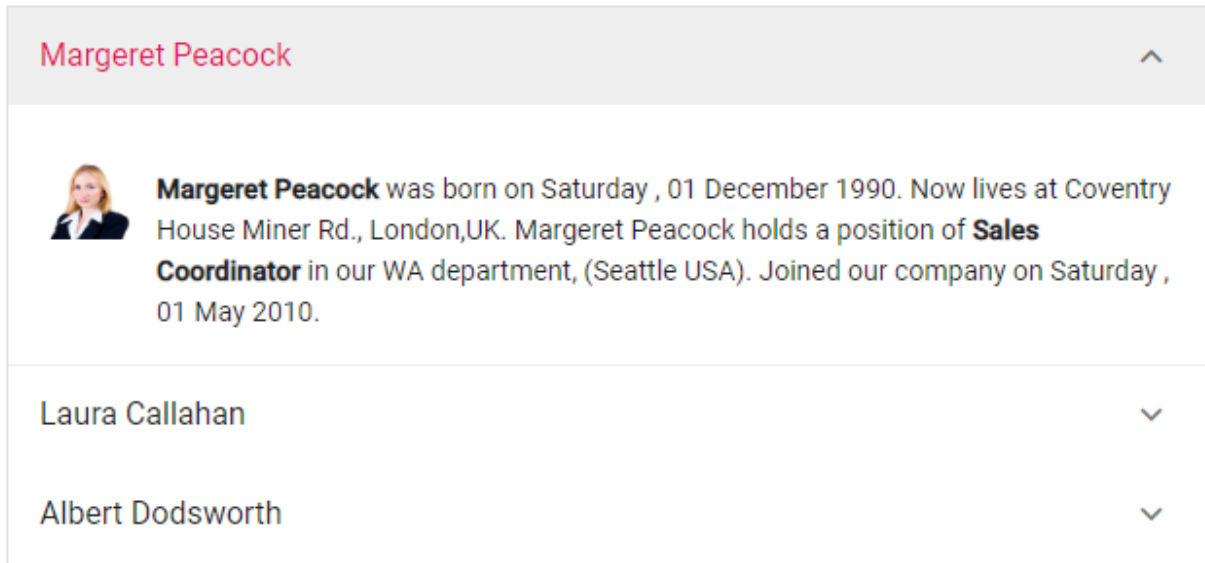


Initialize Accordion using Template

The following code explains how to initialize accordion using **Template**.

ASPX-CS

```
{% include_relative code-snippet/getting-started.razor %}
```



See Also

1. [Getting Started with Syncfusion Blazor for client-side in .NET Core CLI](#)
2. [Getting Started with Syncfusion Blazor for client-side in Visual Studio 2019](#)
3. [Getting Started with Syncfusion Blazor for server-side in .NET Core CLI](#)

Data binding in Blazor Accordion Component

Accordion component provides an option to get the accordion items from the local data. It can be done through iteration of the Accordion Items using conditional **foreach** loop. Accordion only supports the local data for data binding. The [HeaderTemplate](#) and [ContentTemplate](#) properties can be used to render the accordion header and content respectively.

The following sample explains how to initialize accordion items through **templates**.

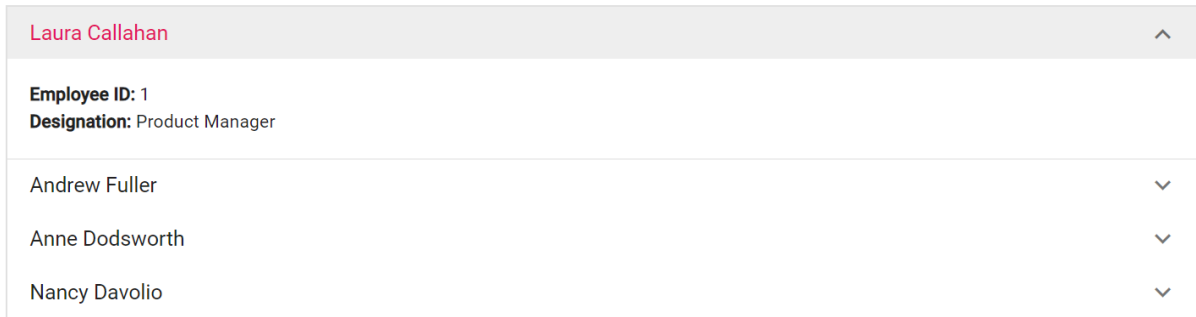
ASPX-CS

```
@using Syncfusion.Blazor.Navigations
<SfAccordion>
<AccordionItems>
@foreach (AccordionData Item in AccordionItems)
{
<AccordionItem>
<HeaderTemplate>
<div>@(Item.EmployeeName) </div>
</HeaderTemplate>
<ContentTemplate>
```



```
<div>
<div><b>Employee ID: </b>@Item.EmployeeId</div>
<div><b>Designation: </b>@Item.Designation</div>
</div>
</ContentTemplate>
</AccordionItem>
}
</AccordionItems>
</SfAccordion>
@code {
List<AccordionData> AccordionItems = new List<AccordionData>()
{
new AccordionData
{
EmployeeId = 1,
EmployeeName = "Laura Callahan",
Designation = "Product Manager",
},
new AccordionData
{
EmployeeId = 3,
EmployeeName = "Andrew Fuller",
Designation = "Team Lead",
},
new AccordionData
{
EmployeeId = 4,
EmployeeName = "Anne Dodsworth",
Designation = "Developer"
},
new AccordionData
{
EmployeeId = 5,
EmployeeName = "Nancy Davolio",
Designation = "Product Manager"
}
};
public class AccordionData
{
public string EmployeeName { get; set; }
public int EmployeeId { get; set; }
public string Designation { get; set; }
}
}
```

Output:



Expand Mode in Blazor Accordion Component

The Accordion supports the two listed types of expand modes while expanding or collapsing the item.

- Single
- Multiple

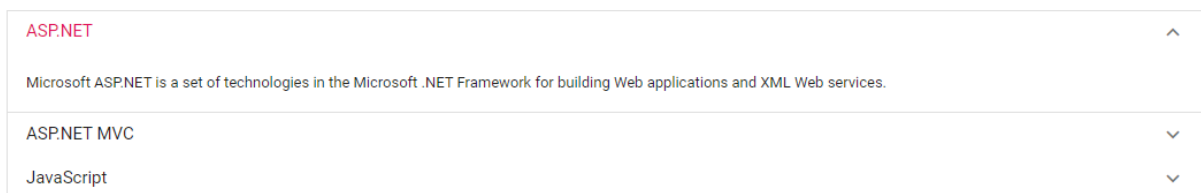
Single

This property enables to expand only one Accordion item at a time. If any new item is expanded, the previously expanded one is collapsed, and the new item is changed to expanded state.

ASPX-CS

```
@using Syncfusion.Blazor.Navigations
<SfAccordion ExpandMode="ExpandMode.Single">
  <AccordionItems>
    <AccordionItem Expanded="true" Header="ASP.NET" Content="Microsoft ASP.NET
    is a set of technologies in the Microsoft .NET Framework for building Web
    applications and XML Web services."></AccordionItem>
    <AccordionItem Header="ASP.NET MVC" Content="The Model-View-Controller (MVC)
    architectural pattern separates an application into three main components:
    the model, the view, and the controller."></AccordionItem>
    <AccordionItem Header="JavaScript" Content="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."></AccordionItem>
  </AccordionItems>
</SfAccordion>
```

Output:



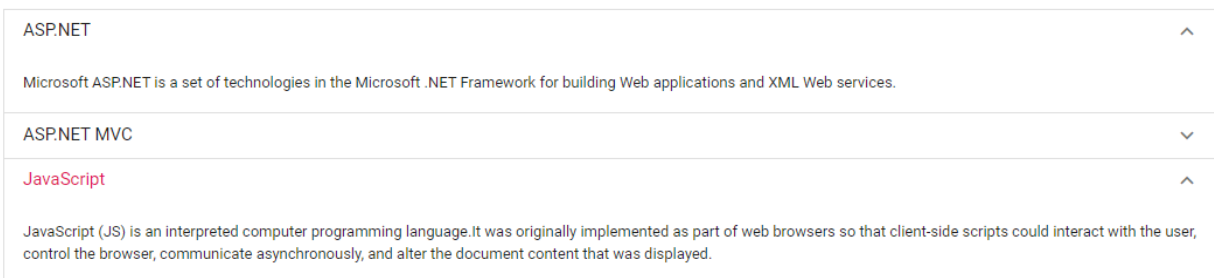
Multiple

The default [ExpandMode](#) of the Accordion is **Multiple**. It enables to expand more than one Accordion item at a time. The expand or collapse action can also be toggled by clicking on it again. For example, expanded item is collapsed when it is clicked again.

ASPX-CS

```
@using Syncfusion.Blazor.Navigations
<SfAccordion ExpandMode="ExpandMode.Multiple" @bind-
ExpandedIndices=ExpandItems>
<AccordionItems>
<AccordionItem Header="ASP.NET" Content="Microsoft ASP.NET is a set of
technologies in the Microsoft .NET Framework for building Web applications
and XML Web services."></AccordionItem>
<AccordionItem Header="ASP.NET MVC" Content="The Model-View-Controller (MVC)
architectural pattern separates an application into three main components:
the model, the view, and the controller."></AccordionItem>
<AccordionItem Header="JavaScript" Content="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."></AccordionItem>
</AccordionItems>
</SfAccordion>
@code {
public int[] ExpandItems = new int[] { 0, 2 };
}
```

Output:



Expanding the items

By default, accordion items were in collapsed state on initial load. To expand a particular item(s) on initial load, either use the [ExpandedIndices](#) property or [Expanded](#) option within the [AccordionItem](#) tag helper. In the following code example, the [ExpandedIndices](#) is used to expand the second and third item.

ASPX-CS

```
@using Syncfusion.Blazor.Navigations
<SfAccordion @bind-ExpandedIndices=ExpandItems>
<AccordionItems>
@foreach (AccordionData Item in AccordionItems)
{
<AccordionItem>
<HeaderTemplate>
```

```
<div>@(Item.EmployeeName)</div>
</HeaderTemplate>
<ContentTemplate>
<div>
<div><b>Employee ID: </b>@Item.EmployeeId</div>
<div><b>Designation: </b>@Item.Designation</div>
</div>
</ContentTemplate>
</AccordionItem>
}
</AccordionItems>
</SfAccordion>
@code{
public int[] ExpandItems = new int[] { 1, 2 };
List<AccordionData> AccordionItems = new List<AccordionData>()
{
new AccordionData
{
EmployeeId = 1,
EmployeeName = "Laura Callahan",
Designation = "Product Manager",
},
new AccordionData
{
EmployeeId = 3,
EmployeeName = "Andrew Fuller",
Designation = "Team Lead",
},
new AccordionData
{
EmployeeId = 4,
EmployeeName = "Anne Dodsworth",
Designation = "Developer"
},
new AccordionData
{
EmployeeId = 5,
EmployeeName = "Nancy Davolio",
Designation = "Product Manager"
}
};
public class AccordionData
{
public string EmployeeName { get; set; }
public int EmployeeId { get; set; }
public string Designation { get; set; }
}
}
```

Output:

Laura Callahan	▼
Andrew Fuller	▲
Employee ID: 3 Designation: Team Lead	
Anne Dodsworth	▲
Employee ID: 4 Designation: Developer	
Nancy Davolio	▼

Animations in Blazor Accordion Component

Accordion supports custom animations for both expand and collapse actions from the provided animation option of the [Animation](#) library. The animation property also allows to set [Easing](#), [Duration](#), and various other effects of an individual's choice.

Default animation is given as [SlideDown](#) for expanding the panel using the [Expand](#) animation property and [SlideUp](#) for collapsing the panel using the [Collapse](#) animation property. The animation can be disabled by setting the animation [Effect](#) as [None](#).

The sample demonstrates the types of animation that suits Accordion. All the animation effects can be checked [here](#).

ASPX-CS

```
@using Syncfusion.Blazor
@using Syncfusion.Blazor.Navigations
@using Syncfusion.Blazor.DropDowns
<div id="container">
<div id="default" style="padding-bottom:75px;">
<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">
<SfDropDownList TValue="AnimationEffect" DataSource="@AnimationData"
TItem="Effect" PopupHeight="150px" Placeholder="Select a animate type"
@bind-Value="ExpandEffect">
<DropDownListEvents ValueChange="ExpandOption" TValue="AnimationEffect"
TItem="Effect"></DropDownListEvents>
<DropDownListFieldSettings Value="ID"
Text="Text"></DropDownListFieldSettings>
</SfDropDownList>
</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">
<SfDropDownList TValue="AnimationEffect" DataSource="@AnimationData"
TItem="Effect" PopupHeight="150px" Placeholder="Select a animate type"
@bind-Value="CollapseEffect">
```

```

<DropDownListEvents ValueChange="CollapseOption" TValue="AnimationEffect"
TItem="Effect"></DropDownListEvents>
<DropDownListFieldSettings Value="ID"
Text="Text"></DropDownListFieldSettings>
</SfDropDownList>
</div>
</div>
</div>
<SfAccordion>
<AccordionAnimationSettings>
<AccordionAnimationCollapse
Effect=@CollapseEffect></AccordionAnimationCollapse>
<AccordionAnimationExpand Effect=@ExpandEffect></AccordionAnimationExpand>
</AccordionAnimationSettings>
<AccordionItems>
<AccordionItem Header="ASP.NET" Content="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.">
</AccordionItem>
<AccordionItem Header="ASP.NET MVC" Content="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.">
</AccordionItem>
<AccordionItem Header="JavaScript" Content="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 Animation development and the creation of desktop
applications.">
</AccordionItem>
</AccordionItems>
</SfAccordion>
</div>
@code{
public AnimationEffect ExpandEffect = AnimationEffect.SlideDown;
public AnimationEffect CollapseEffect = AnimationEffect.SlideUp;
public void
ExpandOption(Syncfusion.Blazor.DropDowns.ChangeEventArgs<AnimationEffect,
Effect> args)
{
this.ExpandEffect = args.Value;
}
public void
CollapseOption(Syncfusion.Blazor.DropDowns.ChangeEventArgs<AnimationEffect,
Effect> args)
{
this.CollapseEffect = args.Value;
}
}

```

```

}
List<Effect> AnimationData = new List<Effect> {
new Effect() { ID= AnimationEffect.SlideDown, Text= "SlideDown" },
new Effect() { ID= AnimationEffect.SlideUp, Text= "SlideUp" },
new Effect() { ID= AnimationEffect.FadeIn, Text= "FadeIn" },
new Effect() { ID= AnimationEffect.FadeOut, Text= "FadeOut" },
new Effect() { ID= AnimationEffect.FadeZoomIn, Text= "FadeZoomIn" },
new Effect() { ID= AnimationEffect.FadeZoomOut, Text= "FadeZoomOut" },
new Effect() { ID= AnimationEffect.ZoomIn, Text= "ZoomIn" },
new Effect() { ID= AnimationEffect.ZoomOut, Text= "ZoomOut" },
new Effect() { ID= AnimationEffect.None, Text= "None" }
};
public class Effect
{
public AnimationEffect ID { get; set; }
public string Text { get; set; }
}
}

```

Output:

Expand Animation	SlideDown
Collapse Animation	FadeIn

ASP.NET	^
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.	
ASP.NET MVC	v
JavaScript	v

Accessibility in Blazor 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-multiselectable`, `aria-disabled`, `aria-expanded`, `aria-selected` and `aria-hidden`. 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.

ARIA attributes

Property	Functionality
-----	-----

| **role** | **Presentation:** It indicates that the element is used to control presentation. This attribute is added to the Accordion element describing the actual role of the element.
 Heading: It identifies the element as a heading that serves as an Accordion header. This attribute is added to all the Accordion header elements describing the actual role of the element. |

| **aria-multiselectable** | It indicates the expand mode in the Accordion. Default value of this attribute is true. If expand mode value is changed as 'single', the attribute value changes to false. |

| **aria-disabled** | It indicates the disabled state of the Accordion and its items. |

| **aria-expanded** | It 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-selected** | It indicates the Selection 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** | It 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-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-level** | It defines the hierarchical level of an Accordion element with its inner level. |

Keyboard interaction

Keyboard navigation is enabled by default. The possible keys are:

Key	Description
Space or Enter	When the focus is on the Accordion header, clicking 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.

Style and Appearance in Blazor Accordion 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 Accordion

Use the following CSS to customize the [Accordion](#).

CSS

```
.e-accordion {
border: 5px solid rgb(173, 255, 47);
}
```


Customizing the Accordion's 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

The following CSS can be viewed to customize the expand and collapse icons of the 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: #2falff !important;
}
```

Content Render Mode in Blazor Accordion Component

Accordion provides support to render the content of all [AccordionItem](#) at initial load which will be maintained in DOM. For that, disable the [LoadOnDemand](#) property to load all the contents.

The default value of the property [LoadOnDemand](#) is true.

ASPX-CS

```
@using Syncfusion.Blazor.Navigations
<SfAccordion LoadOnDemand="false">
<AccordionItems>
<AccordionItem Header="Margeret Peacock" Content="Margeret Peacock was born
on Saturday , 01 December 1990. Now lives at Coventry House Miner Rd.,
London,UK. Margeret Peacock holds a position of Sales Coordinator in our WA
department, (Seattle USA). Joined our company on Saturday , 01 May
2010"></AccordionItem>
<AccordionItem Header="Laura Callahan" Content="Laura Callahan was born on
Tuesday , 06 November 1990. Now lives at Edgeham Hollow Winchester Way,
London,UK. Laura Callahan holds a position of Sales Coordinator in our WA
department, (Seattle USA). Joined our company on Saturday , 01 May
2010"></AccordionItem>
<AccordionItem Header="Albert Dodsworth" Content="Albert Dodsworth was born
on Thursday , 19 October 1989. Now lives at 4726 - 11th Ave. N.E.,
Seattle,USA.Albert Dodsworth holds a position of Sales Representative in our
WA department, (Seattle USA). Joined our company on Friday , 01 May
2009"></AccordionItem>
</AccordionItems>
</SfAccordion>
```

How To

Add/Remove Accordion items in Blazor Accordion Component

Accordion can be added/removed dynamically by iteration of the Accordion Items using conditional **foreach** loop.

In the following demo, initially there are three accordion items in the [AccordionItems](#). On clicking the **Add Item** button, a new item is added to the [AccordionItems](#) resulting in the addition of fourth accordion item to the Accordion component. On clicking the **Remove Item**, the first item of the [AccordionItems](#) is removed from the Accordion component.

ASPX-CS

```
@using Syncfusion.Blazor.Navigations
@using Syncfusion.Blazor.Buttons
<SfButton @onclick="AddItemClick" Content="Add Item"></SfButton>
<SfButton @onclick="RemoveItemClick" Content="Remove Item"></SfButton>
<br />
<br />
<SfAccordion>
<AccordionItems>
```

```
@foreach (AccordionData Item in AccordionItems)
{
    <AccordionItem @bind-Expanded=@Item.IsExpanded>
    <HeaderTemplate>
    <div>@(Item.Header)</div>
    </HeaderTemplate>
    <ContentTemplate>
    <div>@(Item.Content)</div>
    </ContentTemplate>
    </AccordionItem>
}
</AccordionItems>
</SfAccordion>

@code {
List<AccordionData> AccordionItems = new List<AccordionData>()
{
    new AccordionData
    {
        Header = "ASP.NET",
        Content = "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.",
        IsExpanded = true
    },
    new AccordionData
    {
        Header = "ASP.NET MVC",
        Content = "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.",
        IsExpanded = false
    },
    new AccordionData
    {
        Header = "ASP.NET Razor",
        Content = "Razor is an ASP.NET programming syntax used to create dynamic web pages with the C# or Visual Basic .NET programming languages. Razor was in development in June 2010 and was released for Microsoft Visual Studio 2010 in January 2011. Razor is a simple-syntax view engine and was released as part of MVC 3 and the WebMatrix tool set. Side Code content",
        IsExpanded = false
    }
};

public class AccordionData
{
    public string Header { get; set; }
    public string Content { get; set; }
    public bool IsExpanded { get; set; }
}

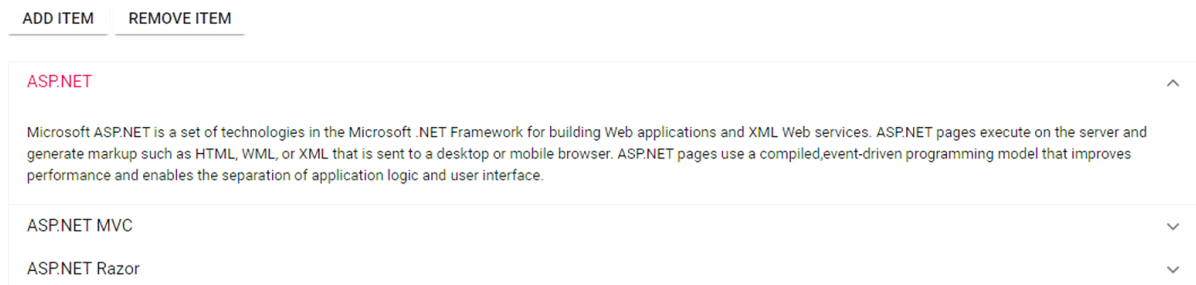
void AddItemClick()
```

```

{
    AccordionItems.Add(new AccordionData
    {
        Header = "JavaScript",
        Content = "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.",
        IsExpanded = false
    });
}
void RemoveItemClick()
{
    if (AccordionItems.Count > 0)
    {
        AccordionItems.RemoveAt(0);
    }
}
}

```

Output:



Show/Hide Accordion item in Blazor Accordion Component

Accordion provides support to show or hide the specified accordion item using the following ways.

- Using conditional rendering.
- Using property.

Using conditional rendering

Accordion provides support to show or hide the specified accordion item dynamically using the conditional **if** statement.

In the following demo, the specified accordion item shows or hides dynamically when the **Show/Hide Item** button is clicked.

ASPX-CS

```

@using Syncfusion.Blazor.Navigations
@using Syncfusion.Blazor.Buttons
<SfButton @onclick="ShowHideItemClick" Content="Show/Hide item"></SfButton>
<br />

```

```
<br />
<SfAccordion>
<AccordionItems>
<AccordionItem Expanded="true">
<HeaderTemplate>
<div>ASP.NET</div>
</HeaderTemplate>
<ContentTemplate>
<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>
</ContentTemplate>
</AccordionItem>
@if (ShowItem)
{
<AccordionItem>
<HeaderTemplate>
<div>ASP.NET MVC</div>
</HeaderTemplate>
<ContentTemplate>
<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>
</ContentTemplate>
</AccordionItem>
}
<AccordionItem>
<HeaderTemplate>
<div>ASP.NET Razor</div>
</HeaderTemplate>
<ContentTemplate>
<div>Razor is an ASP.NET programming syntax used to create dynamic web pages
with the C# or Visual Basic .NET programming languages. Razor was in
development in June 2010 and was released for Microsoft Visual Studio 2010
in January 2011. Razor is a simple-syntax view engine and was released as
part of MVC 3 and the WebMatrix tool set.</div>
</ContentTemplate>
</AccordionItem>
<AccordionItem>
<HeaderTemplate>
<div>JavaScript</div>
</HeaderTemplate>
<ContentTemplate>
<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.</div>
</ContentTemplate>
</AccordionItem>
</AccordionItems>
```

```

</SfAccordion>
@code {
public bool ShowItem = true;
void ShowHideItemClick()
{
ShowItem = !ShowItem;
}
}

```

Using property

Accordion provides support to show or hide the specified accordion item dynamically using the accordion item's [Visible](#) property.

In the following demo, the specified accordion item shows or hides dynamically when the **Show/Hide Item** button is clicked.

ASPX-CS

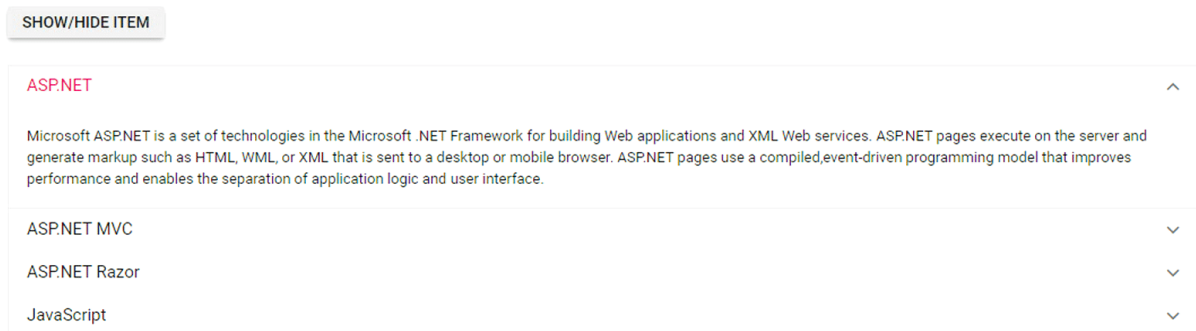
```

@using Syncfusion.Blazor.Navigations
@using Syncfusion.Blazor.Buttons
<SfButton @onclick="ShowHideItemClick" Content="Show/Hide item"></SfButton>
<br />
<br />
<SfAccordion>
<AccordionItems>
<AccordionItem Expanded="true">
<HeaderTemplate>
<div>ASP.NET</div>
</HeaderTemplate>
<ContentTemplate>
<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>
</ContentTemplate>
</AccordionItem>
<AccordionItem Visible=@ShowItem>
<HeaderTemplate>
<div>ASP.NET MVC</div>
</HeaderTemplate>
<ContentTemplate>
<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>
</ContentTemplate>
</AccordionItem>
<AccordionItem>
<HeaderTemplate>
<div>ASP.NET Razor</div>
</HeaderTemplate>

```

```
<ContentTemplate>
<div>Razor is an ASP.NET programming syntax used to create dynamic web pages
with the C# or Visual Basic .NET programming languages. Razor was in
development in June 2010 and was released for Microsoft Visual Studio 2010
in January 2011. Razor is a simple-syntax view engine and was released as
part of MVC 3 and the WebMatrix tool set.</div>
</ContentTemplate>
</AccordionItem>
<AccordionItem>
<HeaderTemplate>
<div>JavaScript</div>
</HeaderTemplate>
<ContentTemplate>
<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.</div>
</ContentTemplate>
</AccordionItem>
</AccordionItems>
</SfAccordion>
@code {
public bool ShowItem = true;
void ShowHideItemClick()
{
ShowItem = !ShowItem;
}
}
```

Output:



Enable or Disable item in Blazor Accordion Component

Accordion provides a support to enable or disable the specified accordion item using accordion item [Disabled](#) property.

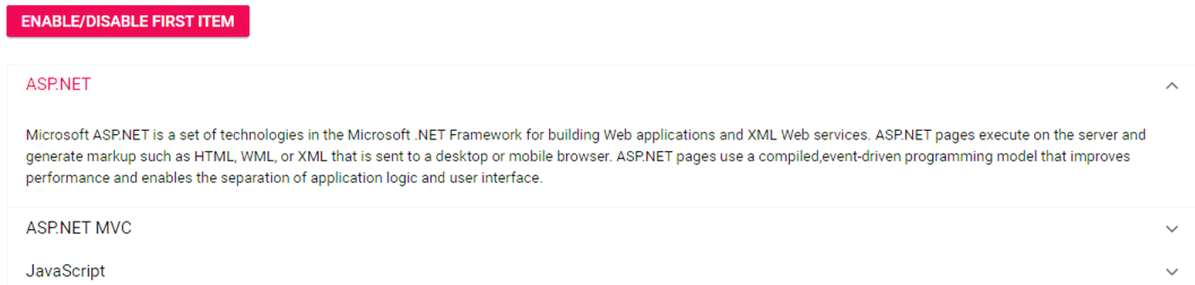
In the following demo, specified accordion item is enabled or disabled dynamically when the **Enable/Disable First Item** button is clicked.

ASPX-CS

```
@using Syncfusion.Blazor.Navigations
@using Syncfusion.Blazor.Buttons
```

```
<SfButton IsPrimary="true" @onclick="EnableItemClick"
Content="Enable/Disable First Item"></SfButton>
<br />
<br />
<SfAccordion>
<AccordionItems>
<AccordionItem Disabled=@IsEnable Expanded="true" Header="ASP.NET"
Content="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."></AccordionItem>
<AccordionItem Header="ASP.NET MVC" Content="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."></AccordionItem>
<AccordionItem Header="JavaScript" Content="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."></AccordionItem>
</AccordionItems>
</SfAccordion>
@code {
public bool IsEnable { get; set; } = false;
public void EnableItemClick()
{
IsEnable = !IsEnable;
}
}
```

Output:



Add Icon to Header in Blazor Accordion Component

The icon custom css class can be added to the Accordion header using the [IconCss](#) property, and also css styles can be added to the defined class. The accordion icon element is rendered before the header text in the DOM element.

ASPX-CS

```
@using Syncfusion.Blazor.Navigations
<SfAccordion>
<AccordionItems>
<AccordionItem Header="Athletics" IconCss="e-athletics e-acrdn-icons"
Expanded="true">
<ContentTemplate>
<ul>
<li><span class="e-acrdn-icons e-content-icon marathon"></span>
Marathon</li>
<li><span class="e-acrdn-icons e-content-icon javelin"></span> Javelin
Throw</li>
<li><span class="e-acrdn-icons e-content-icon discus"></span> Discus
Throw</li>
<li><span class="e-acrdn-icons e-content-icon highjump"></span> High
Jump</li>
<li><span class="e-acrdn-icons e-content-icon longjump"></span> Long
Jump</li>
</ul>
</ContentTemplate>
</AccordionItem>
<AccordionItem Header="Water Games" IconCss="e-water-game e-acrdn-icons">
<ContentTemplate>
<ul>
<li><span class="e-acrdn-icons e-content-icon dive"></span> Diving</li>
<li><span class="e-acrdn-icons e-content-icon swimming"></span>
Swimming</li>
<li><span class="e-acrdn-icons e-content-icon marathan_swim"></span>
Marathon Swimming</li>
<li><span class="e-acrdn-icons e-content-icon sync_swim"></span>
Synchronized Swimming</li>
<li><span class="e-acrdn-icons e-content-icon waterpolo"></span> Water
Polo</li>
</ul>
</ContentTemplate>
</AccordionItem>
<AccordionItem Header="Racing" IconCss="e-racing-games e-acrdn-icons">
<ContentTemplate>
<ul>
<li><span class="e-acrdn-icons e-content-icon cycle_BMX"></span> Cycling
BMX</li>
<li><span class="e-acrdn-icons e-content-icon cycle_Mountain"></span>
Cycling Mountain Bike</li>
<li><span class="e-acrdn-icons e-content-icon cycle"></span> Cycle
Racing</li>
<li><span class="e-acrdn-icons e-content-icon sailing"></span> Sailing</li>
<li><span class="e-acrdn-icons e-content-icon rowing"></span> Rowing</li>
</ul>
</ContentTemplate>
</AccordionItem>
<AccordionItem Header="Indoor Games" IconCss="e-indoor-games e-acrdn-icons">
```

```
<ContentTemplate>
<ul>
<li><span class="e-acrdn-icons e-content-icon tennis"></span> Table
Tennis</li>
<li><span class="e-acrdn-icons e-content-icon badminton"></span>
Badminton</li>
<li><span class="e-acrdn-icons e-content-icon volleyball"></span>
Volleyball</li>
<li><span class="e-acrdn-icons e-content-icon boxing"></span> Boxing</li>
<li><span class="e-acrdn-icons e-content-icon swimming_In"></span>
Swimming</li>
</ul>
</ContentTemplate>
</AccordionItem>
</AccordionItems>
</SfAccordion>
<style>
@@font-face {
font-family: 'acrdn-icons';
src: url(data:application/x-font-ttf;charset=utf-
8;base64,AAEAAAAKAIAAAwAgTlMvMj0gSSSYAAAEoAAAAMVnTlYXNn+g2AAAB5AAAAGRnbHlmslg
PRQAAAnwAABvMaGVhZA6+wXwAAADQAAAAANmhoZWEHfaOBAAAArAAAACRobXR4YcP/xgAAAYAAAAAB
kbG9jYU2IVXoAAAJIAAAANG1heHABLwC3AAABCAAAACBuYW1lNl/OpQAAHkgAAAKFcG9zdBxr6o4
AACDQAAABawABAAADUv9qAFoEAP/i//0D6wABAAAAAAAAAAAAAAAAAGQABAAAAQAAxGQXJ18
PPPUACwPoAAAAANXpvlcAAAAA1em+V//iAAD6wPpAAAAACAACAAAAAAAAAAAAAAAZAKsADAAAAAA
AAgAAAAoACgAAAP8AAAAAAAAAAQPPAZAABQAAAAnoCvAAAAIwCegK8AAAB4AAxAQIAAAIABQMAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAUGZFZABA5wDnFwNS/2oAWgPpAJYAAAAABAAAAAAAAABAAAAAPoAAA
D6AAAA+gAAAPoAAAD6AAAA+j/9wPo//AD6AAAA+gAAAPo/+ID6AAAA+gAAAPoAAAD6AAAA+gAAAP
oAAAD6AAAA+gAAAPoAAAD6AAAA+gAAAPoAAAD6P/9A+gAAAAAAIAAAADAAAAFAADAAEAAAAUAAQ
AUAAAAQABAABADnF//AADnAP//AAAAQAEAAAAAQACAAMABAAFAAYABwAIAAkACgALAAwADQA
OAA8AEAARABIAEwAUABUAFgAXABgAAAAAFgAqgFGakACjAL0A0AEGASOBPFgFjgZeBrIHiggWCMI
JOAoUCpILWgwUDI4NXA3mAAIAAAAAA3sD5wANADcAAAEeARcWNicuAScmJw4BBRc3FxEUdWEOAR8
BHgE/ATUfARY7ATI2LwImpQEzMjY3NS4BIyEnNycBxQUnHDNDBwUnGwkJLTf+Qha70geWBQMDMwQ
MBtZ5HgQLYAgIAStpBc8HCAEBcAf+O5OsFgL/HcKGCEeZHCgGAgEBP+4WutP+3AgFYgQMBk4FAwS
Mi3KJDAsHxGIFBs0IBz4HCZOsFgAABAAAAAADhwPoAAMAFaAdAC8AADcXNy8BHgEHNzEXBzYWFx4
BNwEGFiUOASImNDYyFiUGBwE+ATc+AScuAjEmBw4BAVjxVYMcAwUjgyIKGA4lTyj+rgcDAjKbK0A
rK0Ar/gkkCgFfGjEYRaNML1M2NzRekOJY8lg5DxkII4QjAwECBwIGAVInUFQgKyTAkyuDMjH+oQY
WEizNfkJQJyUCB4cAAAAFAAAAAAPpA+cACwAXACMAWABhAAAlDgEHLgEnPgE3HgElDgEHLgEnPgE
3HgEFHgEXPgE3LgEnDgEDBg8BDgEfARYPASyGBw4BFx4BNz4BPwE+AS8BJjY/AjYWHwEeATczJy4
BIwciLwEuASciNx4BMjY0JiIGAUICSjk4SweBSzG5SgJ3AU54OEsCAks4OEv+XwJmTklNagJnTU5
mhRENfCESEkEGCD9EstQUARMzSEUaHQF4EwYOPwQEBkwECBAFIw83IpgBASIAaSQ0GKRA7IhIGATF
JMDBJMcY4SweBSzG4SweBS0k5SgICSjk4SweBSzhOZwICZ05NZwICZwGbBgpeGk4mhQsLU0EXWCZ
XJ1kaQBLcJJ4aQB5/Bw4FOgIEBQLHHIIBQBoiAQTSiHUBWYQxMUKwMAAEAAAAAAPiA+cAaAb1AJ4
AqgAAEyIHDgEHFT4BNx4BMjY3HgEyNjceATI2Nx4BMjY3HgEyNjceARc1LgEnJiM1ByIHfAYHLgE
1JiM1ByIHfAYHLgEnJiMnFSIHfAYHLgE1JiMnFSIHfAYHLgE1JiMnFSIHfAYHLgE1JiM1AQYWFxY
+ASYNjIMOaQUeAR8BHgEzPgE1FBYXPgE1FBYXPgE3LgEnLgEvAS4BJzc2NycmJw4BEx4BFz4BNy4
BJw4BUggEASiIHigLCyk+KQsLKT4pCwsqPSOkCy09KgsLKT0qCwspHiIjAQQIAQcFJCiJJAQIAQc
FJCiJiWEECAIEIBCQiIyQECAIEIBCQiIyQECAIEIBCQiIyQFCAELDTA0NE4fJzMTEylC/t0NfB4LCh0
TKikqKikqKSoaJAoMaxMOTCpQMhQFBgYBCS0fECg4ATosLDoBATosLDoBGgYDIwMrArKMDRgYDQ0
YGA0NGBgNDRgYDQ0YGA0MGQErAyIEBgEBBgMkAgIkAwYBAQYDJAICJAMGAQGCNfkrDTZnVBAFATGUHII2JgkMAyoDAyoDAyoDAyoDARULDMkNBg8
GDA5SBT02MgEFHzOPAScsOgEBOiwsOgEBOgAAAAAMAAAAAA+ID5wALABcAKQAAARY2NxY2Ny4BJxY
GAx4BFz4BNy4BJw4BBQclFwUWNjCWNjceATCWNjCBAsgqYQQgUxksRBwCotABQzIyQwEBQzIyQ/3
ADwE6Lv66G2wkI10eGF8oJ1kb/sACgjYaBhUJwMTHwhNAQYyQwEBQzIyQwEBQzU7SjUkSQIuNAw
rKhI8OxUgAWsAAv/3AAADEgPnAAwAQQAAQYWFxY2NzYmJyYOASUGFhcWNjCXAwyfAgcGFB8BFjY
/ARChBhY7ATY/ASc2LwE3FRQWOWEYnJURNcclLgEnJgYCYBYXJyLTGRYXJxs6MP28Fis3NmYaYJQ
DAgN1fAQFRgYMBHECHQEJB18MAykBAgVJQAKGPGcJCP6kFlssIDMDhihUGBYWKChUGA8BHBGpYSE
```

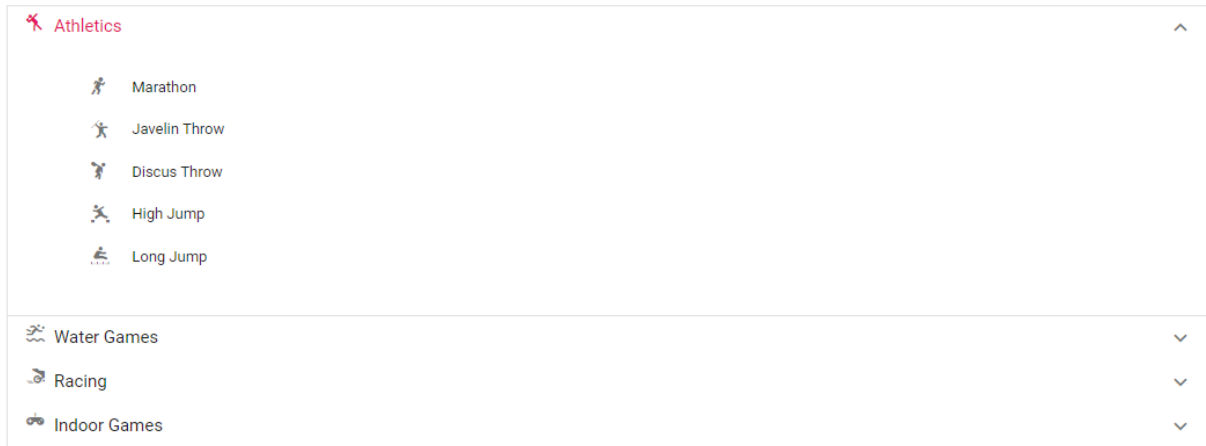
eCSU3/wAGBwZ1jwUMBT0EAQSCArUHCwEM+AQGB1VvdAYJCQYBGQkFySc0AQEYAAL/8AAAAxID5wA
 DACoAACUXJScBJgYXPgEXHgEjLgEHIGYXHgEXJScuAQcmNjceArc3NgInLgEnJgYBI2MBjWb+EA2
 eCgaHOQMBIAJcQwguOQaTQAFIA2uNag8KAXt9EGAHGn8BcWIGr9vWtNMCogOVxgcNVQIaCUkqVU8
 GdCSbBpU7BwQaAQuoNS8OATyABUSGARAAAAABAAAAADtwPnAEwAUACJAjIAABMGJxU2Nz4BFzY
 XHgEzByYPAQ4BFBYXFjI/AT4BJz8BNhc2FjI3PgEzNhYyNz4BMzYXHgEzNSImJyYHJgYiJy4BIyY
 GIicmIgYiJgciAQcjPwEvASYHBg8BFSEVMjY3NjcyFhcWMjc2MhceATMyNzYyFyYzYmJc2Nxy2Nz4
 BNzUnITcXfjY3Nj3JyUeATI2NCYiBkscGxscDh4OHxsKFw1GJiFMCgkJCRITeEVAOBAlfExsfHDc
 3HA4eDhw3NxoHhg4gGw81EBA1DxsgGzc4Gw4fDh82OBsXQDc3NiAbAgN/GTghyBA1HSAIOv7oERC
 LICUSJRAULxghTCILFQsUGCFNIRQYExggJQsTBw1Ezr+lXkMEhsFBAXcF/6NAS9ILy9ILwFzDQF
 ZAQ4HBgEBDQUHShcgVgcVFxYHEhJTDiMTYgkNAQEbDgcFARsOBwUBDQcKWQcHDQEBGg0HBQEaDQ0
 aGgEBH4asFk4JChMSINcHeQQFDwEHCQkJDw8FBAkPDwkJDwEBBwkEDgszAXwFCRINFhFeElkkLy9
 ILy8ABAAAAAAD4gPnAAMABwA8AEkAACUzNSMFMzUjAQcxBxUnJgYPAQYWHwIHBg8BDGefAR4BPwE
 2PwEXFh8BFj8BNiYvASYvAQM3PgEvASYjIiUOARceATc+AScuAgMou7v82bq6ApGt15UHCgIPAgc
 G7z5NBQOABQEEQgQNBX8EBIqLBgafCwYtAwYHkgYEQpmsBgMDHwUJBP49JxYWGfQoKBYWDzE5JJy
 cnALgZHwBJgEGBjwHCwE9aiACA2sEDQVTBQIEZgMCOGAEEASIC2MHDAlpAQmxAQhkBAwFNgg3GVM
 pJxcwGVMPghwBAAAF/+IAAPrA+cACAARAC0APwBHAAABDgEiJjQ2MhY3FAYiJjQ2MhY1FTMyFhQ
 GKwEVFAYiJj0BIyImNDY7ATU0NjIWBxYXFjY3Mx4BNzY3NiYnIQ4BARUzNS4BIgYC5QETHhMTHhN
 0FB0UFB0U/ZBGCg0NCkYNEw5FCg0NCkUOEw3GFyd4vxR3Fb54JxdBe3v94nx7Aec+AREbEQHtDxM
 THhQUZQ8UFB4TFBRFDRQNRQoNDQpFDRQNRQoNDfAnFjxjbW1jPBYPngsgFBcgBr8PDCw4OAAAYAAA
 AA+ID5wALABCAIwAvAFQAXQAAQ4BBY4BJz4BNx4BBQ4BBY4BJz4BNx4BFx4BFz4BNy4BJw4BBR4
 BFz4BNy4BJw4BAQCGRQWHwEVHgE3PgE9ATYmLwE3F4BOWE2NCcjJzQmJyMiBjceATI2NCYiBgO
 rAlU+QVMBAVU/QFP9ugJVPkBUAQJVPj5V6AJ1VlhZAgN0V1Z1/bYCC1hYcwICdVZYcwGXqhASD60
 BIBgQFAIHGJpNagRCJMiIoFXGhsDBhxAASxBLcXBLAF2QVMBAVU/QFMCA1NAQVMBAVU/QFMCA1N
 AWHMCAnVWHMCAnNYVnMEAnVWHMCAnMBjp4UGBIeCFqnFhgDAXkRyAileTRpTwoJAzcefwIYAgc
 yISwsQswsAAAAAAUAAAAA+ID5wBBAEWAVQCEAI0AADcyFhceARcxMzE+ATc+ATczMTIWFx4BFz4
 BNz4BMhYXHgEXNSImJy4BJw4BBw4BIiYnLgEnIzUOAQcOASImJy4BJxMGfH8BNycmByIGJR4BMjY
 0JiIGJQC0AQcGFj8BFwcGDwEjIgyUFhchMjY/AhcWFzZM+ATQmJyMvASYxLwEzJyYnIhceATI2NCY
 iBgESGBETPDafMTwTDxgPBBMaERQ/NDRAExIZJRORFEA0FhwSEz0xMT0UER0qHBITOi4HMj8UEX0
 pHRMTPzMBGw8UziXOCgoOGQM+ASK/Kio/Kf45lQ8YAgIkGn9IeAcEptUUGhoUAQMNfGzYgFQOEQA
 UGhoUg24pAQMFAn0OEAbKASo+Kio+KqMQEHUpAgIpFREQARESFSKCAigWEhAQEHUpAmISEhQoAQE
 nFRiSEhIUJgMBAigWEhISEhYpAQHNEYQJULxSBAEQWYApKT8qKlSdBbGQGYAEGHtd4BwqbGigaAQ4
 MeIBLDAEBGigaAXMiAQIDYgoBGR8qKj8pKQAAwAAAAADCQPnAAQACQAXAAABFgIHEQMMAj8CNSM
 VJgADPgEXBzEhHgEXIT4BNyE1PgEXEAInNRY2JwYmNT4BJwYmAZ5bRhWfHfTxBh8fGf6xAwFzghb
 +qA9gSwGUTWsC/o4Fqmn7HTsqARobJhECKTKDZav+qS0Cnf1VMAGh0QlqEHcC/rf+nBNxhwM/XR4
 mJh2FoubASKBCQYnCCsEGggCFCMBJwoABgAAAAAD4gPnACQARwBTAF0AfACWAAATBiMVMjYyFjI
 2MhYyNjIWMjYyFjM1IiYiBiImIgYiJiIGIiYiEwYiJyYnFRYXFjI2MhYyNjIWMjYzMhc1JiMiBiI
 mIgYiJiIFFBYXPgE3LgEnDgElFBYXPgE0JiIGJQUOAR8CBRYXFjI2MhYyNjIWFzI3Azc+ATUuASc
 iJQC0AR8CBxUWFxYyNjIWMzI3Jzc+ATQmIz4cISE5RDpCOUQ5QjLEOke6RDkhITLEOke6RD1COUQ
 5QjPEARIPegCHBwcSKSQqJCkkKiQpJBYPDg4PFiQpJCokKSQqAm9BNTJDAQFDMjJD/fYoIh8qKj8
 qAlP+nhUfDQnc/quODB1COUQ5QjLEOSEYFsXlGiABJhwE/dfdDRMHAjPHDgWskSMrJBQPDnuPEBQ
 YEgEWDV0bGxsbGxsbG10bGxsbGxsbAUSICAMCOgIDCBERERERBTOfEREREU4zQwICQzMzQwICQ7k
 gKgEBKkAqKkNBbTAjB6nBBAUNGhoaGgEIAWcrBSQaHSYBbikDHhYEaidQAgUIEREE4RoEFiMYAAA
 AAwAAAAADCAPnABIATQBbAAATDwIGfH8BFj8Dni8BJiMiEwYxBwYWHwEWNj8BFwMGHwEHBhYfARY
 2PwE2JyYvATcXMDExFjY/ATYmLwEmBg8BJyImLwEiMSYjJwY3BhYXFjY3NiYnJiMiBpc9AlQEAWd
 UCwhhAigECEgFBgkuAUMCBAY5BgwDLcrZBAi1PwMFB1UGDAJqAwUBAlNaLDAFDQSCBQIFMAUMBFE
 +AQICxAEDA4gJyRQkMDfCfXqkMBgYJT4Ba4oEawYOBQFCQCoMDWwsHQgQBpgGXBgWGDQIEBmUD/vw
 LK8iPBwsDJQMEB+8IBwMCTc0lKAQBBZwFDQqOBAEFYTUBAVcCDAF7MVwXFCQwMVwXCiYAAAMAAAA
 AA8MD5wBRAfOAcAAANw4BBxUyNjc+ATM2Fx4BMj4CMzYXHgEzFjc+ATM2Fx4BMxY3PgEzNhceATM
 1IicuASmMbw4BBYInLgEjJgcOAQciJy4BIyYHDgEHiciuASIGEX4BMjY0JiIGAQMHBgCDBh4BNj8
 BJTY3EZmJy4BBjwOHRAQHAWOHxEhHQ0dIRwcIBAHHQ4dEB8cDh8RIRwOHRAGw4gECEDh0QIBs
 OIBAhHQ4dEB8cDh8QIhwOHRAgGw4gECEdDhwRIBcOICEfggEuQy8vQy4Cf6XyEgq6CQghIwqDAR4
 SCsMMDBUOHRtsBQcBWQcFBQcBDQUHBwOHAQ0FBwENBQcBDQUHAQ0FBwENBQdZDQUHAQ0FBwENBQc
 BDQUHAQ0FBwENBQcBDQUHBwGLIS4uQy4uAa3+6ZIJE/69ESMUCRDjsQkTAU8WLAWFAQ4AAAAAAwA
 AAAAD4gPnACQAMABOAAATBiMVMjYyFjI2MhYyNjIWMjYyFjM1IiYiBiImIgYiJiIGIiYiARQWFz4
 BNy4BJw4BEwUOAR8CBRYXFjI+ARYyPgEWFzI3Azc+ATUuASc+HCEhOUQ6QjLEOUI5RD1COU5ISE
 5RT1COUQ5QjLEOUI6RAKCQTUyQwEBQzIyQ0n+nhUfDQnc/quODB1COUQ5QjLEOSEYFsXlGiABJhw
 BhAlDgXsbGxsbGxtdGxsbGxsbGwEGM0MCAkMzM0MCAkMBHEEFMCMHqcEDBg0aARsaARsBCAFnKwU

384

```
fUmFjZQtXYXRlcl9HYW1lcwdTYWlsaW5nEU1hcmF0aG9uX1N3aW1taW5nCE1hcmF0aG9uBkRpdml
uZw1Td21tbWluZzEJQmFkbWludG9uB1JhY2luZxVTeW5jaHJvbml6ZWRfU3dpbW1pbmcLVm9sbGV
5X0JhbGwJQXRobGV0aWNzFEN5Y2xpbnRhaW5CaWt1CUxvbmdfSnVtcAAAAA==)
format('truetype');
font-weight: normal;
font-style: normal;
}
.e-acrdn-icons {
font-family: 'acrdn-icons';
font-size: 16px;
}
.cycle_BMX::before {
content: "\e702"
}
.javelin::before {
content: "\e700";
}
.marathon::before {
content: "\e70e";
}
.tennis::before {
content: "\e701";
}
.waterpolo::before {
content: "\e703";
}
.swimming::before {
content: "\e704";
position: relative;
top: 5px;
}
.discus::before {
content: "\e705";
}
.boxing::before {
content: "\e706";
}
.rowing::before {
content: "\e707";
}
.highjump::before {
content: "\e708";
}
.cycle::before {
content: "\e70a";
}
.sailing::before {
content: "\e70c";
}
.marathan_swim::before {
content: "\e70d";
}
.boxing::before {
content: "\e706";
}
.dive::before {
content: "\e70f";
```

```
}
.swimming_In::before {
content: "\e710";
position: relative;
top: 2px;
}
.badminton::before {
content: "\e711";
}
.sync_swim::before {
content: "\e713";
position: relative;
top: 3px;
}
.volleyball::before {
content: "\e714";
}
.cycle_Mountain::before {
content: "\e716";
}
.longjump::before {
content: "\e717";
}
.e-athletics::before {
content: "\e715";
}
.e-water-game::before {
content: "\e70b";
}
.e-racing-games::before {
content: "\e712";
}
.e-indoor-games::before {
content: "\e709";
}
.e-acrdn-icons:not(.e-icons) {
padding: 0 16px 0 0;
vertical-align: middle;
}
li {
line-height: 36px;
list-style-type: none;
text-indent: 16px;
}
</style>
```

Output:



Prevent the Expand or Collapse item in Blazor Accordion Component

The expand and collapse of an accordion item can be prevented for a specific condition. For example, if there is a button in the accordion header, clicking on it must prevent the expanding and collapsing. This can be achieved by checking the condition on Accordion [Expanding](#) and [Collapsing](#) events. Refer the following code snippet in which the prevention of collapse and expand action occurs while clicking the Button and DropDownList.

- DropDownList - Prevents the expand and collapse of an accordion item while opening the DropDownList using the [OnOpen](#) event of the DropDownList. It also prevents the expand or collapse of an accordion item while closing or selecting the DropDownList by using the [OnClose](#) and [ValueChange](#) event of the DropDownList.
- Button - Prevents the expand or collapse of an accordion item while clicking the button by using the `onclick` event of the Button.

ASPX-CS

```
@using Syncfusion.Blazor.DropDowns
@using Syncfusion.Blazor.Navigations
@using Syncfusion.Blazor.Buttons
<SfAccordion>
<AccordionEvents Expanding="OnExpanding"
Collapsing="OnCollapsing"></AccordionEvents>
<AccordionItems>
<AccordionItem Expanded="true">
<HeaderTemplate>
<SfDropDownList TValue="string" TItem="Countries" Placeholder="Select a
country" DataSource="@Country" Width="120">
<DropDownListEvents TValue="string" TItem="Countries" OnOpen="OnOpen"
OnClose="OnClose" ValueChange="OnChange"></DropDownListEvents>
<DropDownListFieldSettings Value="CountryName"></DropDownListFieldSettings>
</SfDropDownList>
</HeaderTemplate>
<ContentTemplate>
<div>Dropdown Content</div>
</ContentTemplate>
</AccordionItem>
<AccordionItem>
<HeaderTemplate>
```

```

<SfButton Content="@ButtonText" @onclick="AddRemoveItem">
</SfButton>
</HeaderTemplate>
<ContentTemplate>
<div>Button Content</div>
</ContentTemplate>
</AccordionItem>
</AccordionItems>
</SfAccordion>
@code
{
public bool IsDropdown = false;
public bool IsDropdownItemChanged = false;
public bool IsButton = false;
public string ButtonText = "OnClick";
public List<Countries> Country = new List<Countries>() {
new Countries(){ CountryName= "Australia", CountryId= "2" },
new Countries(){ CountryName= "United States", CountryId= "1" },
};
public class Countries
{
public string CountryName { get; set; }
public string CountryId { get; set; }
}
public void OnOpen()
{
IsDropdown = true;
}
public void OnClose()
{
IsDropdown = !IsDropdownItemChanged;
}
public void OnChange()
{
IsDropdownItemChanged = true;
}
public void AddRemoveItem()
{
IsButton = true;
}
public void OnExpanding(ExpandEventArgs args)
{
if (IsButton || IsDropdown)
{
args.Cancel = true;
PreventItem();
}
}
public void OnCollapsing(CollapseEventArgs args)
{
if (IsButton || IsDropdown)
{
args.Cancel = true;
PreventItem();
}
}
private void PreventItem()

```



```

{
    if (IsButton)
    {
        IsButton = false;
    }
    else if (IsDropdown)
    {
        IsDropdown = false;
        IsDropdownItemChanged = false;
    }
}
}
}

```

Output:

The screenshot shows a Blazor Accordion component with two items. The first item is a dropdown menu with the header "Select a country" and a downward arrow. Below the header is the text "Dropdown Content". The second item is a button with the text "ONCLICK" and an upward arrow. Below the button is the text "Button Content".

Add Nested Accordion in Blazor Accordion Component

Accordion supports to render the nested level of Accordion by using the [ContentTemplate](#) property. To render the nested Accordion, define the nested Accordion elements within the `ContentTemplate` property of the parent Accordion.

ASPX-CS

```

@using Syncfusion.Blazor.Navigations
<SfAccordion>
<AccordionItems>
<AccordionItem Header="Video">
<ContentTemplate>
<SfAccordion>
<AccordionItems>
<AccordionItem Header="Video Track1"></AccordionItem>
<AccordionItem Header="Video Track2"></AccordionItem>
</AccordionItems>
</SfAccordion>
</ContentTemplate>
</AccordionItem>
<AccordionItem Header="Music">
<ContentTemplate>
<SfAccordion>
<AccordionItems>
<AccordionItem Header="Music Track1"></AccordionItem>
<AccordionItem Header="Music Track2"></AccordionItem>
<AccordionItem Header="Music New">
<ContentTemplate>
<SfAccordion>
<AccordionItems>
<AccordionItem Header="New Track1"></AccordionItem>
<AccordionItem Header="New Track2"></AccordionItem>

```

```

</AccordionItems>
</SfAccordion>
</ContentTemplate>
</AccordionItem>
</AccordionItems>
</SfAccordion>
</ContentTemplate>
</AccordionItem>
<AccordionItem Header="Images">
<ContentTemplate>
<SfAccordion>
<AccordionItems>
<AccordionItem Header="Track1"></AccordionItem>
<AccordionItem Header="Track2"></AccordionItem>
</AccordionItems>
</SfAccordion>
</ContentTemplate>
</AccordionItem>
</AccordionItems>
</SfAccordion>

```

Output:

Video	▼
Music	▲
Music Track1	
Music Track2	
Music New	▲
New Track1	
New Track2	
Images	▼

Create Wizard in Blazor Accordion Component

Accordion items can be disabled and expanded dynamically using the accordion item's [Disabled](#) and [Expanded](#) property.

The following demo is designed for simple payment module that enable or disable Accordion based on the sequential validation of each Accordion content.

ASPX-CS

```

@using Syncfusion.Blazor.Navigations
@using Syncfusion.Blazor.Inputs
@using Syncfusion.Blazor.Buttons
@using Syncfusion.Blazor.Calendars
@using Syncfusion.Blazor.Popups

```

```

<div class='template_title'> Online Shopping Payment Module</div>
<SfAccordion ID="AccordionElement" @ref="@Accordion">
  <AccordionEvents Created="OnCreate"></AccordionEvents>
  <AccordionItems>
    <AccordionItem Disabled=@DisableSignInItem @bind-
Expanded="ExpandSignInItem">
      <HeaderTemplate>Sign In</HeaderTemplate>
      <ContentTemplate>
        <div id="Sign_In_Form" style="padding:10px">
          <form id="formId">
            <div class="form-group">
              <div class="e-float-input">
                <SfTextBox @ref="@EmailTextbox" Placeholder="Email"></SfTextBox>
              </div>
              <div class="e-float-input">
                <SfTextBox @ref="@PasswordTextbox" Placeholder="Password"
Type="InputType.Password"></SfTextBox>
              </div>
            </div>
            <div style="text-align: center">
              <SfButton @onclick="@OnSignIn">Continue</SfButton>
            </div>
            @if (EmptyField)
            {
              <div class="Error">* Please fill all fields</div>
            }
          </div>
        </div>
      </ContentTemplate>
    </AccordionItem>
    <AccordionItem Disabled=@DisableDeliveryItem @bind-
Expanded="ExpandDeliveryItem">
      <HeaderTemplate>Delivery Address</HeaderTemplate>
      <ContentTemplate>
        <div>
          <div id="Address_Fill" style="padding:10px">
            <form id="formId_Address">
              <div class="form-group">
                <div class="e-float-input">
                  <SfTextBox @ref="@NameTextbox" Placeholder="Name"></SfTextBox>
                </div>
              </div>
              <div class="form-group">
                <div class="e-float-input">
                  <SfTextBox @ref="@AddressTextbox" Placeholder="Address"></SfTextBox>
                </div>
              </div>
              <div class="form-group">
                <SfNumericTextbox TValue="int" @ref="@MobileNumberTextbox"
Placeholder="Mobile" FloatLabelType="@FloatLabelType.Auto"
ShowSpinButton="false" Format="0"></SfNumericTextbox>
              </div>
            </form>
            <div style="text-align: center">
              <SfButton @onclick="@OnContinue">Continue</SfButton>
            </div>
            @if (EmptyField)
            {

```

```

<div class="Error">* Please fill all fields</div>
}
</div>
</div>
</div>
</ContentTemplate>
</AccordionItem>
<AccordionItem Disabled=@DisableCardItem @bind-Expanded="ExpandCardItem">
<HeaderTemplate> Card Details</HeaderTemplate>
<ContentTemplate>
<div id="Card_Fill" style="padding:10px">
<div class="col-xs-6 col-sm-6 col-lg-6 col-md-6">
<div class="form-group">
<SfNumericTextBox TValue="int" @ref="@CardNumberTextbox" Placeholder="Card
No" FloatLabelType="@FloatLabelType.Auto" ShowSpinButton="false"
Format="0"></SfNumericTextBox>
</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">
<SfTextBox @ref="@CardHolderNameTextbox" Placeholder="CardHolder
Name"></SfTextBox>
</div>
</div>
</div>
<div class="col-xs-6 col-sm-6 col-lg-6 col-md-6">
<SfDatePicker TValue="DateTime" Width="100%" Placeholder="Expiry Date"
Format="MM-yyyy" Readonly="false"></SfDatePicker>
</div>
<div class="col-xs-6 col-sm-6 col-lg-6 col-md-6">
<div class="form-group">
<SfNumericTextBox @ref="@CvvTextbox" TValue="int" Placeholder="CVV"
FloatLabelType=@FloatLabelType.Auto Format="0"
ShowSpinButton="false"></SfNumericTextBox>
</div>
</div>
</div>
<div style="text-align: center">
<SfButton @onclick="@GoBack">Back</SfButton>
<SfButton @onclick="@SaveDetails">Save</SfButton>
@if (EmptyField)
{
<div class="Error">* Please fill all fields</div>
}
</div>
</ContentTemplate>
</AccordionItem>
</AccordionItems>
</SfAccordion>
<SfDialog @ref="AlertDialog" Width=250 Target="#AccordionElement"
IsModal=true Visible=false ShowCloseIcon="true">
<DialogEvents Created="DialogCreate"></DialogEvents>
<DialogTemplates>
<Header><div>Alert</div></Header>
<Content><div>Your payment successfully processed</div></Content>
</DialogTemplates>

```

```

<DialogButtons>
<DialogButton OnClick="@OnSubmit" Content="OK"
IsPrimary="true"></DialogButton>
</DialogButtons>
</SfDialog>
@code{
SfAccordion Accordion;
SfTextBox EmailTextbox;
SfTextBox PasswordTextbox;
SfTextBox NameTextbox;
SfTextBox AddressTextbox;
SfDialog AlertDialog;
SfTextBox CardHolderNameTextbox;
public SfNumericTextBox<int> CardNumberTextbox { get; set; }
public SfNumericTextBox<int> MobileNumberTextbox { get; set; }
public SfNumericTextBox<int> CvvTextbox { get; set; }
public Boolean EmptyField { get; set; } = false;
public Boolean DisableSignInItem { get; set; }
public Boolean DisableDeliveryItem { get; set; }
public Boolean DisableCardItem { get; set; }
public Boolean ExpandSignInItem { get; set; }
public Boolean ExpandDeliveryItem { get; set; }
public Boolean ExpandCardItem { get; set; }
public void OnCreate()
{
DisableDeliveryItem = true;
DisableCardItem = true;
}
public async Task OnSignIn()
{
if (EmailTextbox.Value == null || PasswordTextbox.Value == null)
{
EmptyField = true;
}
else
{
EmptyField = false;
DisableDeliveryItem = false;
await Accordion.SelectAsync(1);
ExpandSignInItem = false;
ExpandDeliveryItem = true;
}
}
public async Task OnContinue()
{
if (NameTextbox.Value == null || AddressTextbox.Value == null ||
MobileNumberTextbox == null)
{
EmptyField = true;
}
else
{
DisableCardItem = false;
await Accordion.SelectAsync(2);
ExpandDeliveryItem = false;
ExpandCardItem = true;
}
}
}

```

```
}
public async Task GoBack()
{
    await Accordion.SelectAsync(1);
    ExpandCardItem = false;
    ExpandDeliveryItem = true;
}
public async Task SaveDetails()
{
    if (CardNumberTextbox == null || CardHolderNameTextbox == null || CvvTextbox
    == null)
    {
        EmptyField = true;
    }
    else
    {
        EmptyField = false;
        await AlertDialog.ShowAsync();
    }
}
public async Task DialogCreate()
{
    await AlertDialog.HideAsync();
}
public async Task OnSubmit(Object args)
{
    await AlertDialog.HideAsync();
    ExpandCardItem = false;
    DisableSignInItem = false;
    DisableDeliveryItem = true;
    DisableCardItem = true;
    await Accordion.SelectAsync(0);
}
}
<style>
.Error {
color: red;
}
</style>
```

Output:

Online Shopping Payment Module

Sign In

Email

Password

CONTINUE

Delivery Address

Card Details

Treeview Integration in Blazor Accordion Component

Accordion supports to render other blazor Components by using the [ContentTemplate](#) property. The other components are given an accordion content like the following, for initializing the component.

ASPX-CS

```
<ContentTemplate>
<SfTreeView TValue="TreeviewItem"></SfTreeView>
</ContentTemplate>
```

The following example shows how to render a TreeView within the Accordion.

ASPX-CS

```
@using Syncfusion.Blazor.Navigations
<SfAccordion>
<AccordionItems>
<AccordionItem>
<HeaderTemplate>Documents</HeaderTemplate>
<ContentTemplate>
<SfTreeView TValue="TreeviewItem">
<TreeViewFieldsSettings TValue="TreeviewItem" Id="Id"
DataSource="@DocumentFolder" Text="FolderName" ParentID="ParentId"
HasChildren="HasSubFolders" Expanded="Expanded"></TreeViewFieldsSettings>
</SfTreeView>
</ContentTemplate>
</AccordionItem>
<AccordionItem>
<HeaderTemplate>Downloads</HeaderTemplate>
<ContentTemplate>
<SfTreeView TValue="TreeviewItem">
<TreeViewFieldsSettings TValue="TreeviewItem" Id="Id"
DataSource="@DownloadFolder" Text="FolderName" ParentID="ParentId"
HasChildren="HasSubFolders" Expanded="Expanded"></TreeViewFieldsSettings>
</SfTreeView>
</ContentTemplate>
```

```

</AccordionItem>
<AccordionItem>
<HeaderTemplate>Pictures</HeaderTemplate>
<ContentTemplate>
<SfTreeView TValue="TreeviewItem">
<TreeViewFieldsSettings TValue="TreeviewItem" Id="Id"
DataSource="@PictureFolder" Text="FolderName" ParentID="ParentId"
HasChildren="HasSubFolders" Expanded="Expanded"></TreeViewFieldsSettings>
</SfTreeView>
</ContentTemplate>
</AccordionItem>
</AccordionItems>
</SfAccordion>
@code{
public class TreeviewItem
{
public string Id { get; set; }
public string ParentId { get; set; }
public string FolderName { get; set; }
public bool Expanded { get; set; }
public bool HasSubFolders { get; set; }
}
List<TreeviewItem> DocumentFolder = new List<TreeviewItem>();
List<TreeviewItem> DownloadFolder = new List<TreeviewItem>();
List<TreeviewItem> PictureFolder = new List<TreeviewItem>();
protected override void OnInitialized()
{
base.OnInitialized();
DocumentFolder.Add(new TreeviewItem
{
Id = "1",
FolderName = "Documents",
HasSubFolders = true,
Expanded = false
});
DocumentFolder.Add(new TreeviewItem
{
Id = "2",
ParentId = "1",
FolderName = "Environment Pollution.docx"
});
DocumentFolder.Add(new TreeviewItem
{
Id = "3",
ParentId = "1",
FolderName = "Global Water, Sanitation, & Hygiene.docx"
});
DocumentFolder.Add(new TreeviewItem
{
Id = "4",
ParentId = "1",
FolderName = "Global Warming.ppt"
});
DocumentFolder.Add(new TreeviewItem
{
Id = "5",
ParentId = "1",

```



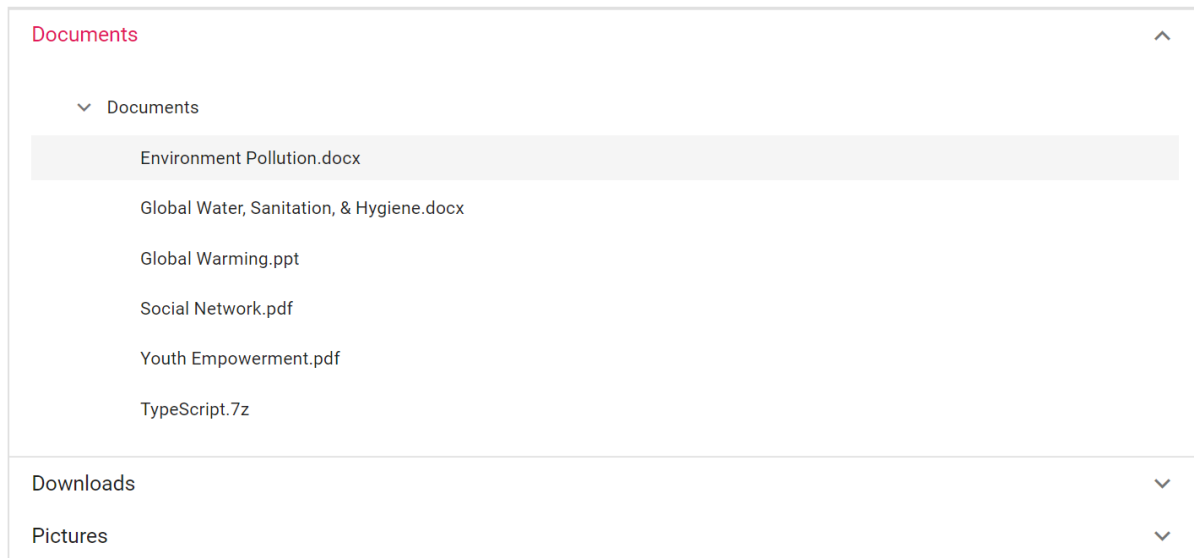
```

FolderName = "Social Network.pdf"
});
DocumentFolder.Add(new TreeViewItem
{
    Id = "6",
    ParentId = "1",
    FolderName = "Youth Empowerment.pdf"
});
DownloadFolder.Add(new TreeViewItem
{
    Id = "1",
    FolderName = "Downloads",
    HasSubFolders = true,
    Expanded = false
});
DownloadFolder.Add(new TreeViewItem
{
    Id = "2",
    ParentId = "1",
    FolderName = "UI-Guide.pdf"
});
DownloadFolder.Add(new TreeViewItem
{
    Id = "3",
    ParentId = "1",
    FolderName = "Tutorials.zip"
});
DownloadFolder.Add(new TreeViewItem
{
    Id = "4",
    ParentId = "1",
    FolderName = "Game.exe"
});
DocumentFolder.Add(new TreeViewItem
{
    Id = "5",
    ParentId = "1",
    FolderName = "TypeScript.7z"
});
PictureFolder.Add(new TreeViewItem
{
    Id = "1",
    FolderName = "Pictures",
    HasSubFolders = true,
    Expanded = true
});
PictureFolder.Add(new TreeViewItem
{
    Id = "2",
    ParentId = "1",
    FolderName = "Camera Roll",
    HasSubFolders = true,
});
PictureFolder.Add(new TreeViewItem
{
    Id = "3",
    ParentId = "2",

```

```
FolderName = "WIN_20160726_094117.JPG"
});
PictureFolder.Add(new TreeViewItem
{
    Id = "4",
    ParentId = "2",
    FolderName = "WIN_20160726_094118.JPG"
});
PictureFolder.Add(new TreeViewItem
{
    Id = "5",
    ParentId = "1",
    FolderName = "Wind.jpg"
});
PictureFolder.Add(new TreeViewItem
{
    Id = "6",
    ParentId = "1",
    FolderName = "Stone.jpg"
});
}
}
```

Output:



Accumulation Chart

<!-- markdownlint-disable MD040 -->

Blazor Accumulation Chart in Server Side App using Visual Studio

This section briefly explains about how to include a [Accumulation Chart](#) in the Blazor Server-Side application. Refer [Getting Started with Syncfusion Blazor for Server-Side in Visual Studio](#) page for the introduction and configuring the common specifications.

Importing Syncfusion Blazor component in the application

1. Install the [Syncfusion.Blazor](#) NuGet package to the application by using the [NuGet Package Manager](#).
2. Add the client-side resources through [CDN](#) or from [NuGet](#) package in the **HEAD** element of the `~/Pages/_Host.cshtml` page. For Internet Explorer 11, kindly refer the polyfills. Refer the [documentation](#) for more information.

HTML

```
<head>
<environment include="Development">
....
....
<script
src="https://github.com/Daddoon/Blazor.Polyfill/releases/download/3.0.1/blaz
or.polyfill.min.js"></script>
</environment>
</head>
```

3. Now add the lodash script to the **HEAD** element of the `/Pages/Host.cshtml` page, since it is been used in the [chart interactive](#) features. The absence of the script will result in console errors.

HTML

```
<head>
<environment include="Development">
....
....
<script
src="https://github.com/Daddoon/Blazor.Polyfill/releases/download/3.0.1/blaz
or.polyfill.min.js"></script>
<script
src="https://cdnjs.cloudflare.com/ajax/libs/lodash.js/4.17.20/lodash.min.js"
></script>
</environment>
</head>
```

Adding component package to the application

Open `~/_Imports.razor` file and include the **Syncfusion.Blazor** namespaces.

ASPX-CS

```
@using Syncfusion.Blazor
@using Syncfusion.Blazor.Charts
```

Add SyncfusionBlazor service in Startup.cs

Open the **Startup.cs** file and add services required by Syncfusion components using **service.AddSyncfusionBlazor()** method. Add this method in the **ConfigureServices** function as follows.

C#

```
using Syncfusion.Blazor;  
namespace BlazorApplication  
{  
    public class Startup  
    {  
        ....  
        ....  
        public void ConfigureServices(IServiceCollection services)  
        {  
            ....  
            ....  
            services.AddSyncfusionBlazor();  
        }  
    }  
}
```

During initial loading, collect and send the individual character size information in-order to render the chart. To avoid any disconnection, increase the buffer size to 64 KB or more over the SignalR connection.

C#

```
using Syncfusion.Blazor;  
namespace BlazorApplication  
{  
    public class Startup  
    {  
        ....  
        ....  
        public void ConfigureServices(IServiceCollection services)  
        {  
            ....  
            ....  
            services.AddSyncfusionBlazor();  
            services.AddSignalR(e => {  
                e.MaximumReceiveMessageSize = 65536;  
            });  
        }  
    }  
}
```

Use the following configuration to host the Blazor server application on **Azure SignalR**.

C#

```
using Syncfusion.Blazor;  
namespace BlazorApplication  
{  
    public class Startup  
    {  
        ....  
        ....  
        public void ConfigureServices(IServiceCollection services)  
        {  
            ....  
            ....  
            services.AddSyncfusionBlazor();  
        }  
    }  
}
```

```
services.AddSignalR(e => {e.MaximumReceiveMessageSize =  
65536;}).AddAzureSignalR();  
}  
}  
}
```

Add Accumulation Chart

To initialize the accumulation chart component add the following code to the **Index.razor** view page under **~/Pages** folder. In a new application, if the **Index.razor** page has any default content template, then those content can be completely removed and the following code can be added.

ASPX-CS

```
@page "/"  
@using Syncfusion.Blazor.Charts  
<SfAccumulationChart>  
<AccumulationChartSeriesCollection>  
<AccumulationChartSeries DataSource="@MedalDetails" XName="Country"  
YName="Medals">  
</AccumulationChartSeries>  
</AccumulationChartSeriesCollection>  
</SfAccumulationChart>  
@code{  
public class ChartData  
{  
public string Country { get; set; }  
public double Medals { get; set; }  
}  
public List<ChartData> MedalDetails = new List<ChartData>  
{  
new ChartData { Country= "United States of America", Medals= 46 },  
new ChartData { Country= "Great Britain", Medals= 27 },  
new ChartData { Country= "China", Medals= 26 },  
new ChartData { Country= "United Kingdom", Medals= 23 },  
new ChartData { Country= "Australia", Medals= 16 },  
new ChartData { Country= "India", Medals= 36 },  
new ChartData { Country= "Nigeria", Medals= 12 },  
new ChartData { Country= "Brazil", Medals= 20 },  
};  
}
```

On successful compilation of the application, the Syncfusion Blazor Accumulation Chart component will render in the web browser.



Add Title

Using the [Title](#) property, a title to the accumulation chart is added to provide the user with quick information about the data plotted in the chart.

ASPX-CS

```
@page "/"
@using Syncfusion.Blazor.Charts
<SfAccumulationChart Title="Olympic Medal Details">
  <AccumulationChartSeriesCollection>
    <AccumulationChartSeries DataSource="@MedalDetails" XName="Country"
      YName="Medals">
    </AccumulationChartSeries>
  </AccumulationChartSeriesCollection>
</SfAccumulationChart>
@code{
  public class ChartData
  {
    public string Country { get; set; }
    public double Medals { get; set; }
  }
  public List<ChartData> MedalDetails = new List<ChartData>
  {
    new ChartData { Country= "United States of America", Medals= 46 },
    new ChartData { Country= "Great Britain", Medals= 27 },
    new ChartData { Country= "China", Medals= 26 },
    new ChartData { Country= "United Kingdom", Medals= 23 },
    new ChartData { Country= "Australia", Medals= 16 },
    new ChartData { Country= "India", Medals= 36 },
    new ChartData { Country= "Nigeria", Medals= 12 },
    new ChartData { Country= "Brazil", Medals= 20 },
  };
}
```

```
}
```

Olympic Medal Details



Add Data Label

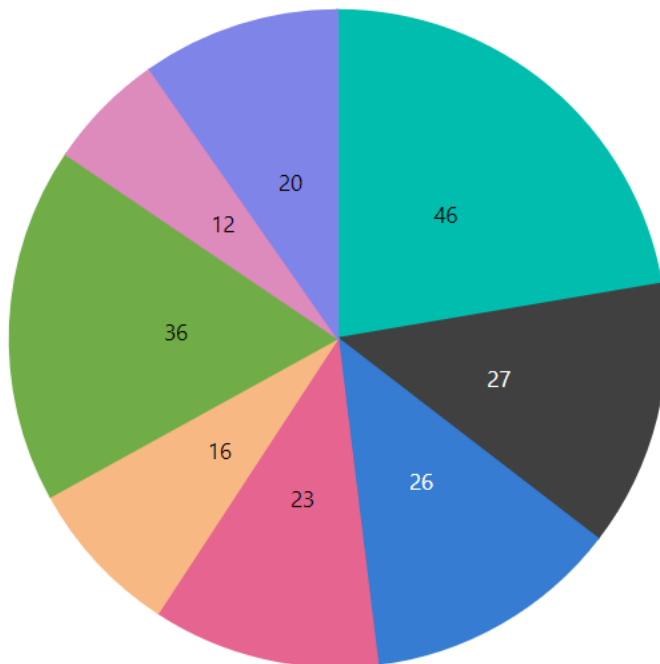
The data labels are added to improve the readability of the accumulation chart. This can be achieved by setting the [Visible](#) property to **true** in the [AccumulationDataLabelSettings](#).

ASPX-CS

```
@page "/"
@using Syncfusion.Blazor.Charts
<SfAccumulationChart Title="Olympic Medal Details">
  <AccumulationChartSeriesCollection>
    <AccumulationChartSeries DataSource="@MedalDetails" XName="Country"
    YName="Medals">
      <AccumulationDataLabelSettings Visible="true"/>
    </AccumulationChartSeries>
  </AccumulationChartSeriesCollection>
</SfAccumulationChart>
@code{
public class ChartData
{
    public string Country { get; set; }
    public double Medals { get; set; }
}
public List<ChartData> MedalDetails = new List<ChartData>
{
    new ChartData { Country= "United States of America", Medals= 46 },
    new ChartData { Country= "Great Britain", Medals= 27 },
}
```

```
new ChartData { Country= "China", Medals= 26 },  
new ChartData { Country= "United Kingdom", Medals= 23 },  
new ChartData { Country= "Australia", Medals= 16 },  
new ChartData { Country= "India", Medals= 36 },  
new ChartData { Country= "Nigeria", Medals= 12 },  
new ChartData { Country= "Brazil", Medals= 20 },  
};  
}
```

Olympic Medal Details



Enable Tooltip

When space constraints prevent from displaying the information using data labels, the tooltip comes in handy. The tooltip can be enabled by setting the [Enable](#) property in [AccumulationChartTooltipSettings](#) to **true**.

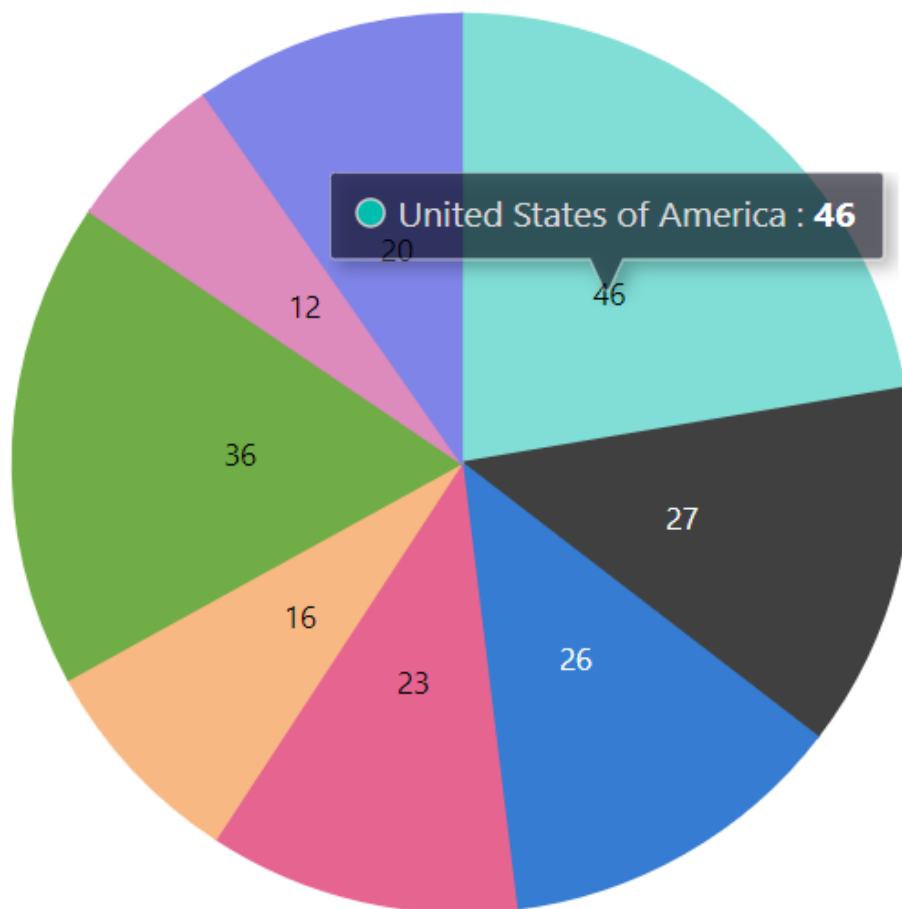
ASPX-CS

```
@page "/"  
@using Syncfusion.Blazor.Charts  
<SfAccumulationChart Title="Olympic Medal Details">  
  <AccumulationChartSeriesCollection>  
    <AccumulationChartSeries DataSource="@MedalDetails" XName="Country"  
      YName="Medals">  
    </AccumulationChartSeries>  
  </AccumulationChartSeriesCollection>  
  <AccumulationChartTooltipSettings  
    Enable="true"></AccumulationChartTooltipSettings>  
</SfAccumulationChart>  
@code{
```



```
public class ChartData
{
    public string Country { get; set; }
    public double Medals { get; set; }
}
public List<ChartData> MedalDetails = new List<ChartData>
{
    new ChartData { Country= "United States of America", Medals= 46 },
    new ChartData { Country= "Great Britain", Medals= 27 },
    new ChartData { Country= "China", Medals= 26 },
    new ChartData { Country= "United Kingdom", Medals= 23 },
    new ChartData { Country= "Australia", Medals= 16 },
    new ChartData { Country= "India", Medals= 36 },
    new ChartData { Country= "Nigeria", Medals= 12 },
    new ChartData { Country= "Brazil", Medals= 20 },
};
```

Olympic Medal Details



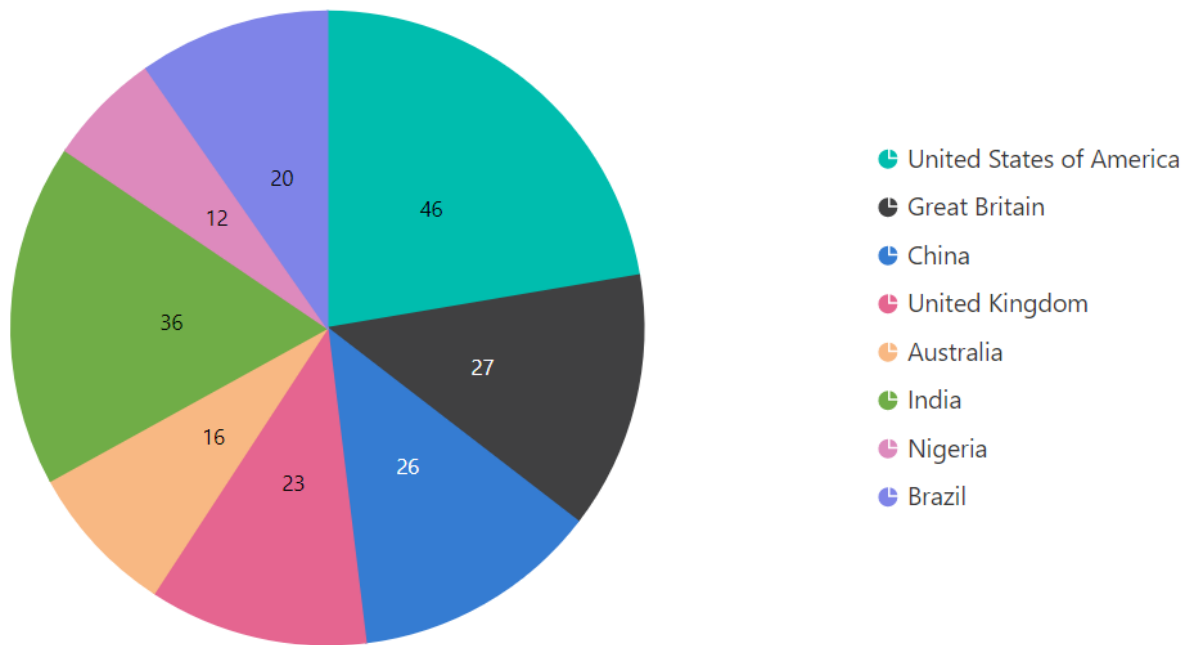
Enable Legend

Legend is used for the accumulation chart by setting the [Visible](#) property to **true** in the [AccumulationChartLegendSettings](#).

ASPX-CS

```
@page "/"
@using Syncfusion.Blazor.Charts
<SfAccumulationChart Title="Olympic Medal Details">
  <AccumulationChartSeriesCollection>
    <AccumulationChartSeries DataSource="@MedalDetails" XName="Country"
    YName="Medals">
      <AccumulationDataLabelSettings
        Visible="true"></AccumulationDataLabelSettings>
    </AccumulationChartSeries>
  </AccumulationChartSeriesCollection>
  <AccumulationChartTooltipSettings
    Enable="true"></AccumulationChartTooltipSettings>
  <AccumulationChartLegendSettings
    Visible="true"></AccumulationChartLegendSettings>
</SfAccumulationChart>
@code{
  public class ChartData
  {
    public string Country { get; set; }
    public double Medals { get; set; }
  }
  public List<ChartData> MedalDetails = new List<ChartData>
  {
    new ChartData { Country= "United States of America", Medals= 46 },
    new ChartData { Country= "Great Britain", Medals= 27 },
    new ChartData { Country= "China", Medals= 26 },
    new ChartData { Country= "United Kingdom", Medals= 23 },
    new ChartData { Country= "Australia", Medals= 16 },
    new ChartData { Country= "India", Medals= 36 },
    new ChartData { Country= "Nigeria", Medals= 12 },
    new ChartData { Country= "Brazil", Medals= 20 },
  };
}
```

Olympic Medal Details



Refer to the fully working sample for accumulation chart [here](#).

See also

- [Getting Started with Syncfusion Blazor for Client-Side in .NET Core CLI](#)
- [Getting Started with Syncfusion Blazor for Server-Side in Visual Studio](#)
- [Getting Started with Syncfusion Blazor for Server-Side in .NET Core CLI](#)

Chart Types

Pie and Doughnut in Blazor Accumulation Chart Component

Pie Chart

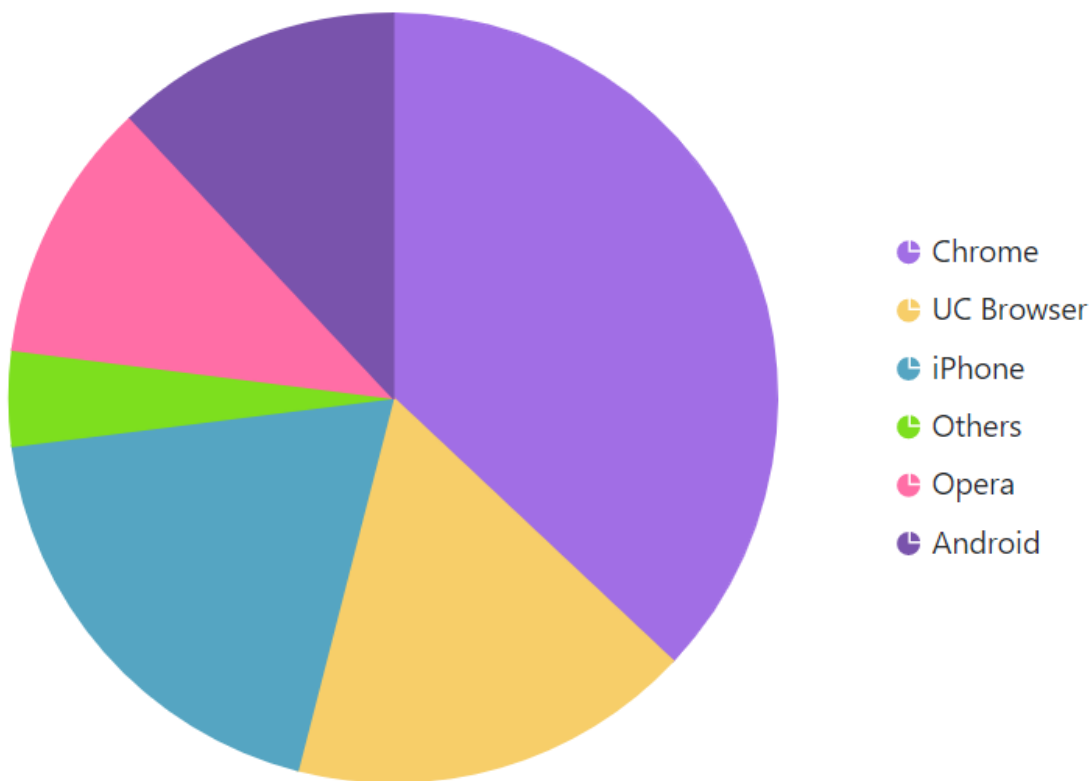
The [Pie Chart](#) is used to represent numeric proportional data in divided slices. To render a [Pie Chart](#), set the series [Type](#) as [Pie](#).

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfAccumulationChart Title="Mobile Browser Statistics">
  <AccumulationChartSeriesCollection>
    <AccumulationChartSeries DataSource="@StatisticsDetails" XName="Browser"
      YName="Users"
      Name="Browser">
    </AccumulationChartSeries>
  </AccumulationChartSeriesCollection>
  <AccumulationChartLegendSettings
    Visible="true"></AccumulationChartLegendSettings>
</SfAccumulationChart>
@code{
```

```
public class Statistics
{
    public string Browser { get; set; }
    public double Users { get; set; }
}
public List<Statistics> StatisticsDetails = new List<Statistics>
{
    new Statistics { Browser = "Chrome", Users = 37 },
    new Statistics { Browser = "UC Browser", Users = 17 },
    new Statistics { Browser = "iPhone", Users = 19 },
    new Statistics { Browser = "Others", Users = 4 },
    new Statistics { Browser = "Opera", Users = 11 },
    new Statistics { Browser = "Android", Users = 12 },
};
}
```

Mobile Browser Statistics



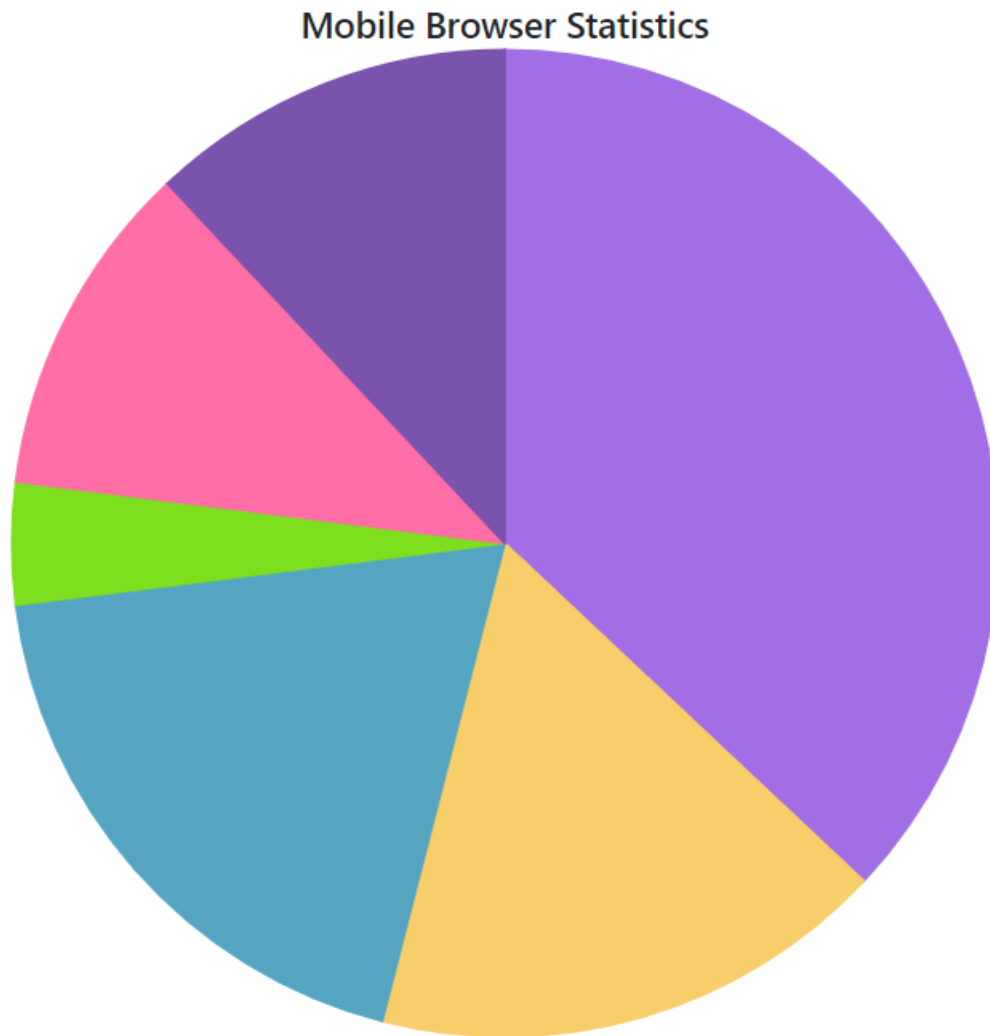
Radius Customization

The radius of the pie series will be set to 80% of its size (minimum of chart width and height) by default. The [Radius](#) property of the series can be used to customize the radius of the pie chart.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfAccumulationChart Title="Mobile Browser Statistics">
  <AccumulationChartSeriesCollection>
    <AccumulationChartSeries DataSource="@StatisticsDetails" XName="Browser"
      YName="Users"
      Name="Browser" Radius="100%">
    </AccumulationChartSeries>
  </AccumulationChartSeriesCollection>
  <AccumulationChartLegendSettings
    Visible="false"></AccumulationChartLegendSettings>
</SfAccumulationChart>

@code{
public class Statistics
{
public string Browser { get; set; }
public double Users { get; set; }
}
public List<Statistics> StatisticsDetails = new List<Statistics>
{
new Statistics { Browser = "Chrome", Users = 37 },
new Statistics { Browser = "UC Browser", Users = 17 },
new Statistics { Browser = "iPhone", Users = 19 },
new Statistics { Browser = "Others", Users = 4 },
new Statistics { Browser = "Opera", Users = 11 },
new Statistics { Browser = "Android", Users = 12 },
};
}
```



Pie Center

The center x and center y can be used to change the pie's center position. The pie series' center x and center y are set to 50% by default. The [AccumulationChartCenter](#) property of the series can be used to customize this.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfAccumulationChart EnableAnimation="false" Title="Mobile Browser
Statistics">
  <AccumulationChartCenter X="70%" Y="60%" />
  <AccumulationChartSeriesCollection>
    <AccumulationChartSeries DataSource="@StatisticsDetails" XName="Browser"
YName="Users" />
  </AccumulationChartSeriesCollection>
  <AccumulationChartLegendSettings Visible="false" />
</SfAccumulationChart>
@code{
public class Statistics
{
  public string Browser { get; set; }
}
```

```

public double Users { get; set; }
}
public List<Statistics> StatisticsDetails = new List<Statistics>
{
    new Statistics { Browser = "Chrome", Users = 37 },
    new Statistics { Browser = "UC Browser", Users = 17 },
    new Statistics { Browser = "iPhone", Users = 19 },
    new Statistics { Browser = "Others", Users = 4 },
    new Statistics { Browser = "Opera", Users = 11 },
    new Statistics { Browser = "Android", Users = 12 },
};
}

```

Mobile Browser Statistics



Various Radius Pie Chart

The [Radius](#) mapping can be used to render the slice with different radius.

ASPX-CS

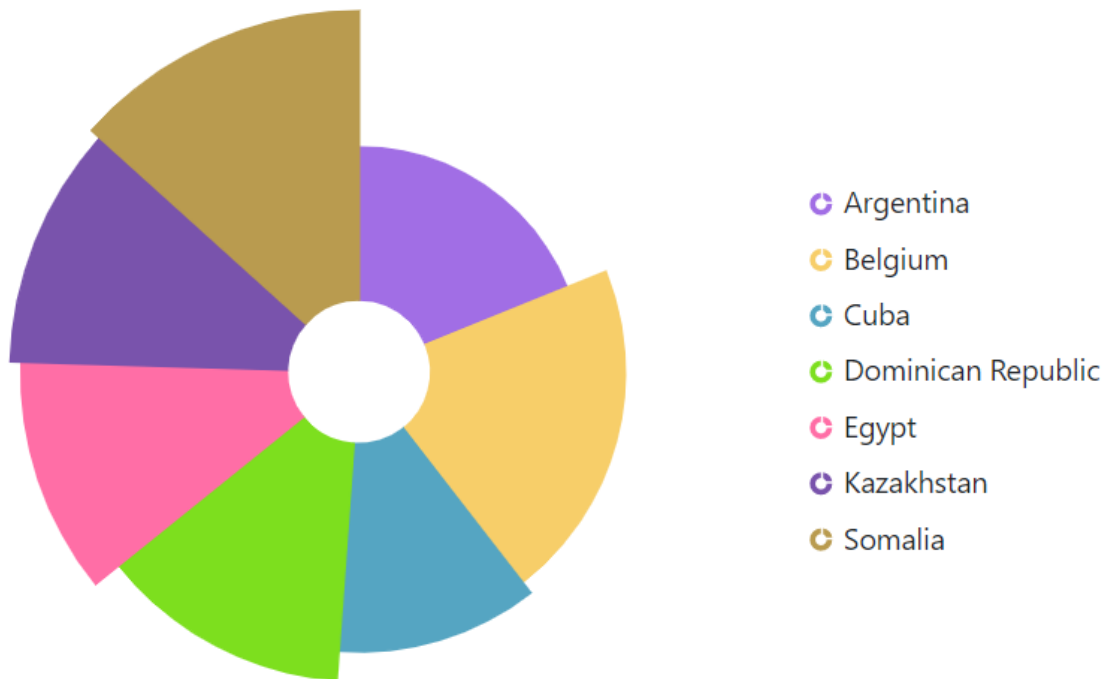
```

@using Syncfusion.Blazor.Charts
<SfAccumulationChart Title="Oil and other liquid imports in USA"
    EnableAnimation="true" EnableSmartLabels="true">

```

```
<AccumulationChartLegendSettings
Visible="true"></AccumulationChartLegendSettings>
<AccumulationChartSeriesCollection>
<AccumulationChartSeries DataSource="@StatisticsDetails" XName="Browser"
YName="Users"
InnerRadius="20%" Radius="R">
</AccumulationChartSeries>
</AccumulationChartSeriesCollection>
</SfAccumulationChart>
@code{
public class Statistics
{
public string Browser { get; set; }
public double Users { get; set; }
public string R { get; set; }
}
public List<Statistics> StatisticsDetails = new List<Statistics>
{
new Statistics { Browser = "Argentina", Users = 505370, R = "100"},
new Statistics { Browser = "Belgium", Users = 551500, R = "118.7"},
new Statistics { Browser = "Cuba", Users = 312685, R = "124.6"},
new Statistics { Browser = "Dominican Republic", Users = 350000, R =
"137.5"},
new Statistics { Browser = "Egypt", Users = 301000, R = "150.8"},
new Statistics { Browser = "Kazakhstan", Users = 300000, R = "155.5"},
new Statistics { Browser = "Somalia", Users = 357022, R = "160.6"}
};
}
```


Oil and other liquid imports in USA

*Doughnut Chart*

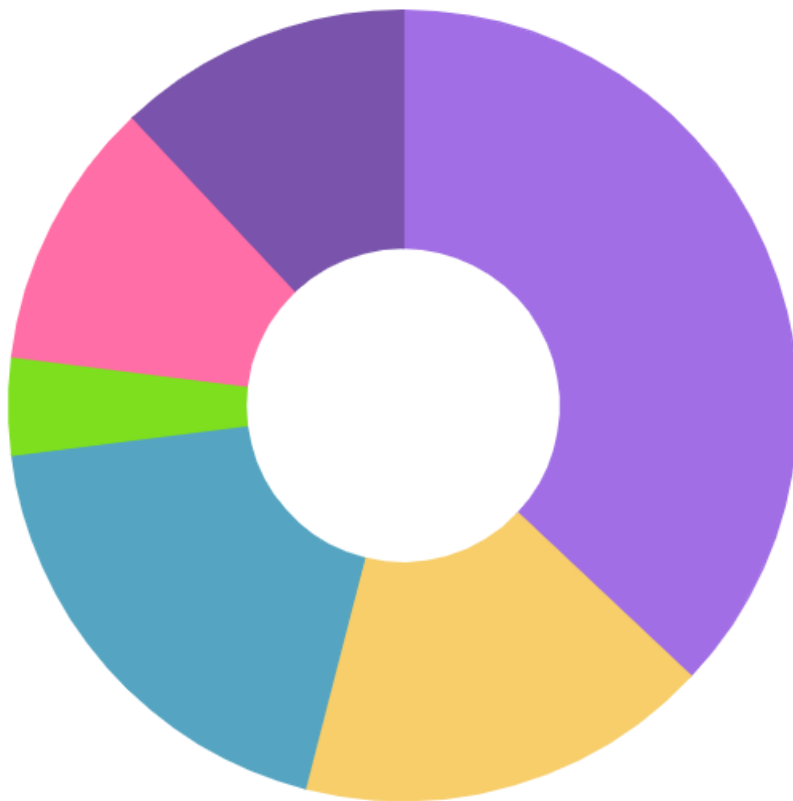
The doughnut chart can be created by setting the [InnerRadius](#) property of the [Pie Chart](#) to a value ranging from 0% to 100%.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfAccumulationChart Title="Mobile Browser Statistics">
  <AccumulationChartSeriesCollection>
    <AccumulationChartSeries DataSource="@StatisticsDetails" XName="Browser"
      YName="Users"
      Name="Browser" InnerRadius="40%">
    </AccumulationChartSeries>
  </AccumulationChartSeriesCollection>
  <AccumulationChartLegendSettings
    Visible="false"></AccumulationChartLegendSettings>
</SfAccumulationChart>
@code{
public class Statistics
{
  public string Browser { get; set; }
  public double Users { get; set; }
}
public List<Statistics> StatisticsDetails = new List<Statistics>
{
  new Statistics { Browser = "Chrome", Users = 37 },
```

```
new Statistics { Browser = "UC Browser", Users = 17 },  
new Statistics { Browser = "iPhone", Users = 19 },  
new Statistics { Browser = "Others", Users = 4 },  
new Statistics { Browser = "Opera", Users = 11 },  
new Statistics { Browser = "Android", Users = 12 },  
};  
}
```

Mobile Browser Statistics



Start and End angles

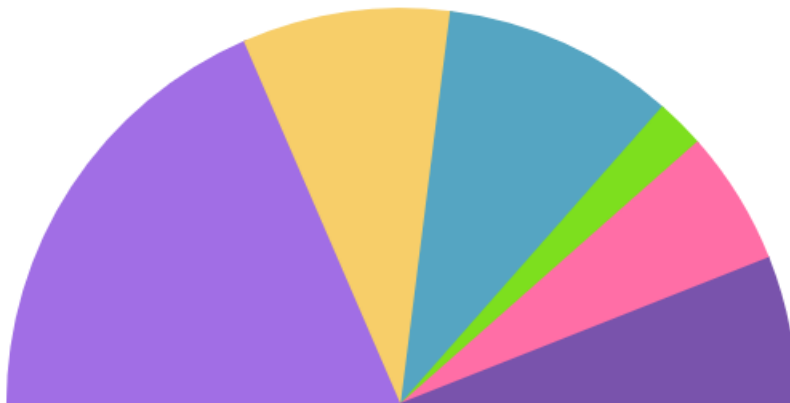
The [StartAngle](#) and [EndAngle](#) properties can be used to customize the start and end angles of the pie series. StartAngle is set to 0 degrees by default, and EndAngle is set to 360 degrees by default. Semi-pie series can be achieved by customizing these properties.

ASPX-CS

```
@using Syncfusion.Blazor.Charts  
<SfAccumulationChart Title="Mobile Browser Statistics">  
  <AccumulationChartSeriesCollection>  
    <AccumulationChartSeries DataSource="@StatisticsDetails" XName="Browser"  
      YName="Users"  
      StartAngle="270" EndAngle="90">
```

```
</AccumulationChartSeries>
</AccumulationChartSeriesCollection>
<AccumulationChartLegendSettings Visible="false">
</AccumulationChartLegendSettings>
</SfAccumulationChart>
@code{
public class Statistics
{
public string Browser { get; set; }
public double Users { get; set; }
}
public List<Statistics> StatisticsDetails = new List<Statistics>
{
new Statistics { Browser = "Chrome", Users = 37 },
new Statistics { Browser = "UC Browser", Users = 17 },
new Statistics { Browser = "iPhone", Users = 19 },
new Statistics { Browser = "Others", Users = 4 },
new Statistics { Browser = "Opera", Users = 11 },
new Statistics { Browser = "Android", Users = 12 },
};
}
```

Mobile Browser Statistics



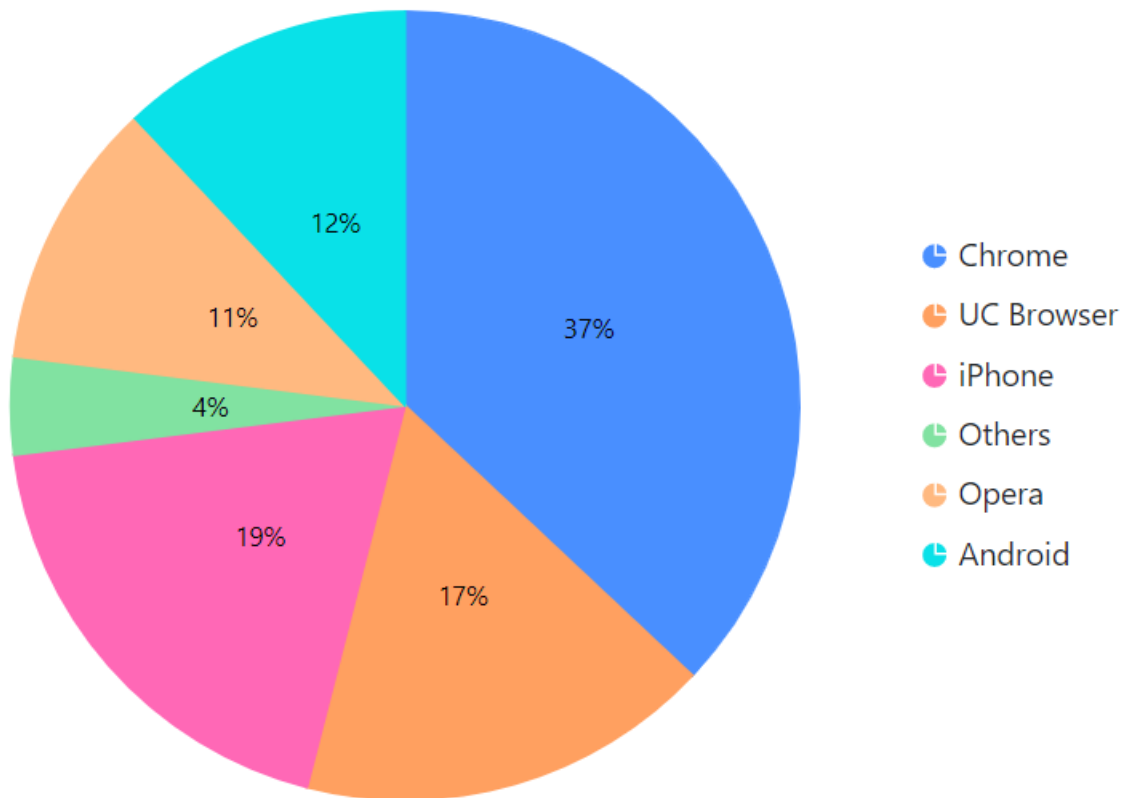
Color and Text Mapping

[PointColorMapping](#) in series and [Name](#) in datalabel can be used to map the fill color and text from the data source to the chart.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfAccumulationChart Title="Mobile Browser Statistics">
  <AccumulationChartLegendSettings
    Visible="true"></AccumulationChartLegendSettings>
  <AccumulationChartSeriesCollection>
    <AccumulationChartSeries DataSource="@StatisticsDetails" XName="Browser"
      YName="Users" Name="Browser" PointColorMapping="Fill">
      <AccumulationDataLabelSettings Visible="true"
        Name="Text"></AccumulationDataLabelSettings>
    </AccumulationChartSeries>
  </AccumulationChartSeriesCollection>
</SfAccumulationChart>
@code{
public class Statistics
{
    public string Browser { get; set; }
    public double Users { get; set; }
    public string Fill { get; set; }
    public string Text { get; set; }
}
public List<Statistics> StatisticsDetails = new List<Statistics>
{
    new Statistics { Browser = "Chrome", Users = 37, Text= "37%", Fill="#498fff"
    },
    new Statistics { Browser = "UC Browser", Users = 17, Text= "17%",
    Fill="#ffa060" },
    new Statistics { Browser = "iPhone", Users = 19, Text= "19%", Fill="#ff68b6"
    },
    new Statistics { Browser = "Others", Users = 4, Text= "4%", Fill="#81e2a1"
    },
    new Statistics { Browser = "Opera", Users = 11, Text= "11%", Fill="#ffb980"
    },
    new Statistics { Browser = "Android", Users = 12, Text= "12%",
    Fill="#09e1e8" },
};
}
```

Mobile Browser Statistics

*Hide pie or doughnut border*

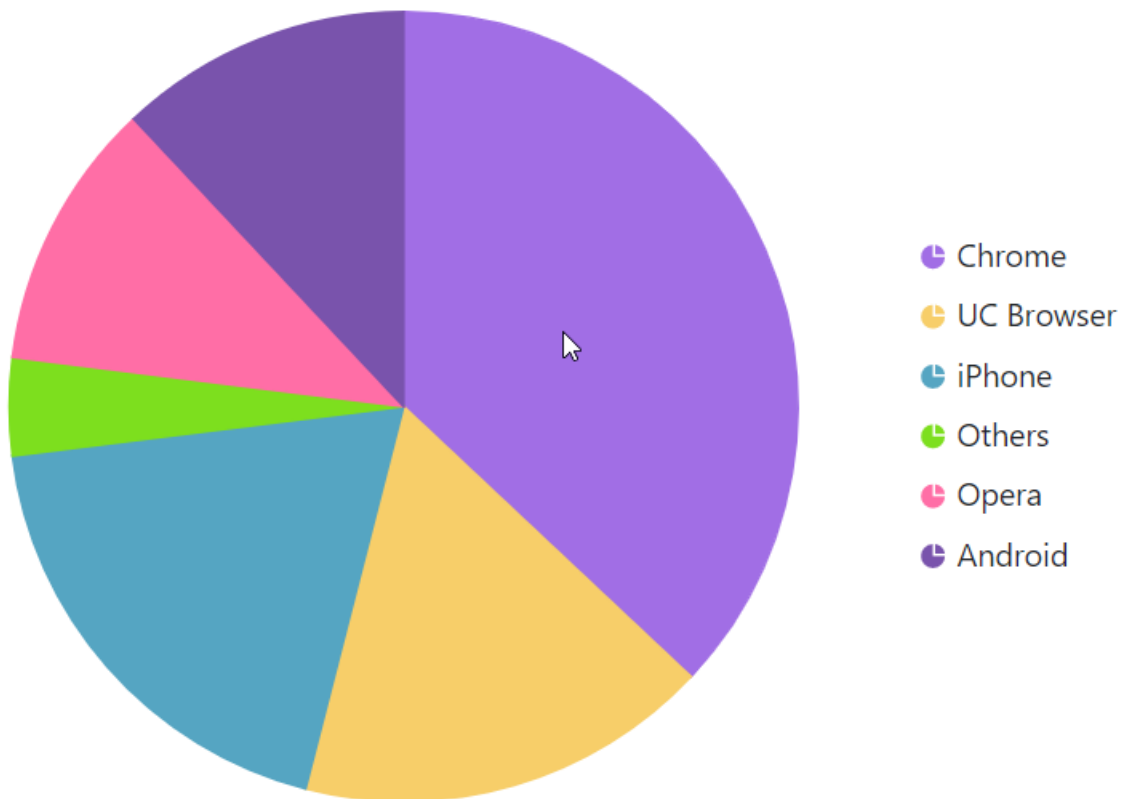
When the mouse hovers over the pie/doughnut chart, the border appears by default. The border can be turned off by setting the [EnableBorderOnMouseMove](#) property to **false**.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfAccumulationChart Title="Mobile Browser Statistics"
EnableBorderOnMouseMove="false">
  <AccumulationChartLegendSettings
Visible="true"></AccumulationChartLegendSettings>
  <AccumulationChartSeriesCollection>
    <AccumulationChartSeries DataSource="@StatisticsDetails" XName="Browser"
YName="Users" Name="Browser">
    </AccumulationChartSeries>
  </AccumulationChartSeriesCollection>
</SfAccumulationChart>
@code{
public class Statistics
{
    public string Browser { get; set; }
    public double Users { get; set; }
}
```

```
}  
public List<Statistics> StatisticsDetails = new List<Statistics>  
{  
    new Statistics { Browser = "Chrome", Users = 37 },  
    new Statistics { Browser = "UC Browser", Users = 17 },  
    new Statistics { Browser = "iPhone", Users = 19 },  
    new Statistics { Browser = "Others", Users = 4 },  
    new Statistics { Browser = "Opera", Users = 11 },  
    new Statistics { Browser = "Android", Users = 12 },  
};  
}
```

Mobile Browser Statistics



Refer to the [Blazor Charts](#) feature tour page for its groundbreaking feature representations and also explore the [Blazor Accumulation Chart Example](#) to know various features of accumulation charts and how it is used to represent numeric proportional data.

See Also

- [Data label](#)
- [Grouping](#)

Pyramid in Blazor Accumulation Chart Component

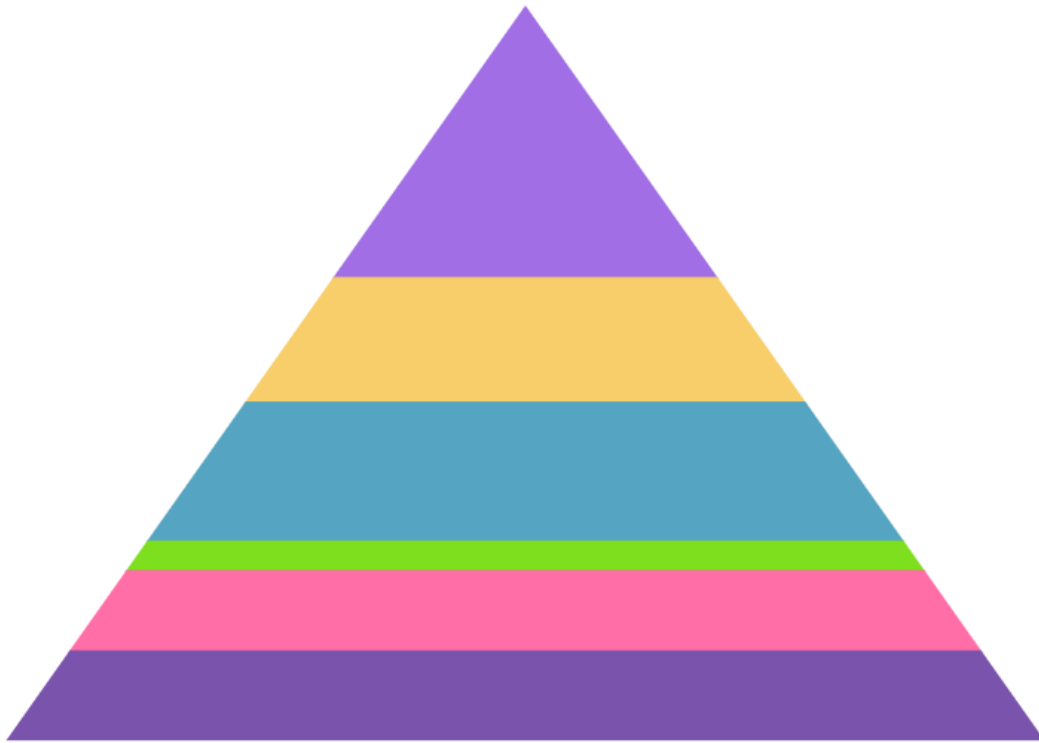
The [Pyramid Chart](#) used to visualize the hierarchical data in upside triangle shape with horizontally divided section. To render the [Pyramid Chart](#), set the series [Type](#) as [Pyramid](#).

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfAccumulationChart Title="Mobile Browser Statistics">
  <AccumulationChartSeriesCollection>
    <AccumulationChartSeries DataSource="@StatisticsDetails" XName="Browser"
      YName="Users" Name="Browser"
      Type="AccumulationType.Pyramid">
    </AccumulationChartSeries>
  </AccumulationChartSeriesCollection>
  <AccumulationChartLegendSettings
    Visible="false"></AccumulationChartLegendSettings>
</SfAccumulationChart>

@code{
public class Statistics
{
public string Browser { get; set; }
public double Users { get; set; }
}
public List<Statistics> StatisticsDetails = new List<Statistics>
{
new Statistics { Browser = "Chrome", Users = 37 },
new Statistics { Browser = "UC Browser", Users = 17 },
new Statistics { Browser = "iPhone", Users = 19 },
new Statistics { Browser = "Others", Users = 4 },
new Statistics { Browser = "Opera", Users = 11 },
new Statistics { Browser = "Android", Users = 12 },
};
}
```

Mobile Browser Statistics



Pyramid Mode

The Pyramid Chart can be rendered in both **Linear** or **Surface** modes by setting [PyramidMode](#) property. The default mode is **Linear**.

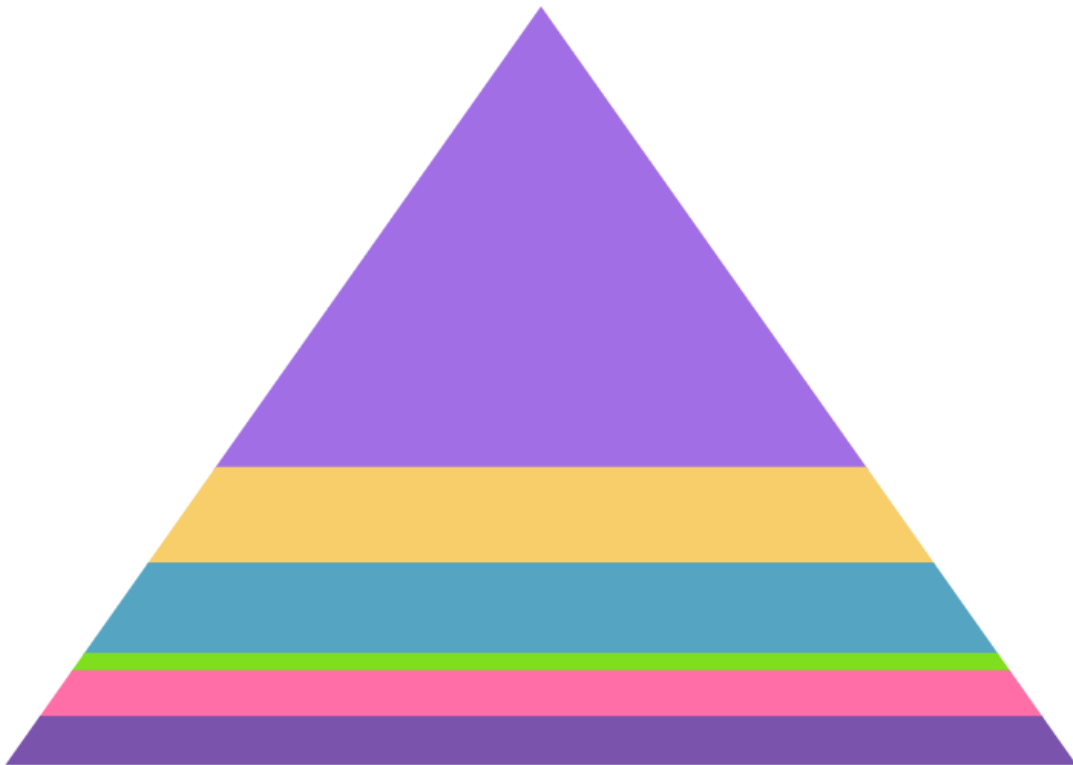
ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfAccumulationChart Title="Mobile Browser Statistics">
  <AccumulationChartSeriesCollection>
    <AccumulationChartSeries DataSource="@StatisticsDetails" XName="Browser"
      YName="Users" Name="Browser"
      Type="AccumulationType.Pyramid" PyramidMode="PyramidMode.Surface">
    </AccumulationChartSeries>
  </AccumulationChartSeriesCollection>
  <AccumulationChartLegendSettings
    Visible="false"></AccumulationChartLegendSettings>
</SfAccumulationChart>
@code{
public class Statistics
{
public string Browser { get; set; }
public double Users { get; set; }
}
public List<Statistics> StatisticsDetails = new List<Statistics>
{
```



```
new Statistics { Browser = "Chrome", Users = 37 },  
new Statistics { Browser = "UC Browser", Users = 17 },  
new Statistics { Browser = "iPhone", Users = 19 },  
new Statistics { Browser = "Others", Users = 4 },  
new Statistics { Browser = "Opera", Users = 11 },  
new Statistics { Browser = "Android", Users = 12 },  
};  
}
```

Mobile Browser Statistics



Pyramid Size

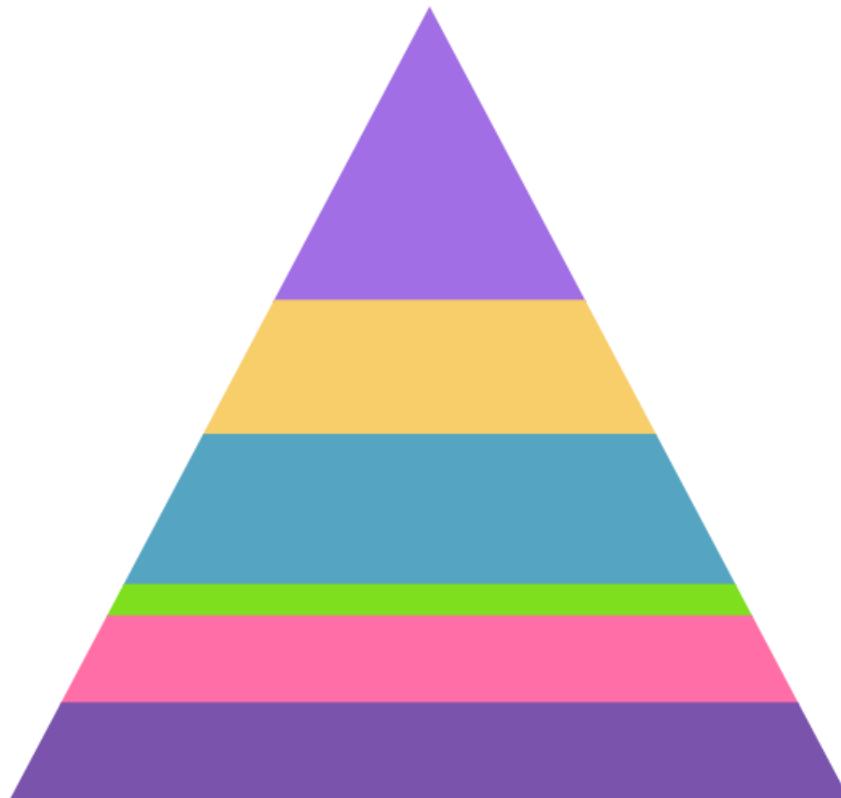
The size of the pyramid chart can be customized by using the [Width](#) and [Height](#) properties.

ASPX-CS

```
@using Syncfusion.Blazor.Charts  
<SfAccumulationChart Title="Mobile Browser Statistics">  
  <AccumulationChartSeriesCollection>  
    <AccumulationChartSeries DataSource="@StatisticsDetails" XName="Browser"  
      YName="Users"  
      Name="Browser" Type="AccumulationType.Pyramid" Width="60%" Height="80%">  
    </AccumulationChartSeries>  
  </AccumulationChartSeriesCollection>
```

```
<AccumulationChartLegendSettings
Visible="false"></AccumulationChartLegendSettings>
</SfAccumulationChart>
@code{
public class Statistics
{
public string Browser { get; set; }
public double Users { get; set; }
}
public List<Statistics> StatisticsDetails = new List<Statistics>
{
new Statistics { Browser = "Chrome", Users = 37 },
new Statistics { Browser = "UC Browser", Users = 17 },
new Statistics { Browser = "iPhone", Users = 19 },
new Statistics { Browser = "Others", Users = 4 },
new Statistics { Browser = "Opera", Users = 11 },
new Statistics { Browser = "Android", Users = 12 },
};
}
```

Mobile Browser Statistics



Gap Between Pyramid Segments

The [Pyramid Chart](#) provides options to customize the space between the segments by using the [GapRatio](#) property of the series. It accepts values ranging from 0 to 1.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfAccumulationChart Title="Mobile Browser Statistics">
  <AccumulationChartSeriesCollection>
    <AccumulationChartSeries DataSource="@StatisticsDetails" XName="Browser"
      YName="Users" Name="Browser"
      Type="AccumulationType.Pyramid" GapRatio="0.2">
    </AccumulationChartSeries>
  </AccumulationChartSeriesCollection>
  <AccumulationChartLegendSettings
    Visible="false"></AccumulationChartLegendSettings>
</SfAccumulationChart>
@code{
public class Statistics
{
  public string Browser { get; set; }
  public double Users { get; set; }
}
public List<Statistics> StatisticsDetails = new List<Statistics>
{
  new Statistics { Browser = "Chrome", Users = 37 },
  new Statistics { Browser = "UC Browser", Users = 17 },
  new Statistics { Browser = "iPhone", Users = 19 },
  new Statistics { Browser = "Others", Users = 4 },
  new Statistics { Browser = "Opera", Users = 11 },
  new Statistics { Browser = "Android", Users = 12 },
};
}
```

Mobile Browser Statistics



Pyramid Explode

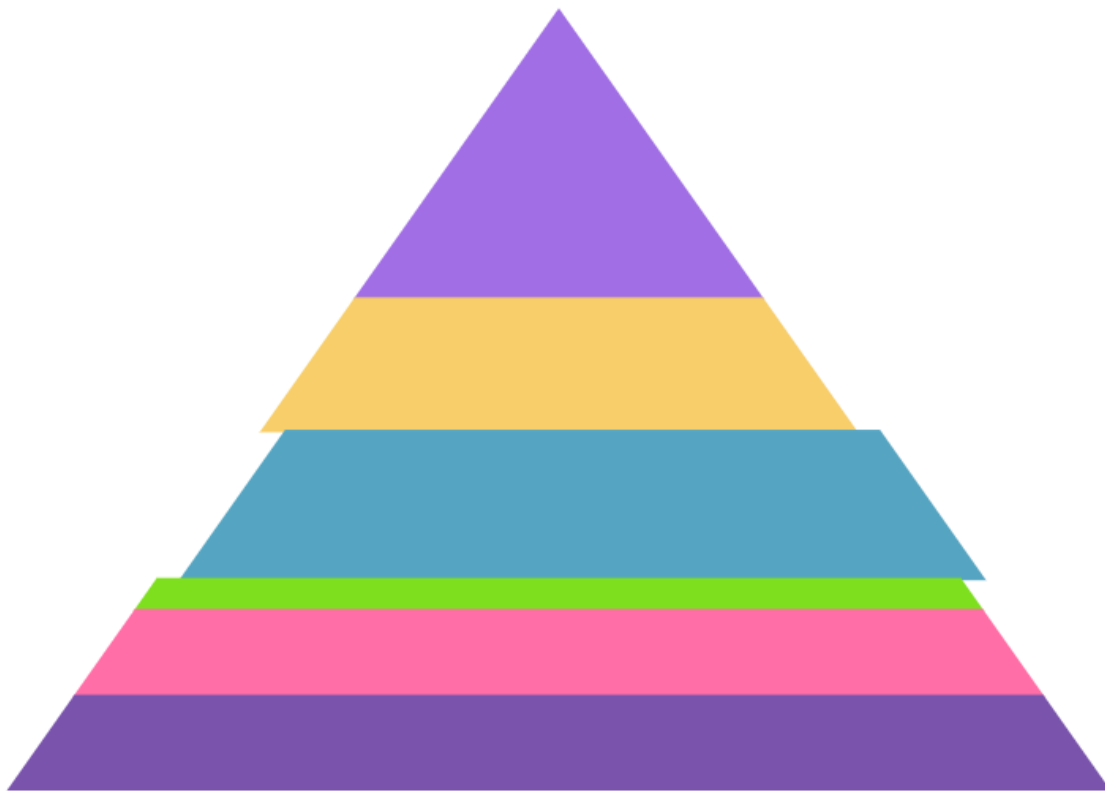
By setting the [Explode](#) property to **true**, points can be exploded on mouse click. Using the [ExplodeIndex](#) property, expand the point on load. The [ExplodeOffset](#) property can be used to set the distance between explosions.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfAccumulationChart Title="Mobile Browser Statistics">
  <AccumulationChartSeriesCollection>
    <AccumulationChartSeries DataSource="@StatisticsDetails" XName="Browser"
      YName="Users" Name="Browser"
      Type="AccumulationType.Pyramid" ExplodeIndex="2" Explode="true"
      ExplodeOffset="10">
    </AccumulationChartSeries>
  </AccumulationChartSeriesCollection>
  <AccumulationChartLegendSettings
    Visible="false"></AccumulationChartLegendSettings>
</SfAccumulationChart>
@code{
public class Statistics
{
  public string Browser { get; set; }
  public double Users { get; set; }
}
```

```
}  
public List<Statistics> StatisticsDetails = new List<Statistics>  
{  
    new Statistics { Browser = "Chrome", Users = 37 },  
    new Statistics { Browser = "UC Browser", Users = 17 },  
    new Statistics { Browser = "iPhone", Users = 19 },  
    new Statistics { Browser = "Others", Users = 4 },  
    new Statistics { Browser = "Opera", Users = 11 },  
    new Statistics { Browser = "Android", Users = 12 },  
};  
}
```

Mobile Browser Statistics



Refer to our [Blazor Charts](#) feature tour page for its groundbreaking feature representations and also explore our [Blazor Accumulation Chart Example](#) to know various features of accumulation charts and how it is used to represent numeric proportional data.

See Also

- [Data label](#)
- [Grouping](#)

Funnel in Blazor Accumulation Chart Component

[Funnel Chart](#) is often used to represent stages in a sales process and to show the amount of potential revenue for each stage. To render the [Funnel Chart](#), set the series [Type](#) as [Funnel](#).

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfAccumulationChart Title="Mobile Browser Statistics">
  <AccumulationChartSeriesCollection>
    <AccumulationChartSeries DataSource="@StatisticsDetails" XName="Browser"
    YName="Users"
    Name="Browser" Type="AccumulationType.Funnel">
  </AccumulationChartSeries>
</AccumulationChartSeriesCollection>
<AccumulationChartLegendSettings
  Visible="false"></AccumulationChartLegendSettings>
</SfAccumulationChart>
@code{
public class Statistics
{
  public string Browser { get; set; }
  public double Users { get; set; }
}
public List<Statistics> StatisticsDetails = new List<Statistics>
{
  new Statistics { Browser = "Chrome", Users = 37 },
  new Statistics { Browser = "UC Browser", Users = 17 },
  new Statistics { Browser = "iPhone", Users = 19 },
  new Statistics { Browser = "Others", Users = 4 },
  new Statistics { Browser = "Opera", Users = 11 },
  new Statistics { Browser = "Android", Users = 12 },
};
}
```

Mobile Browser Statistics



Funnel Size

The size of the funnel chart can be customized by using the [Width](#) and [Height](#) properties.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfAccumulationChart Title="Mobile Browser Statistics">
  <AccumulationChartLegendSettings
    Visible="false"></AccumulationChartLegendSettings>
  <AccumulationChartSeriesCollection>
    <AccumulationChartSeries DataSource="@StatisticsDetails" XName="Browser"
      YName="Users"
      Name="Browser" Type="AccumulationType.Funnel" Width="60%" Height="80%">
    </AccumulationChartSeries>
  </AccumulationChartSeriesCollection>
</SfAccumulationChart>

@code{
public class Statistics
{
    public string Browser { get; set; }
    public double Users { get; set; }
}
public List<Statistics> StatisticsDetails = new List<Statistics>
{
    new Statistics { Browser = "Chrome", Users = 37 },
```

```
new Statistics { Browser = "UC Browser", Users = 17 },  
new Statistics { Browser = "iPhone", Users = 19 },  
new Statistics { Browser = "Others", Users = 4 },  
new Statistics { Browser = "Opera", Users = 11 },  
new Statistics { Browser = "Android", Users = 12 },  
};  
}
```

Mobile Browser Statistics



The [Blazor Funnel Chart](#) example can be explored to learn to render and configure the funnel chart.

Funnel Neck Size

The neck size of the funnel chart can be customized by using the [NeckWidth](#) and [NeckHeight](#) properties.

ASPX-CS

```
@using Syncfusion.Blazor.Charts  
<SfAccumulationChart Title="Mobile Browser Statistics">  
  <AccumulationChartSeriesCollection>  
    <AccumulationChartSeries DataSource="@StatisticsDetails" XName="Browser"  
      YName="Users"  
      Name="Browser" Type="AccumulationType.Funnel" NeckWidth="15%"  
      NeckHeight="18%">
```



```
</AccumulationChartSeries>
</AccumulationChartSeriesCollection>
<AccumulationChartLegendSettings
Visible="false"></AccumulationChartLegendSettings>
</SfAccumulationChart>
@code{
public class Statistics
{
public string Browser { get; set; }
public double Users { get; set; }
}
public List<Statistics> StatisticsDetails = new List<Statistics>
{
new Statistics { Browser = "Chrome", Users = 37 },
new Statistics { Browser = "UC Browser", Users = 17 },
new Statistics { Browser = "iPhone", Users = 19 },
new Statistics { Browser = "Others", Users = 4 },
new Statistics { Browser = "Opera", Users = 11 },
new Statistics { Browser = "Android", Users = 12 },
};
}
```

Mobile Browser Statistics



Gap Between Funnel Segments

[Funnel chart](#) provides options to customize the space between the segments by using the [GapRatio](#) property of the series. It accepts the values ranging from 0 to 1.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfAccumulationChart Title="Mobile Browser Statistics">
  <AccumulationChartSeriesCollection>
    <AccumulationChartSeries DataSource="@StatisticsDetails" XName="Browser"
      YName="Users"
      Name="Browser" Type="AccumulationType.Funnel" NeckWidth="15%"
      NeckHeight="18%">
    </AccumulationChartSeries>
  </AccumulationChartSeriesCollection>
  <AccumulationChartLegendSettings
    Visible="false"></AccumulationChartLegendSettings>
</SfAccumulationChart>
@code{
public class Statistics
{
public string Browser { get; set; }
public double Users { get; set; }
}
public List<Statistics> StatisticsDetails = new List<Statistics>
{
new Statistics { Browser = "Chrome", Users = 37 },
new Statistics { Browser = "UC Browser", Users = 17 },
new Statistics { Browser = "iPhone", Users = 19 },
new Statistics { Browser = "Others", Users = 4 },
new Statistics { Browser = "Opera", Users = 11 },
new Statistics { Browser = "Android", Users = 12 },
};
}
```

Mobile Browser Statistics



Funnel Explode

Points can be exploded on mouse click by setting the [Explode](#) property to **true**. The point on load can be exploded using [ExplodeIndex](#). Explode distance can be set by using [ExplodeOffset](#) property.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfAccumulationChart Title="Mobile Browser Statistics">
  <AccumulationChartSeriesCollection>
    <AccumulationChartSeries DataSource="@StatisticsDetails" XName="Browser"
      YName="Users"
      Name="Browser" Type=" AccumulationType.Funnel" ExplodeIndex="3"
      Explode="true" ExplodeOffset="10%">
    </AccumulationChartSeries>
  </AccumulationChartSeriesCollection>
  <AccumulationChartLegendSettings
    Visible="false"></AccumulationChartLegendSettings>
</SfAccumulationChart>

@code{
  public class Statistics
  {
    public string Browser { get; set; }
    public double Users { get; set; }
  }
}
```

```
public List<Statistics> StatisticsDetails = new List<Statistics>
{
    new Statistics { Browser = "Chrome", Users = 37 },
    new Statistics { Browser = "UC Browser", Users = 17 },
    new Statistics { Browser = "iPhone", Users = 19 },
    new Statistics { Browser = "Others", Users = 4 },
    new Statistics { Browser = "Opera", Users = 11 },
    new Statistics { Browser = "Android", Users = 12 },
};
```

Mobile Browser Statistics



Smart Data Label

Labels will be arranged smartly automatically on the left side of the funnel and pyramid chart, when they overlap with each other.

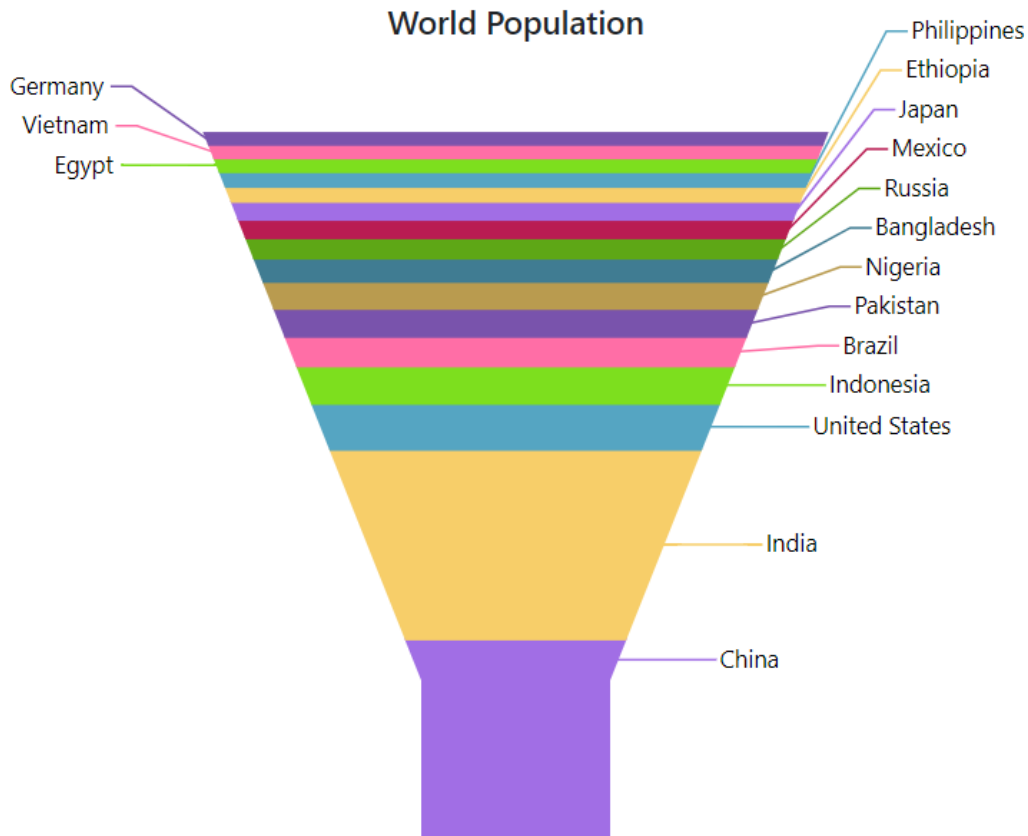
ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfAccumulationChart Title="World Population" EnableAnimation="false">
    <AccumulationChartLegendSettings
        Visible="false"></AccumulationChartLegendSettings>
    <AccumulationChartSeriesCollection>
        <AccumulationChartSeries DataSource="@StatisticsDetails" XName="Country"
            YName="Users"></AccumulationChartSeries>
    </AccumulationChartSeriesCollection>
</SfAccumulationChart>
```

```

Type="AccumulationType.Funnel" Explode="false" Width="50%" Height="80%"
NeckWidth="15%" NeckHeight="18%">
<AccumulationDataLabelSettings Visible="true" Name="Country"
Position="AccumulationLabelPosition.Outside">
<AccumulationChartConnector Length="6%"></AccumulationChartConnector>
</AccumulationDataLabelSettings>
</AccumulationChartSeries>
</AccumulationChartSeriesCollection>
</SfAccumulationChart>
@code{
public class Statistics
{
public string Country { get; set; }
public double Users { get; set; }
}
public List<Statistics> StatisticsDetails = new List<Statistics>
{
new Statistics { Country = "China", Users = 1409517397 },
new Statistics { Country = "India", Users = 1339180127 },
new Statistics { Country = "United States", Users = 324459463 },
new Statistics { Country = "Indonesia", Users = 263991379 },
new Statistics { Country = "Brazil", Users = 209288278 },
new Statistics { Country = "Pakistan", Users = 197015955 },
new Statistics { Country = "Nigeria", Users = 190886311 },
new Statistics { Country = "Bangladesh", Users = 164669751 },
new Statistics { Country = "Russia", Users = 143989754 },
new Statistics { Country = "Mexico", Users = 129163276 },
new Statistics { Country = "Japan", Users = 127484450 },
new Statistics { Country = "Ethiopia", Users = 104957438 },
new Statistics { Country = "Philippines", Users = 104918090 },
new Statistics { Country = "Egypt", Users = 97553151 },
new Statistics { Country = "Vietnam", Users = 95540800 },
new Statistics { Country = "Germany", Users = 82114224 },
};
}

```



Refer to the [Blazor Charts](#) feature tour page for its groundbreaking feature representations and also explore the [Blazor Accumulation Chart Example](#) to know various features of accumulation charts and how it is used to represent numeric proportional data.

See Also

- [Data label](#)
- [Grouping](#)

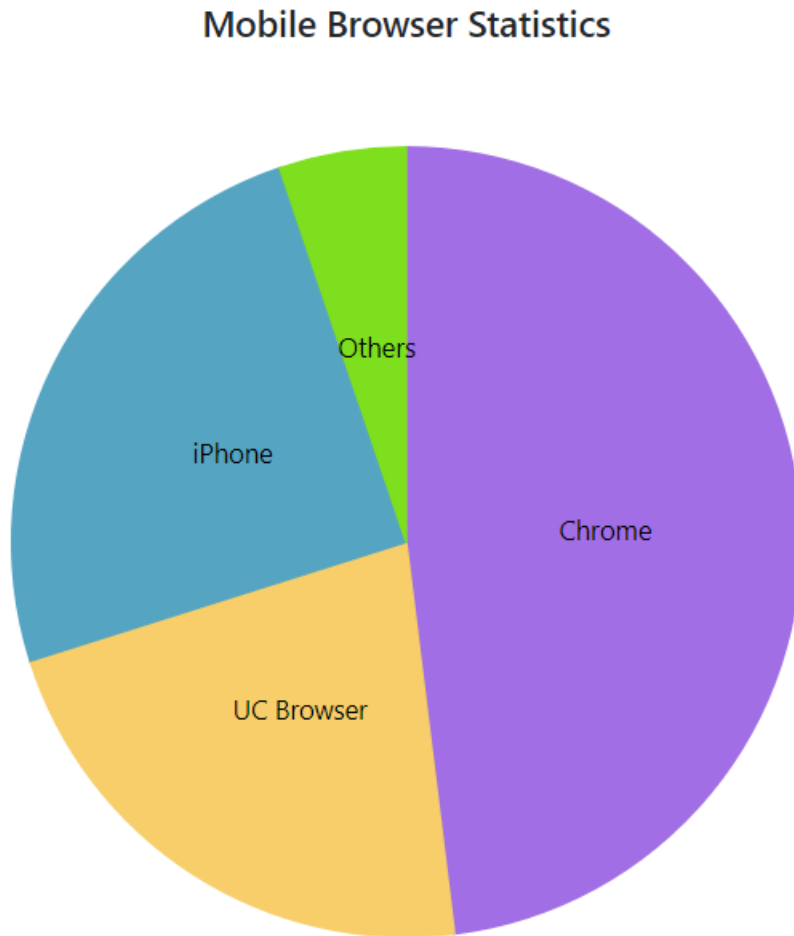
Data Label in Blazor Accumulation Chart Component

The [Visible](#) property in the [AccumulationDataLabelSettings](#) can be used to add a data label to a series point.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfAccumulationChart Title="Mobile Browser Statistics"
EnableSmartLabels="true">
<AccumulationChartLegendSettings
Visible="false"></AccumulationChartLegendSettings>
<AccumulationChartSeriesCollection>
<AccumulationChartSeries DataSource="@StatisticsDetails" XName="Browser"
YName="Users" Name="Browser">
```

```
<AccumulationDataLabelSettings Visible="true"
Name="Browser"></AccumulationDataLabelSettings>
</AccumulationChartSeries>
</AccumulationChartSeriesCollection>
</SfAccumulationChart>
@code{
public class Statistics
{
public string Browser { get; set; }
public double Users { get; set; }
public string Text { get; set; }
public string Fill { get; set; }
}
public List<Statistics> StatisticsDetails = new List<Statistics>
{
new Statistics { Browser = "Chrome", Users = 37, Text= "37%",
Fill="#498fff"},
new Statistics { Browser = "UC Browser", Users = 17, Text= "17%",
Fill="#ffa060"},
new Statistics { Browser = "iPhone", Users = 19, Text= "19%",
Fill="#ff68b6"},
new Statistics { Browser = "Others", Users = 4 , Text= "4%",
Fill="#81e2a1"},
};
}
```



Position

The [Position](#) property in the [AccumulationDataLabelSettings](#) allows the data label to be placed either [Inside](#) or [Outside](#) of the chart.

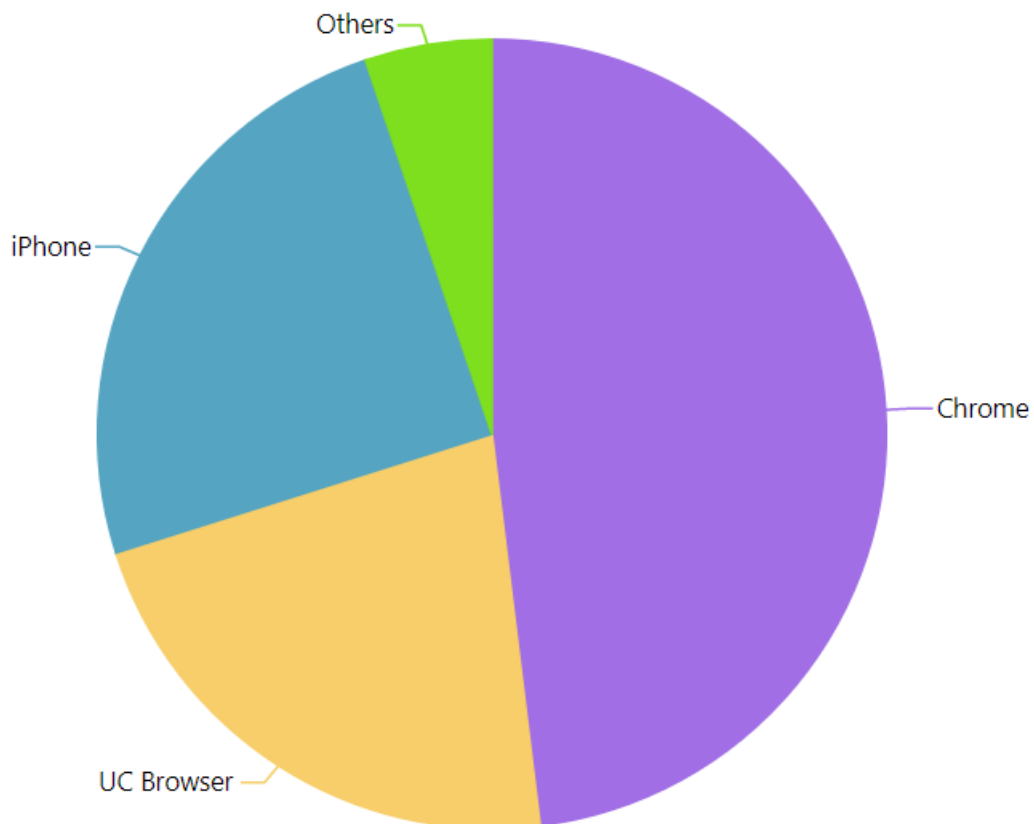
ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfAccumulationChart Title="Mobile Browser Statistics">
  <AccumulationChartLegendSettings
    Visible="false"></AccumulationChartLegendSettings>
  <AccumulationChartSeriesCollection>
    <AccumulationChartSeries DataSource="@StatisticsDetails" XName="Browser"
      YName="Users" Name="Browser">
      <AccumulationDataLabelSettings Visible="true" Name="Browser"
        Position="AccumulationLabelPosition.Outside"></AccumulationDataLabelSettings
      >
    </AccumulationChartSeries>
  </AccumulationChartSeriesCollection>
</SfAccumulationChart>
@code{
public class Statistics
{
```



```
public string Browser { get; set; }
public double Users { get; set; }
public string Text { get; set; }
public string Fill { get; set; }
}
public List<Statistics> StatisticsDetails = new List<Statistics>
{
    new Statistics { Browser = "Chrome", Users = 37, Text= "37%",
    Fill="#498fff"},
    new Statistics { Browser = "UC Browser", Users = 17, Text= "17%",
    Fill="#ffa060"},
    new Statistics { Browser = "iPhone", Users = 19, Text= "19%",
    Fill="#ff68b6"},
    new Statistics { Browser = "Others", Users = 4 , Text= "4%",
    Fill="#81e2a1"},
};
}
```

Mobile Browser Statistics



Smart Labels

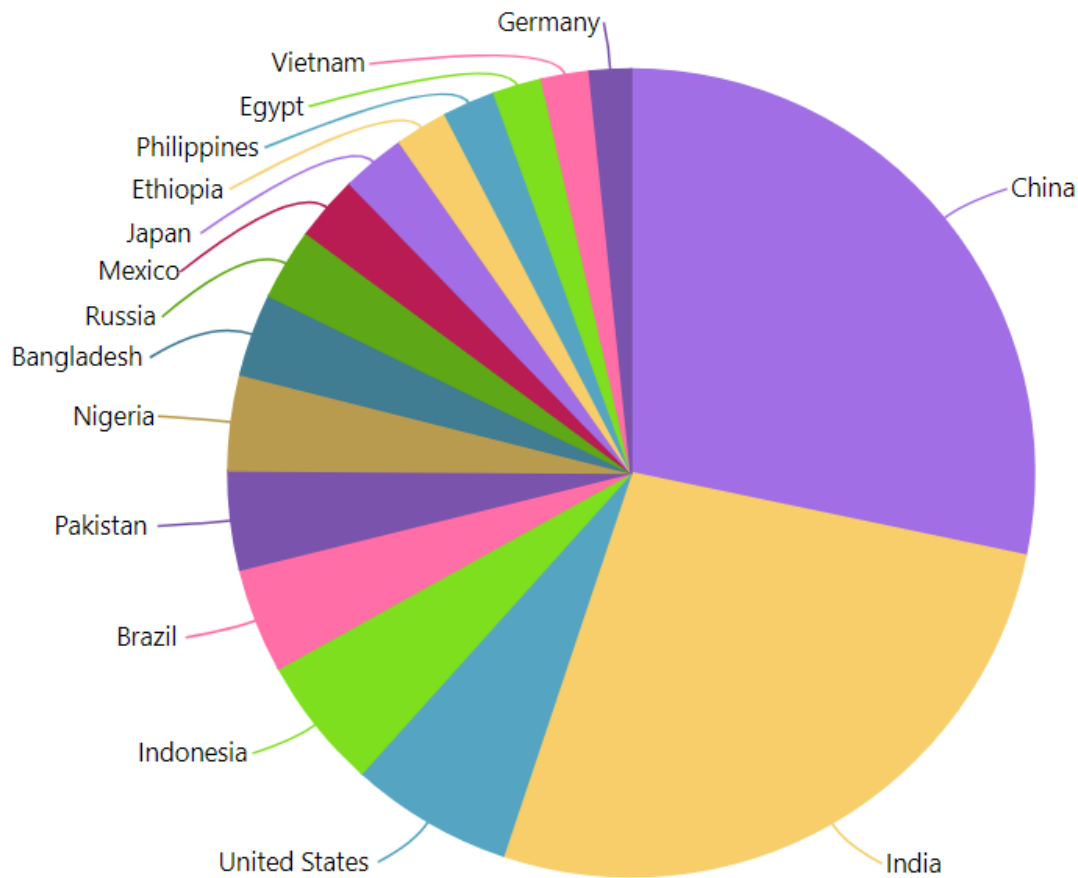
Data labels will be arranged smartly without overlapping with each other. The [EnableSmartLabels](#) property can be used to enable or disable this feature.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfAccumulationChart EnableSmartLabels="true">
  <AccumulationChartLegendSettings
    Visible="false"></AccumulationChartLegendSettings>
  <AccumulationChartSeriesCollection>
    <AccumulationChartSeries DataSource="@StatisticsDetails" XName="Country"
      YName="Users">
      <AccumulationDataLabelSettings Visible="true" Name="Country"
        Position="AccumulationLabelPosition.Outside">
        <AccumulationChartConnector Type="ConnectorType.Curve" Length="20px" />
      </AccumulationDataLabelSettings>
    </AccumulationChartSeries>
  </AccumulationChartSeriesCollection>
</SfAccumulationChart>

@code{
public class Statistics
{
    public string Country { get; set; }
    public double Users { get; set; }
}

public List<Statistics> StatisticsDetails = new List<Statistics>
{
    new Statistics { Country = "China", Users = 1409517397 },
    new Statistics { Country = "India", Users = 1339180127 },
    new Statistics { Country = "United States", Users = 324459463 },
    new Statistics { Country = "Indonesia", Users = 263991379 },
    new Statistics { Country = "Brazil", Users = 209288278 },
    new Statistics { Country = "Pakistan", Users = 197015955 },
    new Statistics { Country = "Nigeria", Users = 190886311 },
    new Statistics { Country = "Bangladesh", Users = 164669751 },
    new Statistics { Country = "Russia", Users = 143989754 },
    new Statistics { Country = "Mexico", Users = 129163276 },
    new Statistics { Country = "Japan", Users = 127484450 },
    new Statistics { Country = "Ethiopia", Users = 104957438 },
    new Statistics { Country = "Philippines", Users = 104918090 },
    new Statistics { Country = "Egypt", Users = 97553151 },
    new Statistics { Country = "Vietnam", Users = 95540800 },
    new Statistics { Country = "Germany", Users = 82114224 },
};
}
```



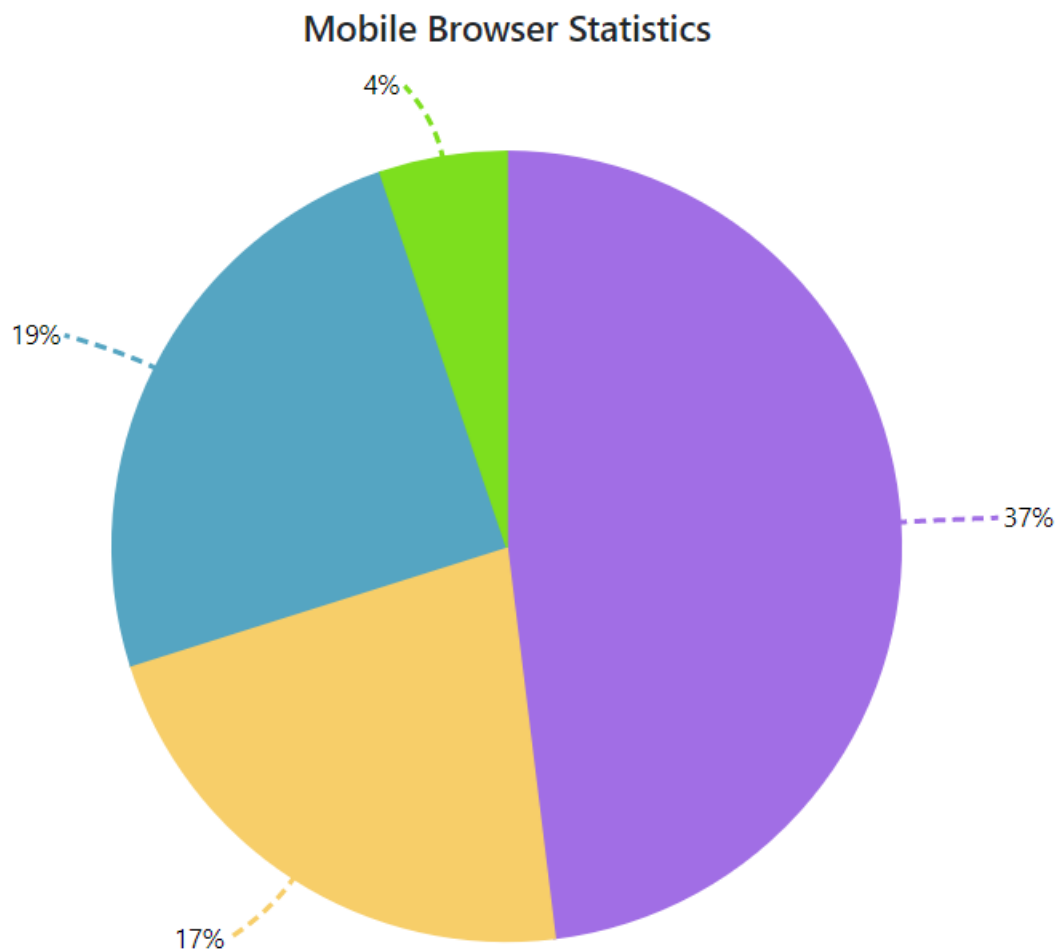
Connector Line

When the data label is placed [Outside](#) the chart, the connector line will be visible. The [Type](#), [Color](#), [Width](#), [Length](#) and [DashArray](#) properties can be used to customize the connector line.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfAccumulationChart Title="Mobile Browser Statistics"
EnableSmartLabels="true">
  <AccumulationChartLegendSettings
Visible="false"></AccumulationChartLegendSettings>
  <AccumulationChartSeriesCollection>
    <AccumulationChartSeries DataSource="@StatisticsDetails" XName="Browser"
YName="Users" Name="Browser">
      <AccumulationDataLabelSettings Visible="true" Name="Text"
Position="AccumulationLabelPosition.Outside">
        <AccumulationChartConnector Color="#f4429e" Length="50px" Width="2"
Type="ConnectorType.Curve" DashArray="5,3"></AccumulationChartConnector>
      </AccumulationDataLabelSettings>
    </AccumulationChartSeries>
  </AccumulationChartSeriesCollection>
</SfAccumulationChart>
@code{
```

```
public class Statistics
{
    public string Browser { get; set; }
    public double Users { get; set; }
    public string Text { get; set; }
    public string Fill { get; set; }
}
public List<Statistics> StatisticsDetails = new List<Statistics>
{
    new Statistics { Browser = "Chrome", Users = 37, Text= "37%",
    Fill="#498fff"},
    new Statistics { Browser = "UC Browser", Users = 17, Text= "17%",
    Fill="#ffa060"},
    new Statistics { Browser = "iPhone", Users = 19, Text= "19%",
    Fill="#ff68b6"},
    new Statistics { Browser = "Others", Users = 4 , Text= "4%",
    Fill="#81e2a1"},
};
```



Text Mapping

The [Name](#) property can be used to map text from a data source to a data label.

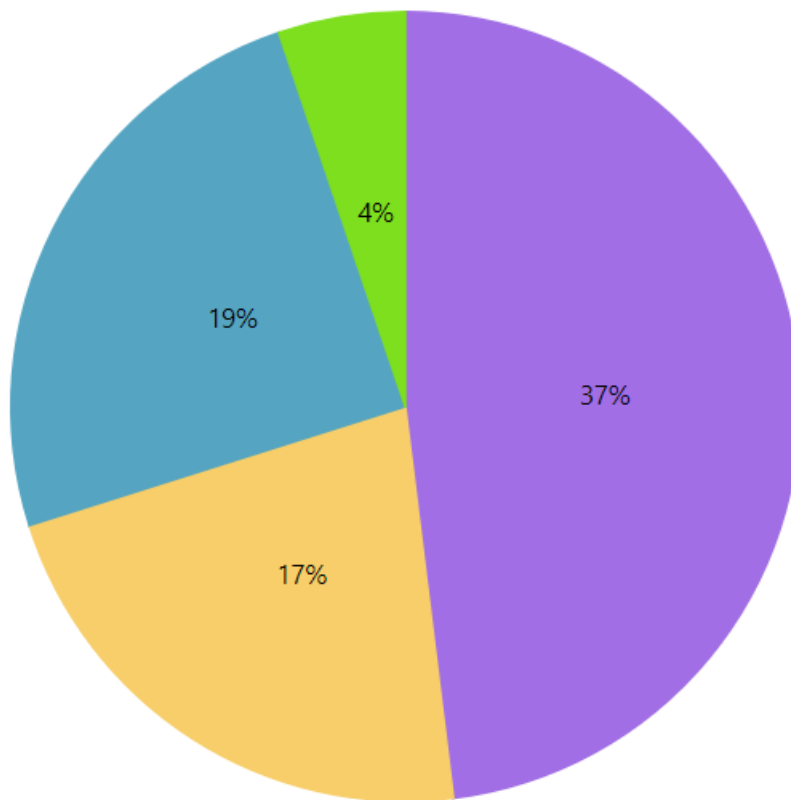
ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfAccumulationChart Title="Mobile Browser Statistics"
EnableSmartLabels="true">
  <AccumulationChartLegendSettings
Visible="false"></AccumulationChartLegendSettings>
  <AccumulationChartSeriesCollection>
    <AccumulationChartSeries DataSource="@StatisticsDetails" XName="Browser"
YName="Users" Name="Browser">
      <AccumulationDataLabelSettings Visible="true" Name="Text">
      </AccumulationDataLabelSettings>
    </AccumulationChartSeries>
  </AccumulationChartSeriesCollection>
</SfAccumulationChart>

@code{
public class Statistics
{
    public string Browser { get; set; }
    public double Users { get; set; }
    public string Text { get; set; }
    public string Fill { get; set; }
}

public List<Statistics> StatisticsDetails = new List<Statistics>
{
    new Statistics { Browser = "Chrome", Users = 37, Text= "37%",
Fill="#498fff"},
    new Statistics { Browser = "UC Browser", Users = 17, Text= "17%",
Fill="#ffa060"},
    new Statistics { Browser = "iPhone", Users = 19, Text= "19%",
Fill="#ff68b6"},
    new Statistics { Browser = "Others", Users = 4 , Text= "4%",
Fill="#81e2a1"},
};
}
```

Mobile Browser Statistics



Refer to the [Blazor Charts](#) feature tour page for its groundbreaking feature representations and also explore the [Blazor Accumulation Chart Example](#) to know various features of accumulation charts and how it is used to represent numeric proportional data.

See Also

- [Tooltip](#)
- [Legend](#)

<!-- markdownlint-disable MD036 -->

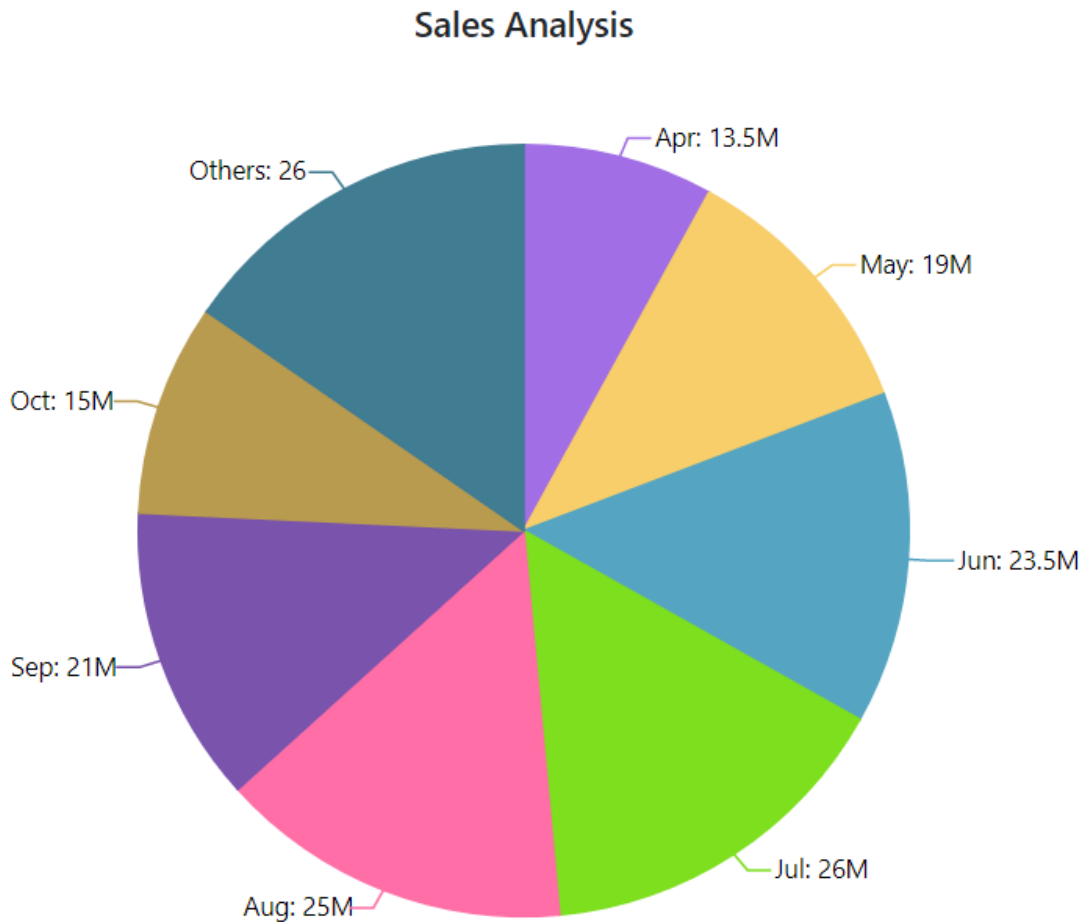
Grouping in Blazor Accumulation Chart Component

The value set to the [GroupTo](#) property can be used to club/group a few points in the series. Points with a value less than [GroupTo](#) are grouped together and displayed as a single point with the label **Others**. In addition, the property value can be set in percentage (percentage of total data points value).

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfAccumulationChart Title="Sales Analysis" EnableSmartLabels="true">
```

```
<AccumulationChartLegendSettings
Visible="false"></AccumulationChartLegendSettings>
<AccumulationChartSeriesCollection>
<AccumulationChartSeries DataSource="@DataSource" XName="XValue"
YName="YValue" Name="Sales" GroupTo="10">
<AccumulationDataLabelSettings Visible="true" Name="Text"
Position="AccumulationLabelPosition.Outside">
</AccumulationDataLabelSettings>
</AccumulationChartSeries>
</AccumulationChartSeriesCollection>
</SfAccumulationChart>
@code{
public class ChartData
{
public string XValue { get; set; }
public double YValue { get; set; }
public string Text { get; set; }
}
public List<ChartData> DataSource = new List<ChartData>
{
new ChartData { XValue = "Jan", YValue = 3, Text= "Jan: 3M" },
new ChartData { XValue = "Feb", YValue = 3.5, Text= "Feb: 3.5M" },
new ChartData { XValue = "Mar", YValue = 7, Text= "'Mar: 7M" },
new ChartData { XValue = "Apr", YValue = 13.5, Text= "Apr: 13.5M" },
new ChartData { XValue = "May", YValue = 19, Text= "May: 19M" },
new ChartData { XValue = "Jun", YValue = 23.5, Text= "Jun: 23.5M" },
new ChartData { XValue = "Jul", YValue = 26, Text= "Jul: 26M" },
new ChartData { XValue = "Aug", YValue = 25, Text= "Aug: 25M" },
new ChartData { XValue = "Sep", YValue = 21, Text= "Sep: 21M" },
new ChartData { XValue = "Oct", YValue = 15, Text= "Oct: 15M" },
new ChartData { XValue = "Nov", YValue = 9, Text= "Nov: 9M" },
new ChartData { XValue = "Dec", YValue = 3.5, Text= "Dec: 3.5M" }
};
}
```



Pie Grouping

Broken Slice

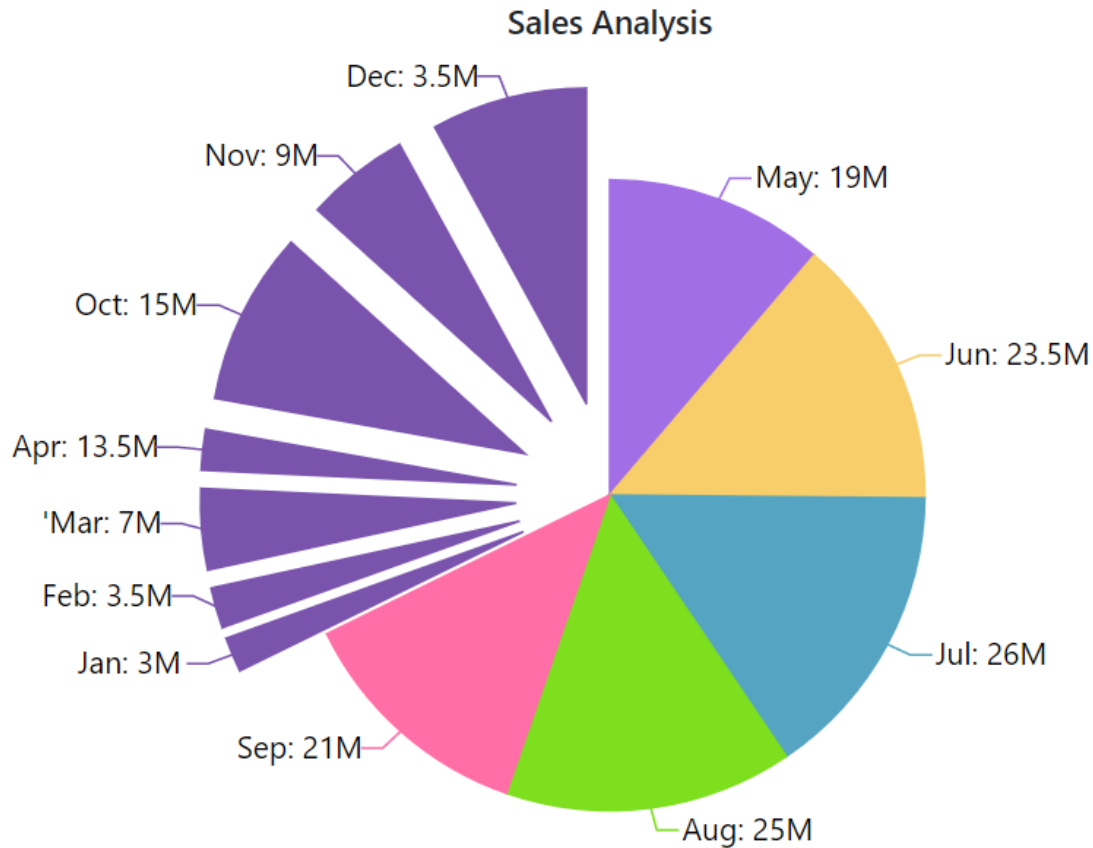
The points that have been grouped together will appear as a single slice with the label **Others**, which will explode and break into separate slices when clicked.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfAccumulationChart Title="Sales Analysis" EnableSmartLabels="true">
  <AccumulationChartTooltipSettings
    Enable="true"></AccumulationChartTooltipSettings>
  <AccumulationChartLegendSettings
    Visible="false"></AccumulationChartLegendSettings>
  <AccumulationChartSeriesCollection>
    <AccumulationChartSeries DataSource="@DataSource" XName="XValue"
      YName="YValue" Explode="true" Radius="70%" GroupTo="15">
      <AccumulationDataLabelSettings Visible="true" Name="Text"
        Position="AccumulationLabelPosition.Outside">
        <AccumulationChartConnector Type="ConnectorType.Line"
          Length="5%"></AccumulationChartConnector>
        <AccumulationChartDataLabelFont
          Size="14px"></AccumulationChartDataLabelFont>
      </AccumulationDataLabelSettings>
    </AccumulationChartSeries>
  </AccumulationChartSeriesCollection>
</SfAccumulationChart>
```



```
</AccumulationChartSeries>
</AccumulationChartSeriesCollection>
</SfAccumulationChart>
@code{
public class ChartData
{
public string XValue { get; set; }
public double YValue { get; set; }
public string Text { get; set; }
}
public List<ChartData> DataSource = new List<ChartData>
{
new ChartData { XValue = "Jan", YValue = 3, Text= "Jan: 3M" },
new ChartData { XValue = "Feb", YValue = 3.5, Text= "Feb: 3.5M" },
new ChartData { XValue = "Mar", YValue = 7, Text= "'Mar: 7M" },
new ChartData { XValue = "Apr", YValue = 3.5, Text= "Apr: 13.5M" },
new ChartData { XValue = "May", YValue = 19, Text= "May: 19M" },
new ChartData { XValue = "Jun", YValue = 23.5, Text= "Jun: 23.5M" },
new ChartData { XValue = "Jul", YValue = 26, Text= "Jul: 26M" },
new ChartData { XValue = "Aug", YValue = 25, Text= "Aug: 25M" },
new ChartData { XValue = "Sep", YValue = 21, Text= "Sep: 21M" },
new ChartData { XValue = "Oct", YValue = 15, Text= "Oct: 15M" },
new ChartData { XValue = "Nov", YValue = 9, Text= "Nov: 9M" },
new ChartData { XValue = "Dec", YValue = 13.5, Text= "Dec: 3.5M" }
};
}
```



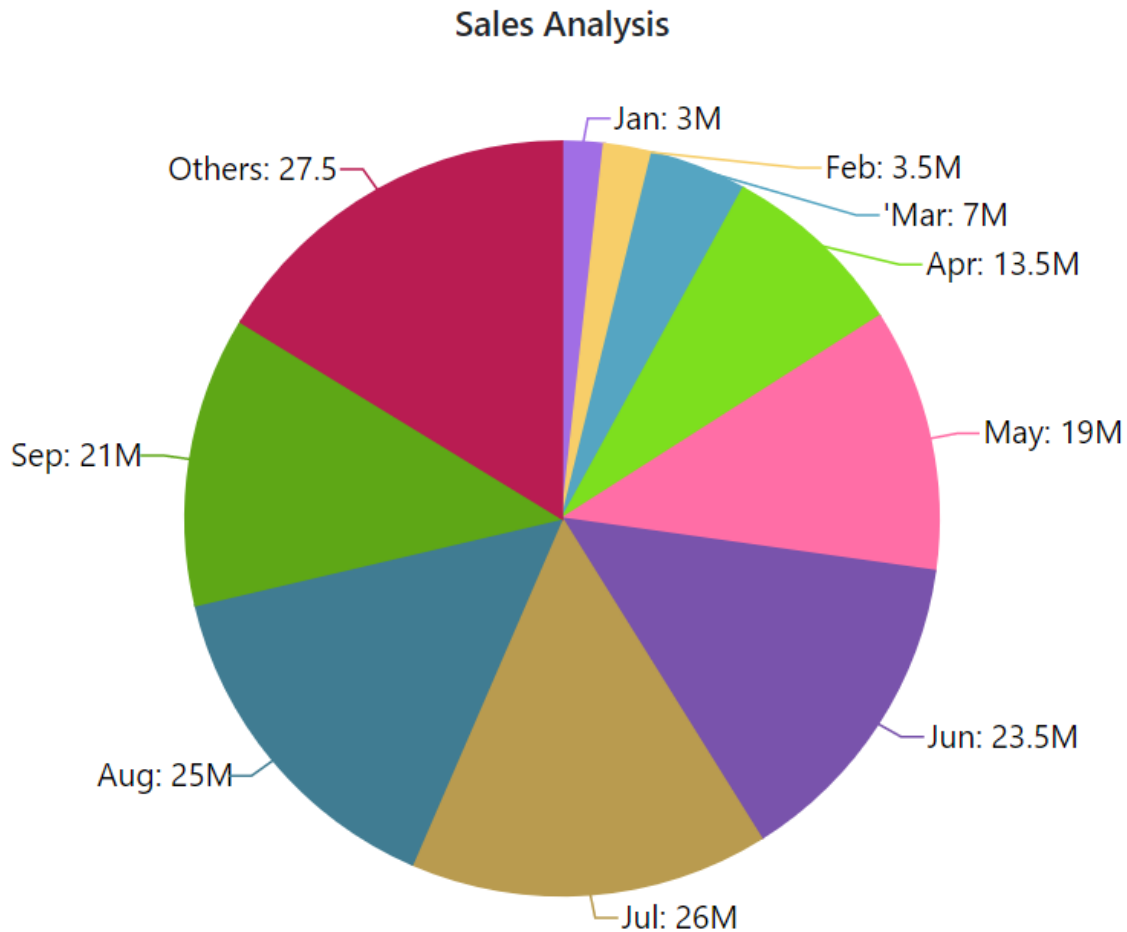
Group Mode

When the [GroupMode](#) property is set to [Point](#), the points are displayed as separate slices according to the [GroupTo](#) value. The remaining points will be grouped into a single slice and displayed.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfAccumulationChart Title="Sales Analysis" EnableSmartLabels="true">
  <AccumulationChartTooltipSettings
    Enable="true"></AccumulationChartTooltipSettings>
  <AccumulationChartLegendSettings
    Visible="false"></AccumulationChartLegendSettings>
  <AccumulationChartSeriesCollection>
    <AccumulationChartSeries DataSource="@DataSource" XName="XValue"
      YName="YValue" GroupTo="9" GroupMode=GroupMode.Point>
      <AccumulationDataLabelSettings Visible="true" Name="Text"
        Position="AccumulationLabelPosition.Outside">
        <AccumulationChartConnector Type="ConnectorType.Line"
          Length="5%"></AccumulationChartConnector>
        <AccumulationChartDataLabelFont
          Size="14px"></AccumulationChartDataLabelFont>
      </AccumulationDataLabelSettings>
    </AccumulationChartSeries>
  </AccumulationChartSeriesCollection>
```

```
</SfAccumulationChart>
@code{
public class ChartData
{
public string XValue { get; set; }
public double YValue { get; set; }
public string Text { get; set; }
}
public List<ChartData> DataSource = new List<ChartData>
{
new ChartData { XValue = "Jan", YValue = 3, Text= "Jan: 3M" },
new ChartData { XValue = "Feb", YValue = 3.5, Text= "Feb: 3.5M" },
new ChartData { XValue = "Mar", YValue = 7, Text= "'Mar: 7M" },
new ChartData { XValue = "Apr", YValue = 13.5, Text= "Apr: 13.5M" },
new ChartData { XValue = "May", YValue = 19, Text= "May: 19M" },
new ChartData { XValue = "Jun", YValue = 23.5, Text= "Jun: 23.5M" },
new ChartData { XValue = "Jul", YValue = 26, Text= "Jul: 26M" },
new ChartData { XValue = "Aug", YValue = 25, Text= "Aug: 25M" },
new ChartData { XValue = "Sep", YValue = 21, Text= "Sep: 21M" },
new ChartData { XValue = "Oct", YValue = 15, Text= "Oct: 15M" },
new ChartData { XValue = "Nov", YValue = 9, Text= "Nov: 9M" },
new ChartData { XValue = "Dec", YValue = 3.5, Text= "Dec: 3.5M" }
};
}
```



Refer to the [Blazor Charts](#) feature tour page for its groundbreaking feature representations and also explore the [Blazor Accumulation Chart Example](#) to know various features of accumulation charts and how it is used to represent numeric proportional data.

- [Data Label](#)
- [Tooltip](#)
- [Legend](#)

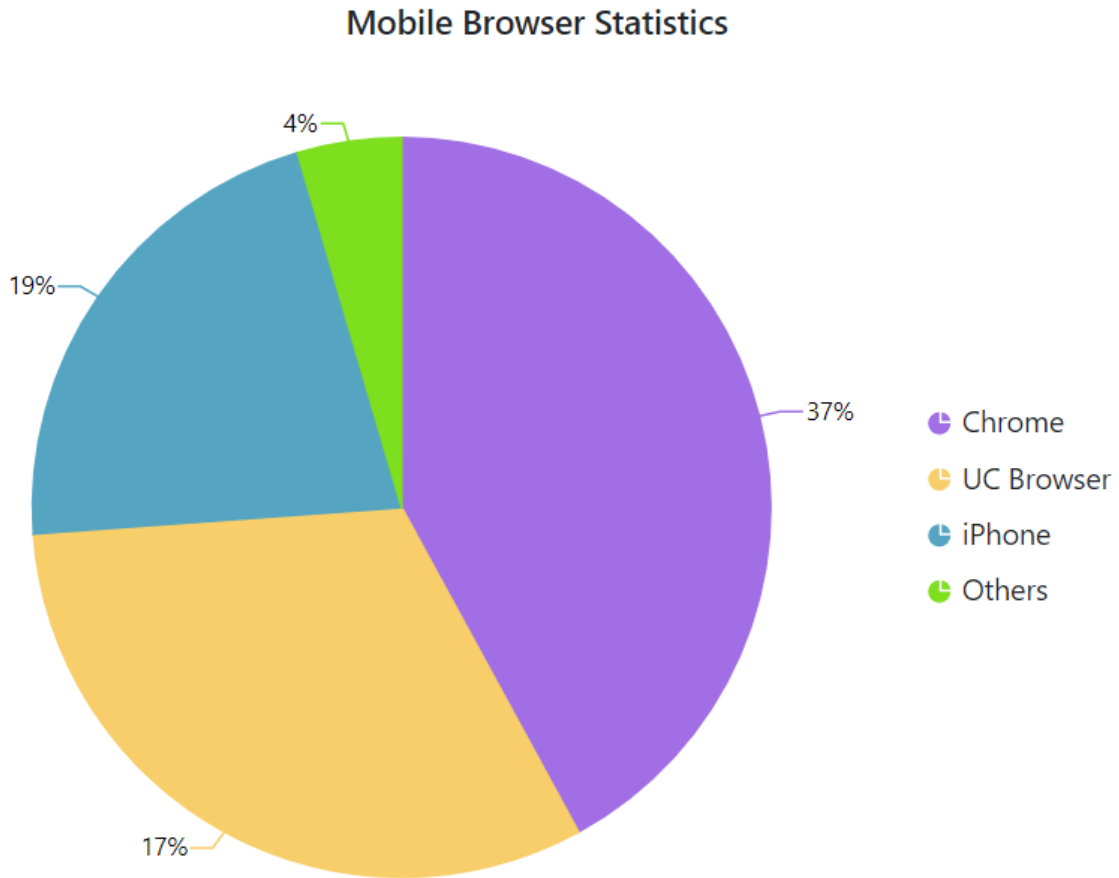
Empty Points in Blazor Accumulation Chart Component

Data points that contain **NaN** or **null** value are considered as empty points. The empty data points can be ignored or not plotted in the chart. Those points can be customized using the [AccumulationChartEmptyPointSettings](#) in series.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfAccumulationChart Title="Mobile Browser Statistics">
  <AccumulationChartSeriesCollection>
    <AccumulationChartSeries DataSource="@StatisticsDetails" XName="Browser"
      YName="Users"
      Name="Profit">
```

```
<AccumulationChartEmptyPointSettings
Mode="@Mode"></AccumulationChartEmptyPointSettings>
<AccumulationDataLabelSettings Visible="true" Name="Text"
Position="AccumulationLabelPosition.Outside"></AccumulationDataLabelSettings
>
</AccumulationChartSeries>
</AccumulationChartSeriesCollection>
</SfAccumulationChart>
@code{
public EmptyPointMode Mode = EmptyPointMode.Gap;
public class Statistics
{
public string Browser { get; set; }
public double? Users { get; set; }
public string Text { get; set; }
public string Fill { get; set; }
}
public List<Statistics> StatisticsDetails = new List<Statistics>
{
new Statistics { Browser = "Chrome", Users = 37, Text= "37%",
Fill="#498fff"},
new Statistics { Browser = "UC Browser", Users = null, Text= "17%",
Fill="#ffa060"},
new Statistics { Browser = "iPhone", Users = 19, Text= "19%",
Fill="#ff68b6"},
new Statistics { Browser = "Others", Users = 4 , Text= "4%",
Fill="#81e2a1"},
};
}
```



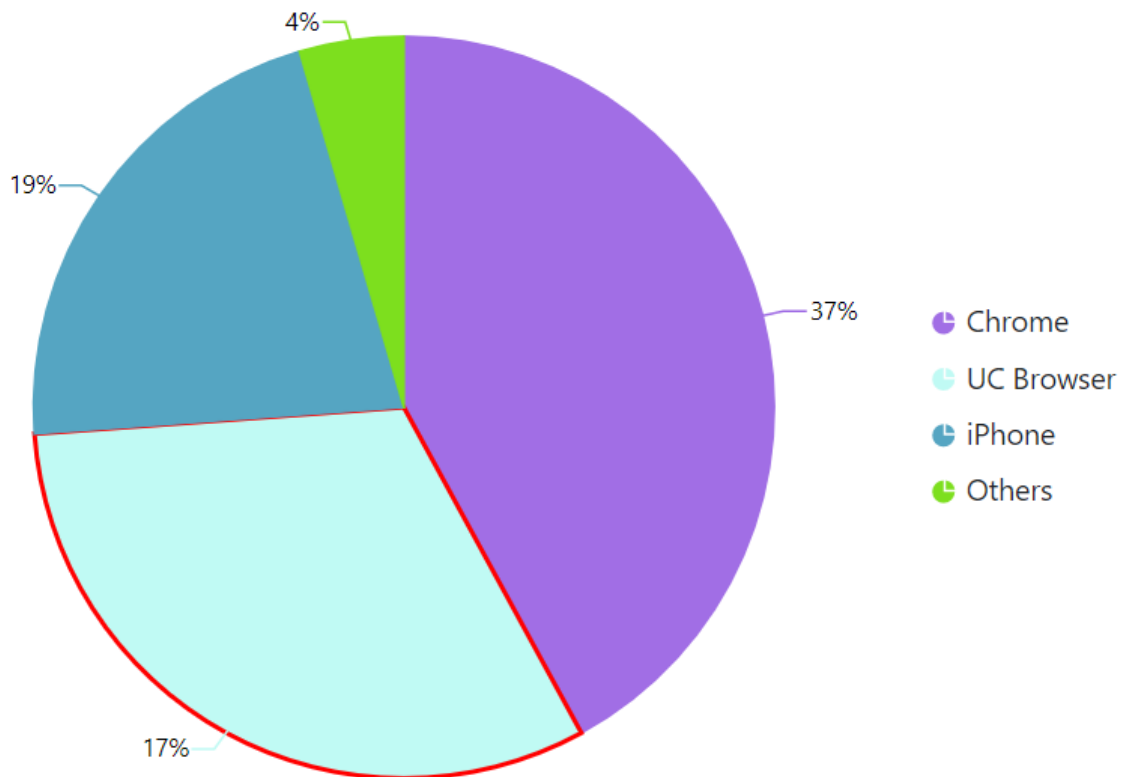
Customization

The [Mode](#) property can be used to handle the visibility of the empty points. The default mode of the empty point is **Gap**. Other supported modes are **Average**, **Drop** and **Zero**. The [Fill](#) property can be used to set a specific color for an empty point, and the [Border](#) property can be used to set the border for an empty point.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfAccumulationChart Title="Mobile Browser Statistics">
  <AccumulationChartSeriesCollection>
    <AccumulationChartSeries DataSource="@StatisticsDetails" XName="Browser"
      YName="Users"
      Name="Profit">
      <AccumulationChartEmptyPointSettings Mode="@Mode" Fill="#c0faf4">
        <AccumulationChartEmptyPointBorder Color="red"
          Width="2"></AccumulationChartEmptyPointBorder>
      </AccumulationChartEmptyPointSettings>
      <AccumulationDataLabelSettings Visible="true" Name="Text"
        Position="AccumulationLabelPosition.Outside"></AccumulationDataLabelSettings>
    </AccumulationChartSeries>
  </AccumulationChartSeriesCollection>
</SfAccumulationChart>
@code{
```

```
public EmptyPointMode Mode = EmptyPointMode.Average;
public class Statistics
{
    public string Browser { get; set; }
    public double? Users { get; set; }
    public string Text { get; set; }
    public string Fill { get; set; }
}
public List<Statistics> StatisticsDetails = new List<Statistics>
{
    new Statistics { Browser = "Chrome", Users = 37, Text= "37%",
    Fill="#498fff"},
    new Statistics { Browser = "UC Browser", Users = null, Text= "17%",
    Fill="#ffa060"},
    new Statistics { Browser = "iPhone", Users = 19, Text= "19%",
    Fill="#ff68b6"},
    new Statistics { Browser = "Others", Users = 4 , Text= "4%",
    Fill="#81e2a1"},
};
}
```

Mobile Browser Statistics

Refer to the [Blazor Charts](#) feature tour page for its groundbreaking feature representations and also explore the [Blazor Chart Example](#) to know about the various chart types and how to represent time-dependent data, showing trends at equal intervals.

See Also

- [Data Label](#)
- [Tooltip](#)
- [Legend](#)

Annotation in Blazor Accumulation Chart Component

Annotations are texts, shapes, or images that are used to highlight a specific region of interest in a chart. The [AccumulationChartAnnotation](#) property allows to add annotations to the chart. Specify the element that needs to be displayed in the accumulation chart area by using the [Content](#) property of the annotation.

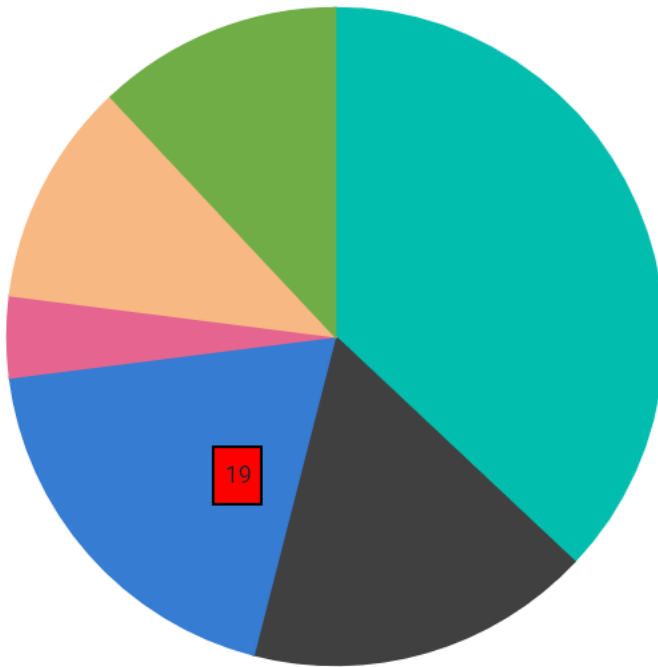
ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfAccumulationChart Title="Mobile Browser Statistics">
  <AccumulationChartSeriesCollection>
    <AccumulationChartSeries DataSource="@StatisticsDetails" XName="Browser"
      YName="Users" Name="Browser">
      <AccumulationDataLabelSettings Visible="true" Name="Text"
        Position="AccumulationLabelPosition.Outside"></AccumulationDataLabelSettings
      >
    </AccumulationChartSeries>
  </AccumulationChartSeriesCollection>
  <AccumulationChartAnnotations>
    <AccumulationChartAnnotation X="Opera" Y="11"
      CoordinateUnits="@Syncfusion.Blazor.Charts.Units.Point"
      Region="@Syncfusion.Blazor.Charts.Regions.Series">
      <ContentTemplate>
        <div style='border: 1px solid black;background-color:red;padding: 5px 5px
          5px 5px'>19</div>
      </ContentTemplate>
    </AccumulationChartAnnotation>
  </AccumulationChartAnnotations>
</SfAccumulationChart>

@code{
public class Statistics
{
    public string Browser { get; set; }
    public double Users { get; set; }
}

public List<Statistics> StatisticsDetails = new List<Statistics>
{
    new Statistics { Browser = "Chrome", Users = 37 },
    new Statistics { Browser = "UC Browser", Users = 17 },
    new Statistics { Browser = "iPhone", Users = 19 },
    new Statistics { Browser = "Others", Users = 4 },
    new Statistics { Browser = "Opera", Users = 11 },
    new Statistics { Browser = "Android", Users = 12 },
};
}
```


Mobile Browser Statistics



Region

The [Region](#) property can be used to insert annotations in relation to a series or a chart. By default, it is positioned with respect to a [Accumulation Chart](#).

ASPX-CS

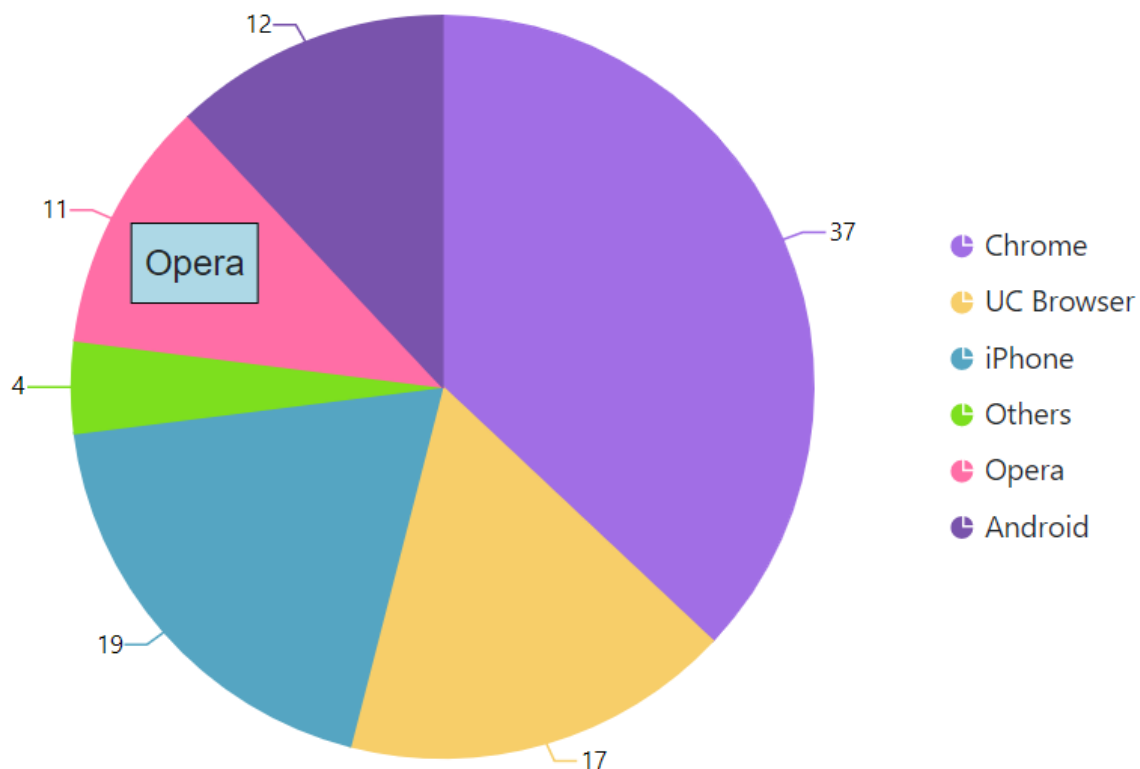
```
@using Syncfusion.Blazor.Charts
<SfAccumulationChart Title="Mobile Browser Statistics">
  <AccumulationChartSeriesCollection>
    <AccumulationChartSeries DataSource="@StatisticsDetails" XName="Browser"
      YName="Users" Name="Browser">
      <AccumulationDataLabelSettings Visible="true" Name="Text"
        Position="AccumulationLabelPosition.Outside"></AccumulationDataLabelSettings>
    </AccumulationChartSeries>
  </AccumulationChartSeriesCollection>
  <AccumulationChartAnnotations>
    <AccumulationChartAnnotation X=130 Y=180
      CoordinateUnits="@Syncfusion.Blazor.Charts.Units.Pixel"
      Region="@Syncfusion.Blazor.Charts.Regions.Chart">
      <ContentTemplate>
        <div style='border: 1px solid black;background-color:lightblue;padding: 5px
          5px 5px 5px'>Opera</div>
      </ContentTemplate>
    </AccumulationChartAnnotation>
  </AccumulationChartAnnotations>
</SfAccumulationChart>
```

```

</AccumulationChartAnnotations>
</SfAccumulationChart>
@code{
public class Statistics
{
public string Browser { get; set; }
public double Users { get; set; }
}
public List<Statistics> StatisticsDetails = new List<Statistics>
{
new Statistics { Browser = "Chrome", Users = 37 },
new Statistics { Browser = "UC Browser", Users = 17 },
new Statistics { Browser = "iPhone", Users = 19 },
new Statistics { Browser = "Others", Users = 4 },
new Statistics { Browser = "Opera", Users = 11 },
new Statistics { Browser = "Android", Users = 12 },
};
}

```

Mobile Browser Statistics



Co-ordinate Units

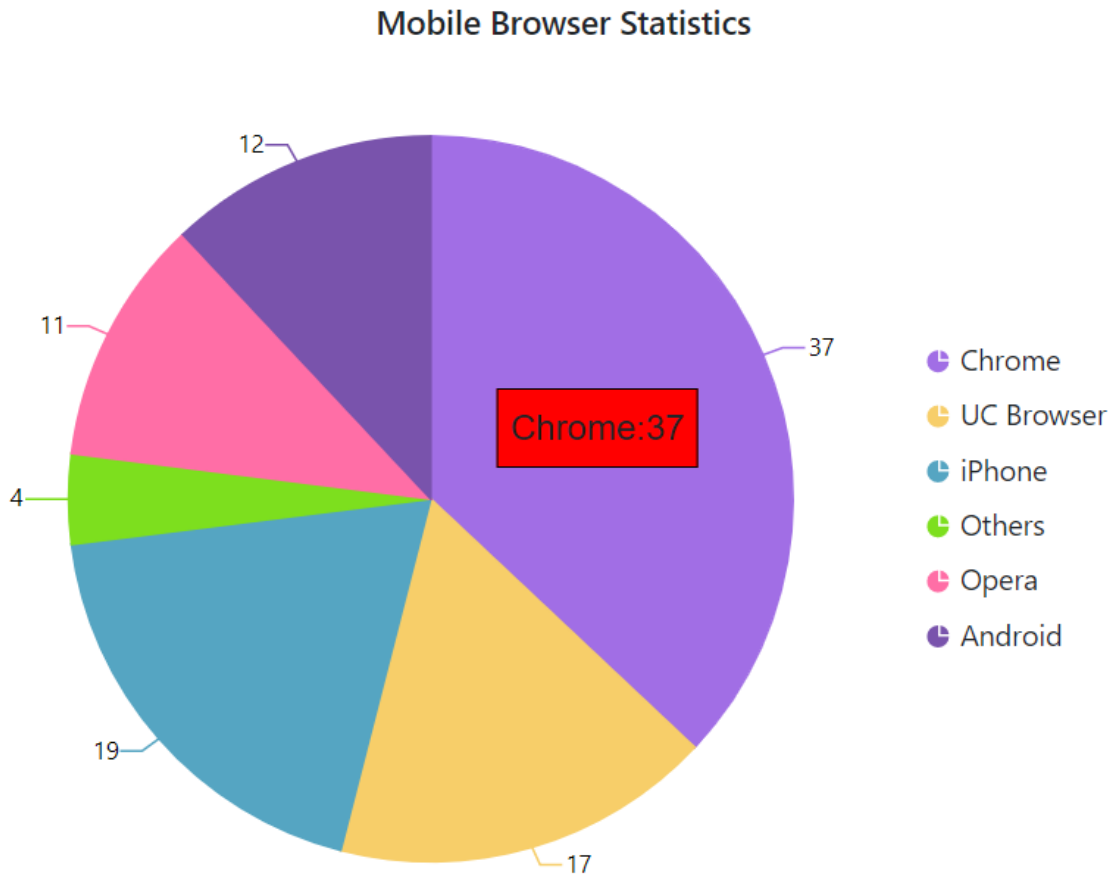
The [CoordinateUnits](#) property allows to specify the annotation's coordinate units either in [Pixel](#) or [Point](#).

ASPX-CS

```

@using Syncfusion.Blazor.Charts
<SfAccumulationChart Title="Mobile Browser Statistics">
<AccumulationChartSeriesCollection>
<AccumulationChartSeries DataSource="@StatisticsDetails" XName="Browser"
YName="Users" Name="Browser">
<AccumulationDataLabelSettings Visible="true" Name="Text"
Position="AccumulationLabelPosition.Outside"></AccumulationDataLabelSettings
>
</AccumulationChartSeries>
</AccumulationChartSeriesCollection>
<AccumulationChartAnnotations>
<AccumulationChartAnnotation X="Chrome" Y="37"
CoordinateUnits="@Syncfusion.Blazor.Charts.Units.Point"
Region="@Syncfusion.Blazor.Charts.Regions.Series">
<ContentTemplate>
<div style='border: 1px solid black;background-color:red;padding: 5px 5px
5px 5px'>Chrome:37</div>
</ContentTemplate>
</AccumulationChartAnnotation>
</AccumulationChartAnnotations>
</SfAccumulationChart>
@code{
public class Statistics
{
public string Browser { get; set; }
public double Users { get; set; }
}
public List<Statistics> StatisticsDetails = new List<Statistics>
{
new Statistics { Browser = "Chrome", Users = 37 },
new Statistics { Browser = "UC Browser", Users = 17 },
new Statistics { Browser = "iPhone", Users = 19 },
new Statistics { Browser = "Others", Users = 4 },
new Statistics { Browser = "Opera", Users = 11 },
new Statistics { Browser = "Android", Users = 12 },
};
}

```



Refer to the [Blazor Charts](#) feature tour page for its groundbreaking feature representations and also explore the [Blazor Accumulation Chart Example](#) to know about the various features of accumulation charts and how it is used to represent numeric proportional data.

See Also

- [Data Label](#)
- [Tooltip](#)
- [Legend](#)

Animation in Blazor Accumulation Chart Component

You can customize animation for a series using [Animation](#) property. You can enable or disable animation of the series using [Enable](#) property. [Duration](#) specifies the duration of an animation and [Delay](#) allows us to start the animation at desire time.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfAccumulationChart Title="Mobile Browser Statistics">
  <AccumulationChartLegendSettings
    Visible="false"></AccumulationChartLegendSettings>
  <AccumulationChartSeriesCollection>
    <AccumulationChartSeries DataSource="@StatisticsDetails" XName="Browser"
      YName="Users" Name="Browser">
```

```

<AccumulationDataLabelSettings
Visible="true"></AccumulationDataLabelSettings>
<AccumulationChartAnimation Enable="false"></AccumulationChartAnimation>
</AccumulationChartSeries>
</AccumulationChartSeriesCollection>
</SfAccumulationChart>
@code{
public class Statistics
{
public string Browser;
public double Users;
}
public List<Statistics> StatisticsDetails = new List<Statistics>
{
new Statistics { Browser = "Chrome", Users = 37 },
new Statistics { Browser = "UC Browser", Users = 17 },
new Statistics { Browser = "iPhone", Users = 19 },
new Statistics { Browser = "Others", Users = 4 },
new Statistics { Browser = "Opera", Users = 11 },
new Statistics { Browser = "Android", Users = 12 },
};
}

```

Tooltip in Blazor Accumulation Chart Component

The [Enable](#) property in [AccumulationChartTooltipSettings](#) can be set to **true** to enable the tooltip.

ASPX-CS

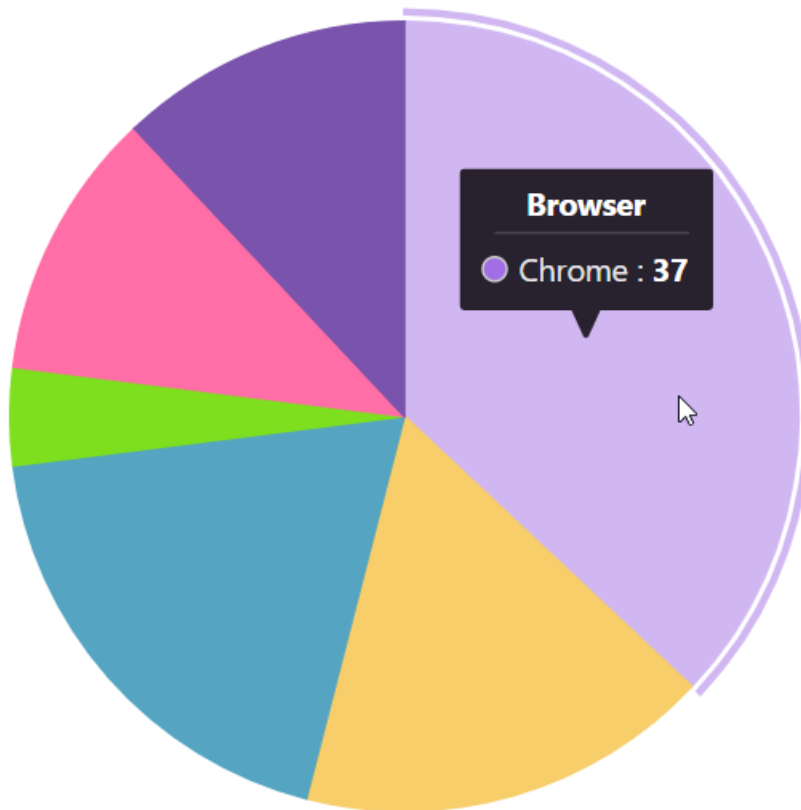
```

@using Syncfusion.Blazor.Charts
<SfAccumulationChart Title="Mobile Browser Statistics">
<AccumulationChartSeriesCollection>
<AccumulationChartSeries DataSource="@StatisticsDetails" XName="Browser"
YName="Users"
Name="Browser">
</AccumulationChartSeries>
</AccumulationChartSeriesCollection>
<AccumulationChartLegendSettings
Visible="false"></AccumulationChartLegendSettings>
<AccumulationChartTooltipSettings
Enable="true"></AccumulationChartTooltipSettings>
</SfAccumulationChart>
@code{
public class Statistics
{
public string Browser { get; set; }
public double Users { get; set; }
}
public List<Statistics> StatisticsDetails = new List<Statistics>
{
new Statistics { Browser = "Chrome", Users = 37 },
new Statistics { Browser = "UC Browser", Users = 17 },
new Statistics { Browser = "iPhone", Users = 19 },
new Statistics { Browser = "Others", Users = 4 },
new Statistics { Browser = "Opera", Users = 11 },
new Statistics { Browser = "Android", Users = 12 },
};
}

```

```
}
```

Mobile Browser Statistics



Header

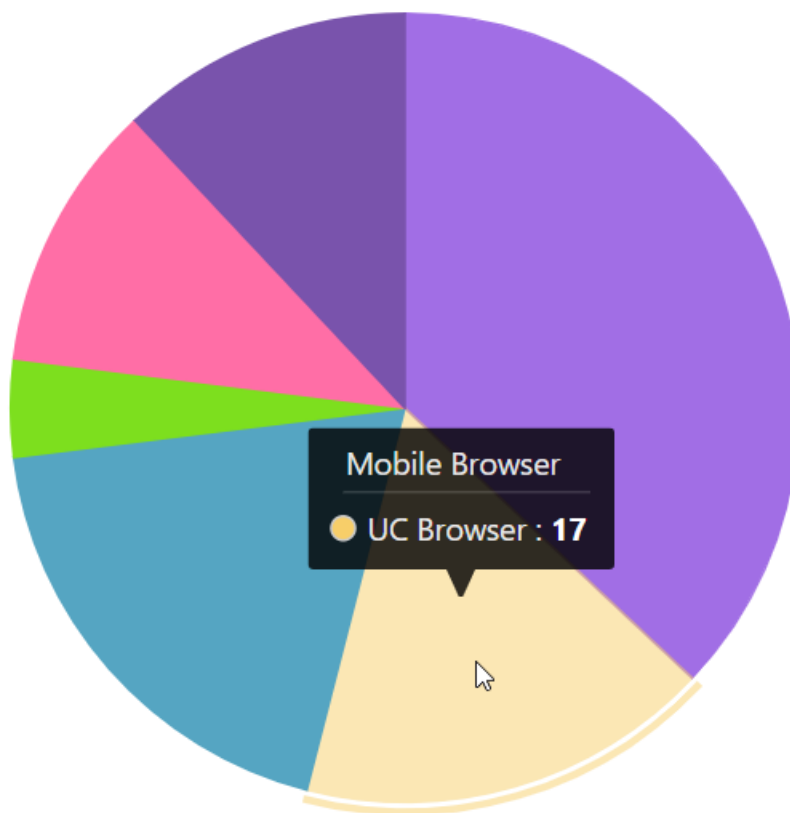
The [Header](#) property in [AccumulationChartTooltipSettings](#) can be used to specify the tooltip's header.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfAccumulationChart Title="Mobile Browser Statistics">
  <AccumulationChartSeriesCollection>
    <AccumulationChartSeries DataSource="@StatisticsDetails" XName="Browser"
      YName="Users"
      Name="Browser">
    </AccumulationChartSeries>
  </AccumulationChartSeriesCollection>
  <AccumulationChartLegendSettings
    Visible="false"></AccumulationChartLegendSettings>
  <AccumulationChartTooltipSettings Enable="true" Header="Mobile
    Browser"></AccumulationChartTooltipSettings>
</SfAccumulationChart>
@code{
```

```
public class Statistics
{
    public string Browser { get; set; }
    public double Users { get; set; }
}
public List<Statistics> StatisticsDetails = new List<Statistics>
{
    new Statistics { Browser = "Chrome", Users = 37 },
    new Statistics { Browser = "UC Browser", Users = 17 },
    new Statistics { Browser = "iPhone", Users = 19 },
    new Statistics { Browser = "Others", Users = 4 },
    new Statistics { Browser = "Opera", Users = 11 },
    new Statistics { Browser = "Android", Users = 12 },
};
```

Mobile Browser Statistics



Tooltip Format

By default, tooltip shows information about x and y value in points. In addition, further customization can be done in the tooltip. For example, the format `${point.x} : ${point.y}%` shows point x value and customized point y value.

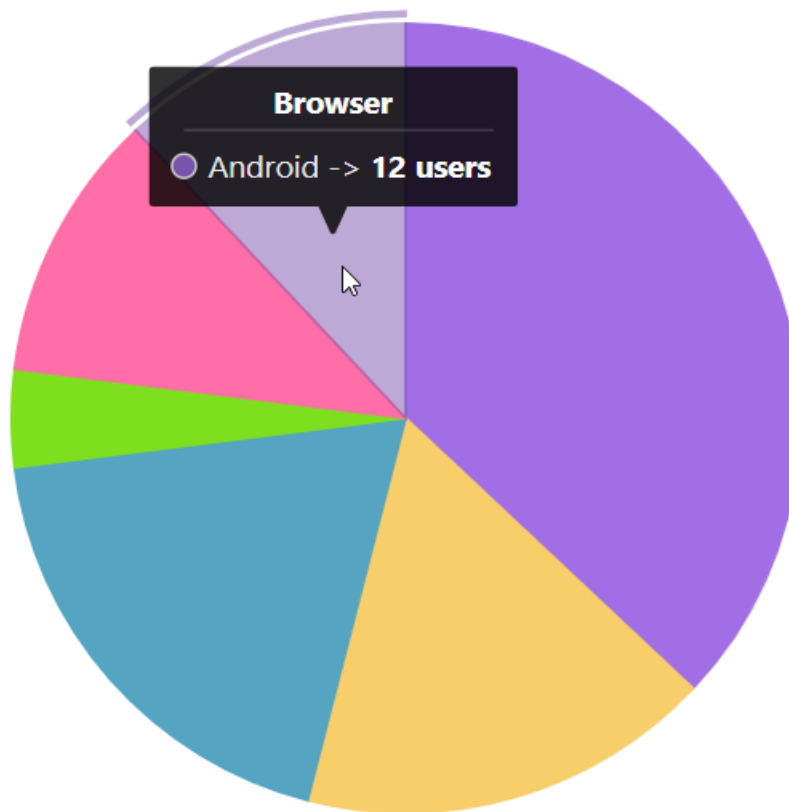
ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfAccumulationChart Title="Mobile Browser Statistics">
  <AccumulationChartSeriesCollection>
    <AccumulationChartSeries DataSource="@StatisticsDetails" XName="Browser"
      YName="Users"
      Name="Browser">
    </AccumulationChartSeries>
  </AccumulationChartSeriesCollection>
  <AccumulationChartLegendSettings
    Visible="false"></AccumulationChartLegendSettings>
  <AccumulationChartTooltipSettings Enable="true" Format="${point.x} ->
    <b>${point.y} users</b>"></AccumulationChartTooltipSettings>
</SfAccumulationChart>

@code{
public class Statistics
{
public string Browser { get; set; }
public double Users { get; set; }
}

public List<Statistics> StatisticsDetails = new List<Statistics>
{
new Statistics { Browser = "Chrome", Users = 37 },
new Statistics { Browser = "UC Browser", Users = 17 },
new Statistics { Browser = "iPhone", Users = 19 },
new Statistics { Browser = "Others", Users = 4 },
new Statistics { Browser = "Opera", Users = 11 },
new Statistics { Browser = "Android", Users = 12 },
};
}
```


Mobile Browser Statistics



Tooltip Customization

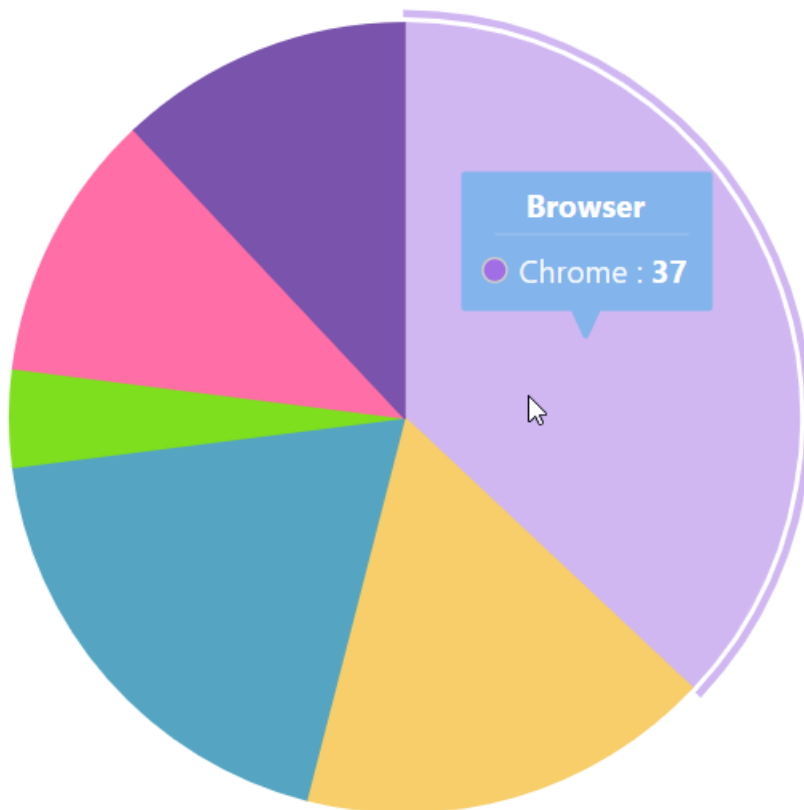
The [Fill](#) and [AccumulationChartTooltipBorder](#) are used to customize the background color and the border of the tooltip respectively. The [AccumulationChartTooltipTextStyle](#) in the tooltip is used to customize the font size of the tooltip text.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfAccumulationChart Title="Mobile Browser Statistics">
  <AccumulationChartSeriesCollection>
    <AccumulationChartSeries DataSource="@StatisticsDetails" XName="Browser"
      YName="Users"
      Name="Browser">
    </AccumulationChartSeries>
  </AccumulationChartSeriesCollection>
  <AccumulationChartLegendSettings
    Visible="false"></AccumulationChartLegendSettings>
  <AccumulationChartTooltipSettings Enable="true" Format="{point.x} :
    <b>{point.y}</b>" Fill="#7bb4eb">
    <AccumulationChartTooltipBorder Color="red"
      Width="2"></AccumulationChartTooltipBorder>
  </AccumulationChartTooltipSettings>
```

```
</SfAccumulationChart>
@code{
public class Statistics
{
    public string Browser { get; set; }
    public double Users { get; set; }
}
public List<Statistics> StatisticsDetails = new List<Statistics>
{
    new Statistics { Browser = "Chrome", Users = 37 },
    new Statistics { Browser = "UC Browser", Users = 17 },
    new Statistics { Browser = "iPhone", Users = 19 },
    new Statistics { Browser = "Others", Users = 4 },
    new Statistics { Browser = "Opera", Users = 11 },
    new Statistics { Browser = "Android", Users = 12 },
};
}
```

Mobile Browser Statistics



Tooltip Text Mapping

By default, tooltip shows information of x and y value in points. In addition, by using the [TooltipMappingName](#), more information from the datasource can be displayed in the tooltip. To display the specified tooltip content, **\$point.tooltip** can be used as a placeholder.

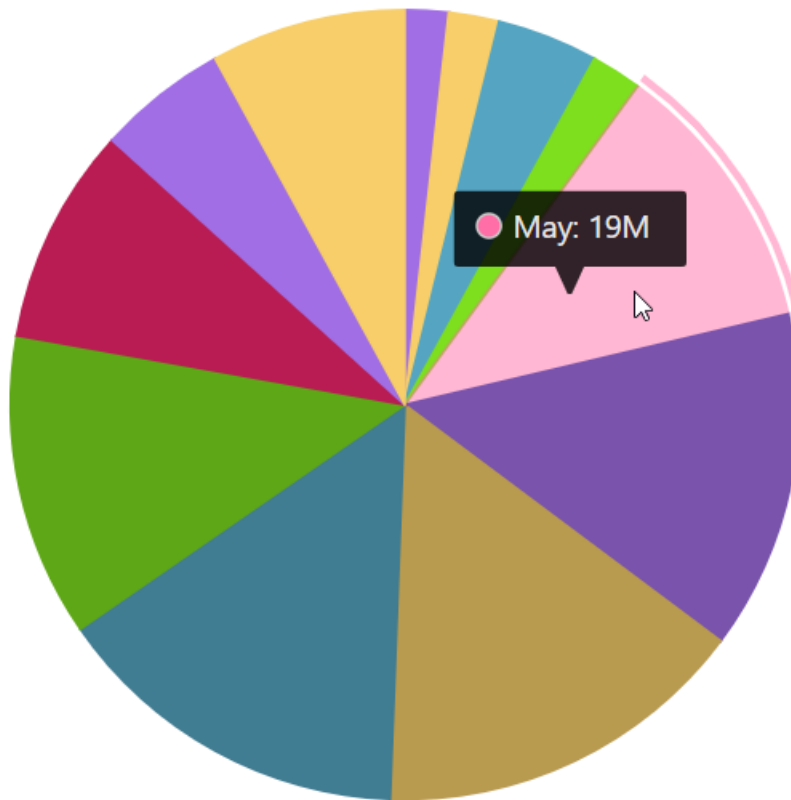
ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfAccumulationChart Title="Sales Analysis">
  <AccumulationChartSeriesCollection>
    <AccumulationChartSeries DataSource="@StatisticsDetails" XName="XValue"
      YName="YValue"
      TooltipMappingName="Text">
    </AccumulationChartSeries>
  </AccumulationChartSeriesCollection>
  <AccumulationChartLegendSettings
    Visible="false"></AccumulationChartLegendSettings>
  <AccumulationChartTooltipSettings Enable="true" Format="{point.tooltip}"
  ></AccumulationChartTooltipSettings>
</SfAccumulationChart>

@code{
public class ChartData
{
public string XValue { get; set; }
public double YValue { get; set; }
public string Text { get; set; }
}

public List<ChartData> DataSource = new List<ChartData>
{
new ChartData { XValue = "Jan", YValue = 3, Text= "Jan: 3M" },
new ChartData { XValue = "Feb", YValue = 3.5, Text= "Feb: 3.5M" },
new ChartData { XValue = "Mar", YValue = 7, Text= "Mar: 7M" },
new ChartData { XValue = "Apr", YValue = 3.5, Text= "Apr: 13.5M" },
new ChartData { XValue = "May", YValue = 19, Text= "May: 19M" },
new ChartData { XValue = "Jun", YValue = 23.5, Text= "Jun: 23.5M" },
new ChartData { XValue = "Jul", YValue = 26, Text= "Jul: 26M" },
new ChartData { XValue = "Aug", YValue = 25, Text= "Aug: 25M" },
new ChartData { XValue = "Sep", YValue = 21, Text= "Sep: 21M" },
new ChartData { XValue = "Oct", YValue = 15, Text= "Oct: 15M" },
new ChartData { XValue = "Nov", YValue = 9, Text= "Nov: 9M" },
new ChartData { XValue = "Dec", YValue = 13.5, Text= "Dec: 3.5M" }
};
}
```

Sales Analysis



Refer to the [Blazor Charts](#) feature tour page for its groundbreaking feature representations and also explore the [Blazor Accumulation Chart Example](#) to know about the various features of accumulation charts and how it is used to represent numeric proportional data.

See Also

- [Grouping](#)
- [Data label](#)

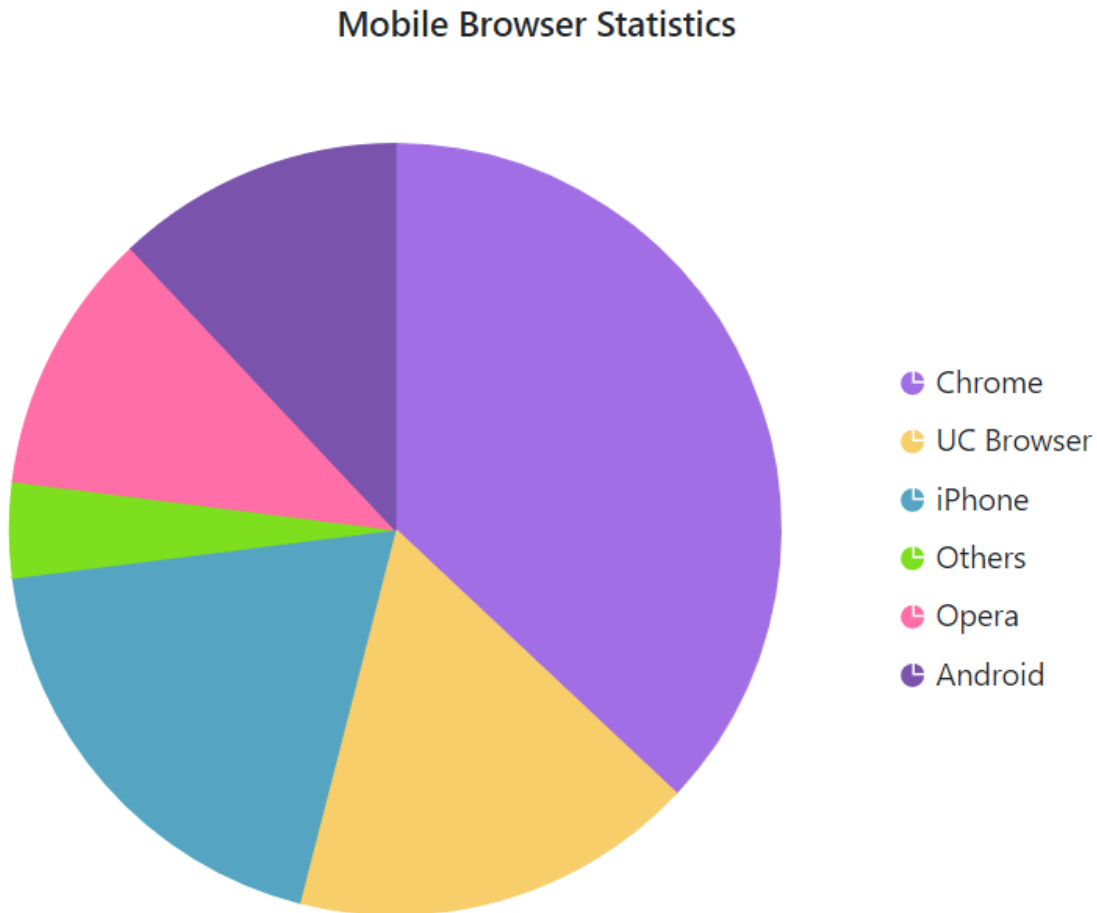
Legend in Blazor Accumulation Chart Component

The legend is available for accumulation charts, just like it is for charts, and it provides information about the points. If the chart's width is large, the legend will be placed on the right, and if the chart's height is large, the legend will be placed on the bottom. The legend for a point can be collapsed by assigning an empty string to the point's x value.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfAccumulationChart Title="Mobile Browser Statistics">
```

```
<AccumulationChartSeriesCollection>
<AccumulationChartSeries DataSource="@StatisticsDetails" XName="Browser"
YName="Users"
Name="Browser">
</AccumulationChartSeries>
</AccumulationChartSeriesCollection>
<AccumulationChartLegendSettings
Visible="true"></AccumulationChartLegendSettings>
</SfAccumulationChart>
@code{
public class Statistics
{
public string Browser { get; set; }
public double Users { get; set; }
}
public List<Statistics> StatisticsDetails = new List<Statistics>
{
new Statistics { Browser = "Chrome", Users = 37 },
new Statistics { Browser = "UC Browser", Users = 17 },
new Statistics { Browser = "iPhone", Users = 19 },
new Statistics { Browser = "Others", Users = 4 },
new Statistics { Browser = "Opera", Users = 11 },
new Statistics { Browser = "Android", Users = 12 },
};
}
```



Position and Alignment

The legend can be placed at [Left](#), [Right](#), [Top](#) or [Bottom](#) [Custom](#) position of the chart using the [Position](#) property. The [Alignment](#) property can also be used to align the legend to the chart's [Center](#), [Far](#) or [Near](#).

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfAccumulationChart Title="Mobile Browser Statistics">
  <AccumulationChartSeriesCollection>
    <AccumulationChartSeries DataSource="@StatisticsDetails" XName="Browser"
      YName="Users"
      Name="Browser">
    </AccumulationChartSeries>
  </AccumulationChartSeriesCollection>
  <AccumulationChartLegendSettings Visible="true"
    Position="LegendPosition.Top"></AccumulationChartLegendSettings>
</SfAccumulationChart>

@code{
  public class Statistics
  {
    public string Browser { get; set; }
    public double Users { get; set; }
  }
}
```

```
public List<Statistics> StatisticsDetails = new List<Statistics>
{
    new Statistics { Browser = "Chrome", Users = 37 },
    new Statistics { Browser = "UC Browser", Users = 17 },
    new Statistics { Browser = "iPhone", Users = 19 },
    new Statistics { Browser = "Others", Users = 4 },
    new Statistics { Browser = "Opera", Users = 11 },
    new Statistics { Browser = "Android", Users = 12 },
};
```

Mobile Browser Statistics

Chrome UC Browser iPhone Others Opera Android



Legend Shape

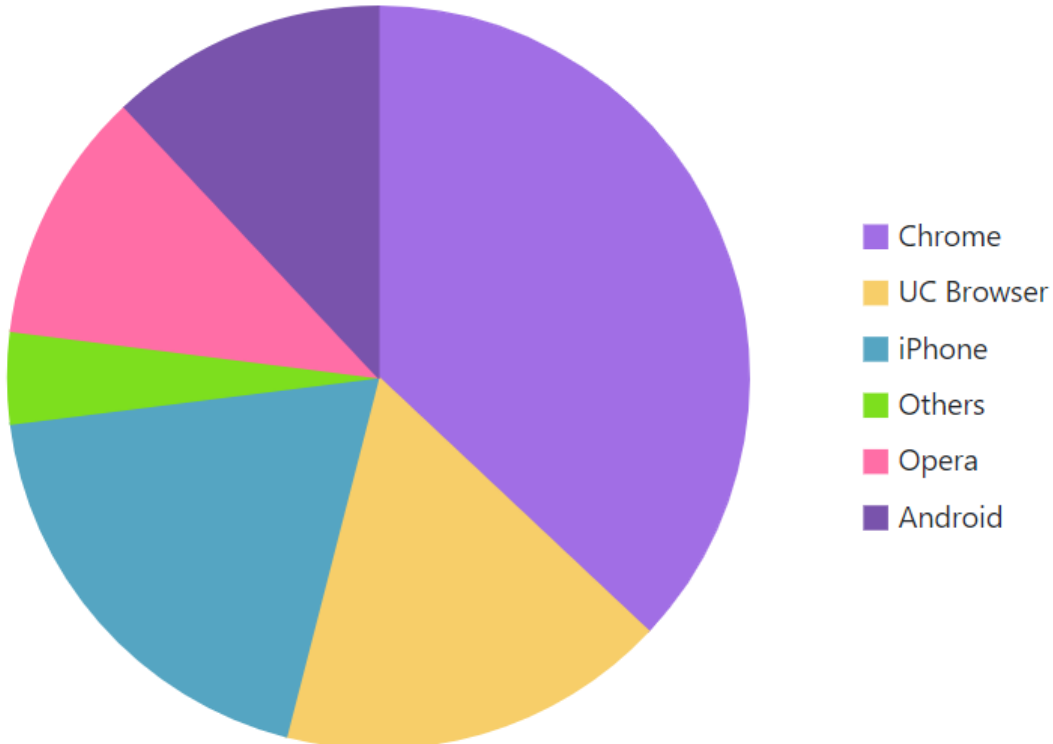
The [LegendShape](#) property in the [Series](#) can be used to change the shape of the legend icon. The default icon shape for legends is [SeriesType](#).

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfAccumulationChart Title="Mobile Browser Statistics">
  <AccumulationChartSeriesCollection>
    <AccumulationChartSeries LegendShape="LegendShape.Rectangle"
      DataSource="@StatisticsDetails" XName="Browser" YName="Users"
```

```
Name="Browser">
</AccumulationChartSeries>
</AccumulationChartSeriesCollection>
<AccumulationChartLegendSettings
Visible="true"></AccumulationChartLegendSettings>
</SfAccumulationChart>
@code{
public class Statistics
{
public string Browser { get; set; }
public double Users { get; set; }
}
public List<Statistics> StatisticsDetails = new List<Statistics>
{
new Statistics { Browser = "Chrome", Users = 37 },
new Statistics { Browser = "UC Browser", Users = 17 },
new Statistics { Browser = "iPhone", Users = 19 },
new Statistics { Browser = "Others", Users = 4 },
new Statistics { Browser = "Opera", Users = 11 },
new Statistics { Browser = "Android", Users = 12 },
};
}
```

Mobile Browser Statistics



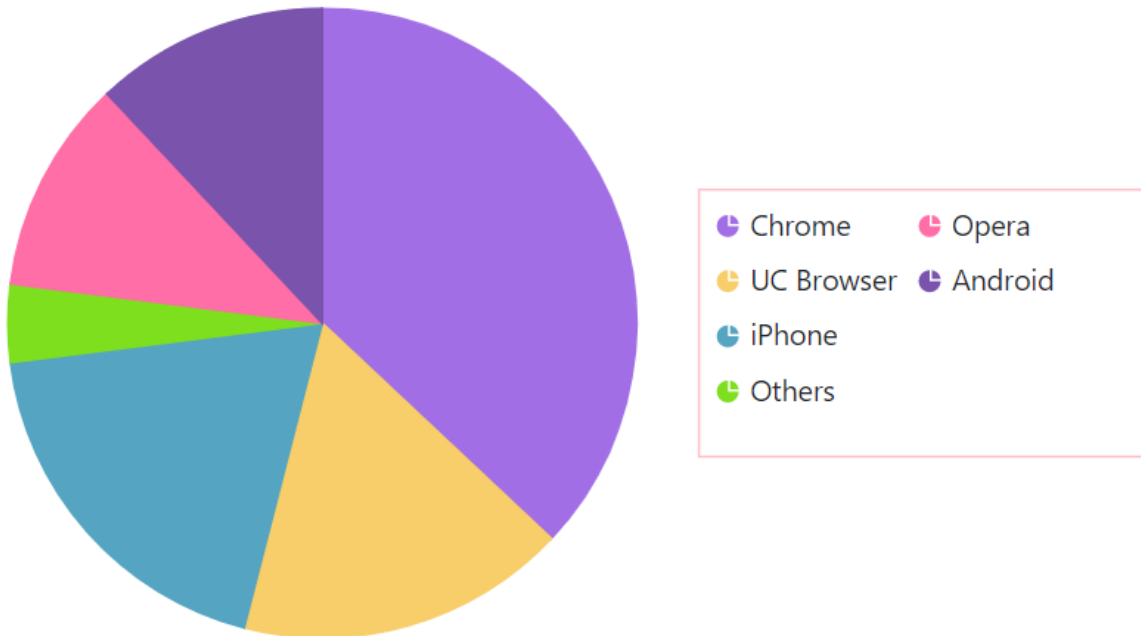
Legend Size

The legend size can be customized by using the [Width](#) and [Height](#) properties of the [AccumulationChartLegendSettings](#).

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfAccumulationChart Title="Mobile Browser Statistics">
  <AccumulationChartSeriesCollection>
    <AccumulationChartSeries DataSource="@StatisticsDetails" XName="Browser"
      YName="Users"
      Name="Browser">
    </AccumulationChartSeries>
  </AccumulationChartSeriesCollection>
  <AccumulationChartLegendSettings Visible="true" Height="28%" Width="44%">
    <AccumulationChartLegendBorder Color="Pink"
      Width="1"></AccumulationChartLegendBorder>
  </AccumulationChartLegendSettings>
</SfAccumulationChart>
@code{
public class Statistics
{
  public string Browser { get; set; }
  public double Users { get; set; }
}
public List<Statistics> StatisticsDetails = new List<Statistics>
{
  new Statistics { Browser = "Chrome", Users = 37 },
  new Statistics { Browser = "UC Browser", Users = 17 },
  new Statistics { Browser = "iPhone", Users = 19 },
  new Statistics { Browser = "Others", Users = 4 },
  new Statistics { Browser = "Opera", Users = 11 },
  new Statistics { Browser = "Android", Users = 12 },
};
}
```

Mobile Browser Statistics



Legend Shape Size

The [ShapeHeight](#) and [ShapeWidth](#) properties can be used to adjust the dimensions of the legend shape.

ASPX-CS

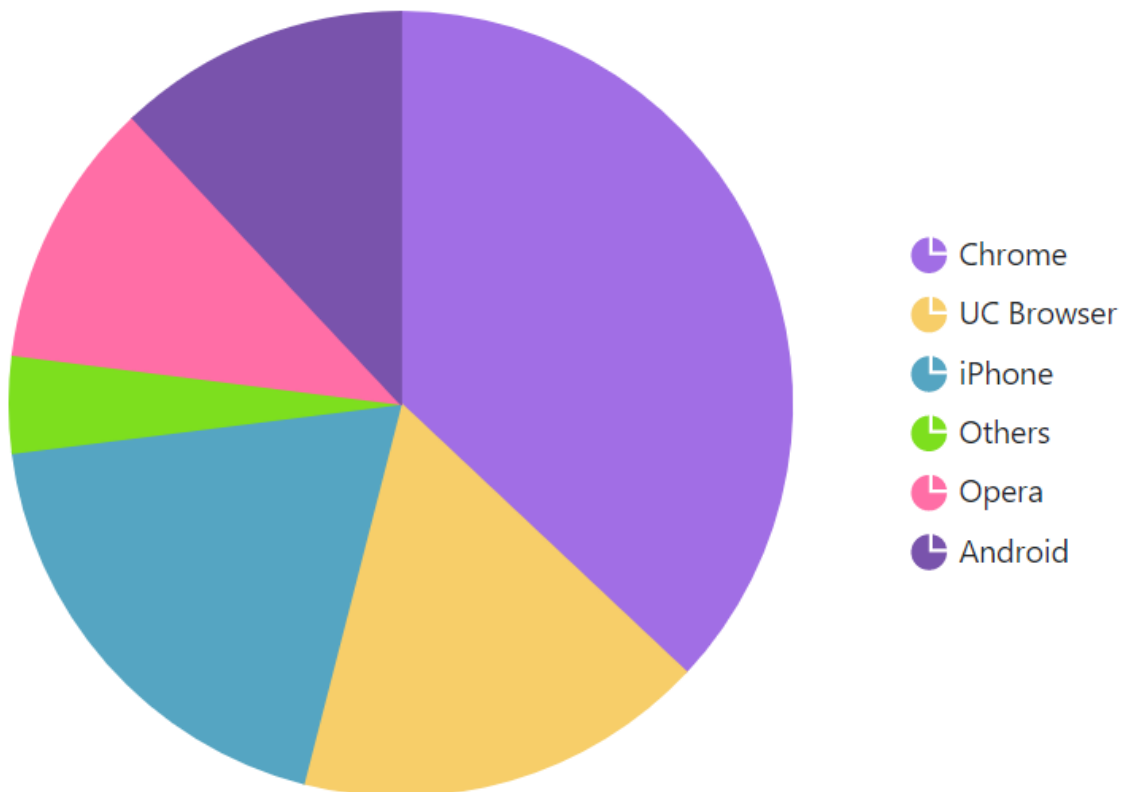
```
@using Syncfusion.Blazor.Charts
<SfAccumulationChart Title="Mobile Browser Statistics">
  <AccumulationChartSeriesCollection>
    <AccumulationChartSeries DataSource="@StatisticsDetails" XName="Browser"
      YName="Users"
      Name="Browser">
    </AccumulationChartSeries>
  </AccumulationChartSeriesCollection>
  <AccumulationChartLegendSettings Visible="true" ShapeHeight="15"
    ShapeWidth="15">
  </AccumulationChartLegendSettings>
</SfAccumulationChart>
@code{
public class Statistics
{
  public string Browser { get; set; }
  public double Users { get; set; }
}
public List<Statistics> StatisticsDetails = new List<Statistics>
{
  new Statistics { Browser = "Chrome", Users = 37 },
  new Statistics { Browser = "UC Browser", Users = 17 },
```

```

new Statistics { Browser = "iPhone", Users = 19 },
new Statistics { Browser = "Others", Users = 4 },
new Statistics { Browser = "Opera", Users = 11 },
new Statistics { Browser = "Android", Users = 12 },
};

```

Mobile Browser Statistics



Paging for Legend

When the legend items exceed legend bounds, paging will be enabled by default. End user can view each legend item using the navigation buttons to navigate between pages.

ASPX-CS

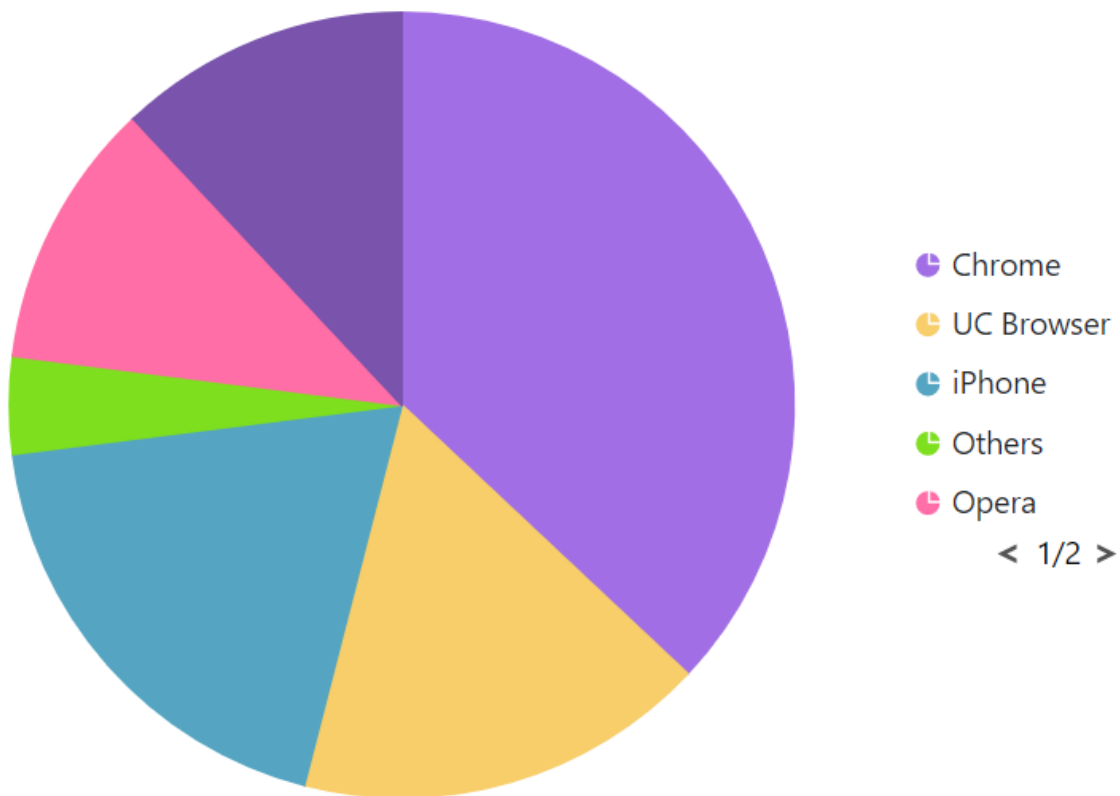
```

@using Syncfusion.Blazor.Charts
<SfAccumulationChart Title="Mobile Browser Statistics">
  <AccumulationChartSeriesCollection>
    <AccumulationChartSeries DataSource="@StatisticsDetails" XName="Browser"
      YName="Users"
      Name="Browser">
    </AccumulationChartSeries>
  </AccumulationChartSeriesCollection>

```

```
<AccumulationChartLegendSettings Visible="true" Height="150" Width="100">
</AccumulationChartLegendSettings>
</SfAccumulationChart>
@code{
public class Statistics
{
public string Browser{ get; set; }
public double Users { get; set; }
}
public List<Statistics> StatisticsDetails = new List<Statistics>
{
new Statistics { Browser = "Chrome", Users = 37 },
new Statistics { Browser = "UC Browser", Users = 17 },
new Statistics { Browser = "iPhone", Users = 19 },
new Statistics { Browser = "Others", Users = 4 },
new Statistics { Browser = "Opera", Users = 11 },
new Statistics { Browser = "Android", Users = 12 },
};
}
```

Mobile Browser Statistics



Refer to the [Blazor Charts](#) feature tour page for its groundbreaking feature representations and also explore the [Blazor Accumulation Chart Example](#) to know about the various features of accumulation charts and how it is used to represent numeric proportional data.

- [Grouping](#)
- [Data label](#)

<!-- markdownlint-disable MD036 -->

Title and Subtitle in Blazor Accumulation Chart Component

The [Title](#) property can be used to give the accumulation chart a title in-order to provide information about the data displayed.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfAccumulationChart Title="Mobile Browser Statistics">
<AccumulationChartSeriesCollection>
<AccumulationChartSeries DataSource="@StatisticsDetails" XName="Browser"
YName="Users"
Name="Browser">
</AccumulationChartSeries>
</AccumulationChartSeriesCollection>
<AccumulationChartLegendSettings
Visible="false"></AccumulationChartLegendSettings>
</SfAccumulationChart>
@code{
public class Statistics
{
public string Browser { get; set; }
public double Users { get; set; }
}
public List<Statistics> StatisticsDetails = new List<Statistics>
{
new Statistics { Browser = "Chrome", Users = 37 },
new Statistics { Browser = "UC Browser", Users = 17 },
new Statistics { Browser = "iPhone", Users = 19 },
new Statistics { Browser = "Others", Users = 4 },
new Statistics { Browser = "Opera", Users = 11 },
new Statistics { Browser = "Android", Users = 12 },
};
}
```

Mobile Browser Statistics



Title Customization

The [Title](#) can be customized to be defined in the [AccumulationChartTitleStyle](#).

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfAccumulationChart Title="Mobile Browser Statistics">
  <AccumulationChartTitleStyle FontFamily="Arial" FontWeight="regular"
  Color="#E27F2D" Size="23px"></AccumulationChartTitleStyle>
  <AccumulationChartSeriesCollection>
    <AccumulationChartSeries DataSource="@StatisticsDetails" XName="Browser"
    YName="Users"
    Name="Browser">
  </AccumulationChartSeries>
  </AccumulationChartSeriesCollection>
  <AccumulationChartLegendSettings
  Visible="false"></AccumulationChartLegendSettings>
</SfAccumulationChart>
@code{
public class Statistics
{
  public string Browser { get; set; }
  public double Users { get; set; }
}
```

```
}  
public List<Statistics> StatisticsDetails = new List<Statistics>  
{  
    new Statistics { Browser = "Chrome", Users = 37 },  
    new Statistics { Browser = "UC Browser", Users = 17 },  
    new Statistics { Browser = "iPhone", Users = 19 },  
    new Statistics { Browser = "Others", Users = 4 },  
    new Statistics { Browser = "Opera", Users = 11 },  
    new Statistics { Browser = "Android", Users = 12 },  
};  
}
```

Mobile Browser Statistics



Subtitle

The [SubTitle](#) property can be used to give the accumulation chart a subtitle in-order to provide an additional information about the data displayed.

ASPX-CS

```
@using Syncfusion.Blazor.Charts  
<SfAccumulationChart Title="Mobile Browser Statistics" SubTitle="In the year  
2014 - 2015">
```

```
<AccumulationChartLegendSettings
Visible="false"></AccumulationChartLegendSettings>
<AccumulationChartSeriesCollection>
<AccumulationChartSeries DataSource="@StatisticsDetails" XName="Browser"
YName="Users">
</AccumulationChartSeries>
</AccumulationChartSeriesCollection>
</SfAccumulationChart>
@code{
public class Statistics
{
public string Browser { get; set; }
public double Users { get; set; }
}
public List<Statistics> StatisticsDetails = new List<Statistics>
{
new Statistics { Browser = "Chrome", Users = 37 },
new Statistics { Browser = "UC Browser", Users = 17 },
new Statistics { Browser = "iPhone", Users = 19 },
new Statistics { Browser = "Others", Users = 4 },
new Statistics { Browser = "Opera", Users = 11 },
new Statistics { Browser = "Android", Users = 12 },
};
}
```


Mobile Browser Statistics In the year 2014 - 2015



Subtitle Customization

The [SubTitle](#) can be customized to be defined in the [AccumulationChartSubTitleStyle](#).

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfAccumulationChart Title="Mobile Browser Statistics" SubTitle="In the year
2014 - 2015">
  <AccumulationChartSubTitleStyle FontFamily="Arial" FontWeight="regular"
  Color="#E27F2D" Size="13px"></AccumulationChartSubTitleStyle>
  <AccumulationChartLegendSettings
  Visible="false"></AccumulationChartLegendSettings>
  <AccumulationChartSeriesCollection>
    <AccumulationChartSeries DataSource="@StatisticsDetails" XName="Browser"
    YName="Users">
    </AccumulationChartSeries>
  </AccumulationChartSeriesCollection>
</SfAccumulationChart>
@code{
public class Statistics
{
public string Browser { get; set; }
}
```

```
public double Users { get; set; }  
}  
public List<Statistics> StatisticsDetails = new List<Statistics>  
{  
    new Statistics { Browser = "Chrome", Users = 37 },  
    new Statistics { Browser = "UC Browser", Users = 17 },  
    new Statistics { Browser = "iPhone", Users = 19 },  
    new Statistics { Browser = "Others", Users = 4 },  
    new Statistics { Browser = "Opera", Users = 11 },  
    new Statistics { Browser = "Android", Users = 12 },  
};  
}
```

Mobile Browser Statistics In the year 2014 - 2015



Refer to the [Blazor Charts](#) feature tour page for its groundbreaking feature representations and also explore the [Blazor Accumulation Chart Example](#) to know various features of accumulation charts and how it is used to represent numeric proportional data.

See Also

- [Grouping](#)

- [Data label](#)

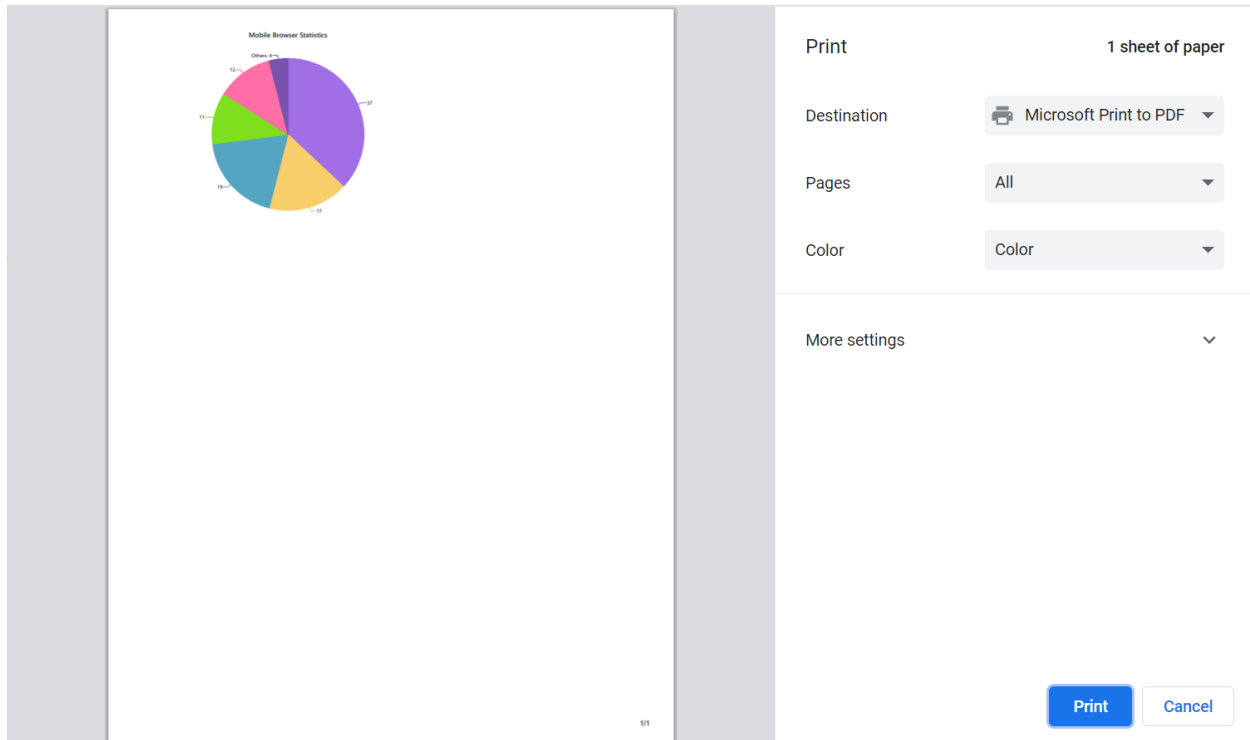
Print and Export in Blazor Accumulation Chart Component

Print

The `PrintAsync` method can be used to print a rendered chart directly from the browser.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
@using Syncfusion.Blazor.Buttons
<SfAccumulationChart @ref="ChartObj" Title="Mobile Browser Statistics"
EnableSmartLabels="true">
<AccumulationChartLegendSettings
Visible="false"></AccumulationChartLegendSettings>
<AccumulationChartSeriesCollection>
<AccumulationChartSeries DataSource="@StatisticsDetails" XName="Browser"
YName="Users" Name="RIO" GroupTo="10">
<AccumulationDataLabelSettings Visible="true" Name="Text"
Position="AccumulationLabelPosition.Outside">
</AccumulationDataLabelSettings>
</AccumulationChartSeries>
</AccumulationChartSeriesCollection>
</SfAccumulationChart>
<SfButton ID="button" Content="Print" @onclick="@Click" IsPrimary="true"
CssClass="e-flat"></SfButton>
@code{
SfAccumulationChart ChartObj;
private async Task Click()
{
await ChartObj.PrintAsync();
}
public class Statistics
{
public string Browser { get; set; }
public double Users { get; set; }
}
public List<Statistics> StatisticsDetails = new List<Statistics>
{
new Statistics { Browser = "Chrome", Users = 37 },
new Statistics { Browser = "UC Browser", Users = 17 },
new Statistics { Browser = "iPhone", Users = 19 },
new Statistics { Browser = "Others", Users = 4 },
new Statistics { Browser = "Opera", Users = 11 },
new Statistics { Browser = "Android", Users = 12 },
};
}
```



Export

Using the `ExportAsync` method, the rendered chart can be exported to [JPEG](#), [PNG](#), [SVG](#), or [PDF](#) format. The `Export Type` specifies the image format and `FileName` specifies the name of the exported file. Both of these parameters are required input parameters for this method.

The optional parameters for this method are,

- `Orientation` - Specifies the portrait or landscape orientation in the PDF document.
- `AllowDownload` - Specifies whether to download or not. If not, base64 string will be returned.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
@using Syncfusion.Blazor.Buttons
<SfAccumulationChart @ref="ChartObj" Title="Mobile Browser Statistics"
EnableSmartLabels="true">
  <AccumulationChartLegendSettings
Visible="false"></AccumulationChartLegendSettings>
  <AccumulationChartSeriesCollection>
    <AccumulationChartSeries DataSource="@StatisticsDetails" XName="Browser"
YName="Users" Name="RIO" GroupTo="10">
      <AccumulationDataLabelSettings Visible="true" Name="Text"
Position="AccumulationLabelPosition.Outside">
      </AccumulationDataLabelSettings>
    </AccumulationChartSeries>
  </AccumulationChartSeriesCollection>
</SfAccumulationChart>
<SfButton ID="button" Content="Export" @onclick="@Click" IsPrimary="true"
CssClass="e-flat"></SfButton>
@code{
```

```
SfAccumulationChart ChartObj;
private async Task Click()
{
    await ChartObj.ExportAsync(ExportType.JPEG, "chart");
}
public class Statistics
{
    public string Browser { get; set; }
    public double Users { get; set; }
}
public List<Statistics> StatisticsDetails = new List<Statistics>
{
    new Statistics { Browser = "Chrome", Users = 37 },
    new Statistics { Browser = "UC Browser", Users = 17 },
    new Statistics { Browser = "iPhone", Users = 19 },
    new Statistics { Browser = "Others", Users = 4 },
    new Statistics { Browser = "Opera", Users = 11 },
    new Statistics { Browser = "Android", Users = 12 },
};
}
```

Refer to the [Blazor Charts](#) feature tour page for its groundbreaking feature representations and also explore the [Blazor Chart Example](#) to know various chart types and how to represent time-dependent data, showing trends at equal intervals.

See Also

- [Data Label](#)
- [Tooltip](#)
- [Legend](#)

Events in Blazor Accumulation Chart Component

In this section, the list of events of Accumulation Chart component is provided which will be triggered for appropriate accumulation chart actions.

The events should be provided to the Accumulation Chart using [AccumulationChartEvents](#) component.

From v18.4.*, a few additional following events are added to the Accumulation Chart component.

Event Name |

AfterExport | [OnExportComplete](#)

OnPrint | [OnPrintComplete](#)

Resized | [SizeChanged](#)

From v18.4.*, the following previous release events are removed from the Accumulation Chart component.

Event Name |

OnChartMouseClicked |

OnChartMouseDown |

OnChartMouseLeave|

OnChartMouseMove|

OnChartMouseUp|

PointMoved|

OnDataLabelRender

[OnDataLabelRender](#) event triggers, before datalabel for series is rendered.

Arguments

The following properties are available in the [AccumulationTextRenderEventArgs](#).

- [Color](#) – Specifies the color for the data label text.
- [Border](#) – Specifies the color and the width of the data label border.
- [Font](#) – Specifies the font information of the data label.
- [Text](#) – Specifies the text to be displayed in the data label.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfAccumulationChart Title="Mobile Browser Statistics">
  <AccumulationChartEvents
    OnDataLabelRender="DataLabelRenderEvent"></AccumulationChartEvents>
  <AccumulationChartSeriesCollection>
    <AccumulationChartSeries DataSource="@StatisticsDetails" XName="Browser"
      YName="Users">
      <AccumulationDataLabelSettings
        Visible="true"></AccumulationDataLabelSettings>
    </AccumulationChartSeries>
  </AccumulationChartSeriesCollection>
</SfAccumulationChart>

@code{
public class Statistics
{
public string Browser { get; set; }
public double Users { get; set; }
}
public List<Statistics> StatisticsDetails = new List<Statistics>
{
new Statistics { Browser = "Chrome", Users = 37 },
new Statistics { Browser = "UC Browser", Users = 17 },
new Statistics { Browser = "iPhone", Users = 19 },
new Statistics { Browser = "Others", Users = 4 },
new Statistics { Browser = "Opera", Users = 11 },
new Statistics { Browser = "Android", Users = 12 },
};
public void DataLabelRenderEvent(AccumulationTextRenderEventArgs args)
{
// Here you can customize your code
}
}
```

OnLegendItemRender

[OnLegendItemRender](#) event triggers, before legend getting rendered.

Arguments

The following properties are available in the [AccumulationLegendRenderEventArgs](#).

- [Fill](#) – Specifies the fill color of the legend item's icon.
- [Shape](#) – Specifies the shape of the legend item's icon.
- [Text](#) – Specifies the text to be displayed in the legend item.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfAccumulationChart Title="Mobile Browser Statistics">
  <AccumulationChartEvents
    OnLegendItemRender="LegendRenderEvent"></AccumulationChartEvents>
  <AccumulationChartSeriesCollection>
    <AccumulationChartSeries DataSource="@StatisticsDetails" XName="Browser"
      YName="Users"
      Name="Browser">
    </AccumulationChartSeries>
  </AccumulationChartSeriesCollection>
  <AccumulationChartLegendSettings
    Visible="true"></AccumulationChartLegendSettings>
</SfAccumulationChart>

@code{
public class Statistics
{
    public string Browser { get; set; }
    public double Users { get; set; }
}

public List<Statistics> StatisticsDetails = new List<Statistics>
{
    new Statistics { Browser = "Chrome", Users = 37 },
    new Statistics { Browser = "UC Browser", Users = 17 },
    new Statistics { Browser = "iPhone", Users = 19 },
    new Statistics { Browser = "Others", Users = 4 },
    new Statistics { Browser = "Opera", Users = 11 },
    new Statistics { Browser = "Android", Users = 12 },
};

public void LegendRenderEvent(AccumulationLegendRenderEventArgs args)
{
    // Here you can customize your code
}
```

OnPointRender

[OnPointRender](#) event triggers, before each points for the accumulation chart is rendered.

Arguments

The following properties are available in the [AccumulationPointRenderEventArgs](#).

- [Border](#) – Specifies the color and the width of the point border.
- [Fill](#) – Specifies the fill color of the point.

ASPX-CS

```

@using Syncfusion.Blazor.Charts
<SfAccumulationChart Title="Mobile Browser Statistics">
<AccumulationChartEvents
OnPointRender="PointRenderEvent"></AccumulationChartEvents>
<AccumulationChartSeriesCollection>
<AccumulationChartSeries DataSource="@StatisticsDetails" XName="Browser"
YName="Users">
</AccumulationChartSeries>
</AccumulationChartSeriesCollection>
</SfAccumulationChart>
@code{
public class Statistics
{
public string Browser { get; set; }
public double Users { get; set; }
}
public List<Statistics> StatisticsDetails = new List<Statistics>
{
new Statistics { Browser = "Chrome", Users = 37 },
new Statistics { Browser = "UC Browser", Users = 17 },
new Statistics { Browser = "iPhone", Users = 19 },
new Statistics { Browser = "Others", Users = 4 },
new Statistics { Browser = "Opera", Users = 11 },
new Statistics { Browser = "Android", Users = 12 },
};
public void PointRenderEvent(AccumulationPointRenderEventArgs args)
{
// Here you can customize your code
}
}

```

OnExportComplete

[OnExportComplete](#) event triggers after exporting the accumulation chart.

Arguments

The following field is available in the [ExportEventArgs](#).

- [DataUrl](#) – Specifies the DataUrl of the exported file.

ASPX-CS

```

@using Syncfusion.Blazor.Charts
<button @onclick="Export" class="btn-success">Export</button>
<SfAccumulationChart Title="Mobile Browser Statistics" @ref="AccChart">
<AccumulationChartEvents
OnExportComplete="ExportCompleteEvent"></AccumulationChartEvents>
<AccumulationChartSeriesCollection>
<AccumulationChartSeries DataSource="@StatisticsDetails" XName="Browser"
YName="Users">
</AccumulationChartSeries>
</AccumulationChartSeriesCollection>
</SfAccumulationChart>
@code{

```



```

SfAccumulationChart AccChart;
public class Statistics
{
    public string Browser { get; set; }
    public double Users { get; set; }
}
public List<Statistics> StatisticsDetails = new List<Statistics>
{
    new Statistics { Browser = "Chrome", Users = 37 },
    new Statistics { Browser = "UC Browser", Users = 17 },
    new Statistics { Browser = "iPhone", Users = 19 },
    new Statistics { Browser = "Others", Users = 4 },
    new Statistics { Browser = "Opera", Users = 11 },
    new Statistics { Browser = "Android", Users = 12 },
};
public void Export()
{
    AccChart.Export(ExportType.JPEG, "Charts");
}
public void ExportCompleteEvent(ExportEventArgs args)
{
    // Here you can customize your code
}
}

```

OnPrintComplete

OnPrintComplete event triggers, after printing the accumulation chart

ASPX-CS

```

@using Syncfusion.Blazor.Charts
<button onclick="Print" class="btn-success">Print</button>
<SfAccumulationChart Title="Mobile Browser Statistics" @ref="AccChart">
    <AccumulationChartEvents
        OnPrintComplete="PrintCompleteEvent"></AccumulationChartEvents>
    <AccumulationChartSeriesCollection>
        <AccumulationChartSeries DataSource="@StatisticsDetails" XName="Browser"
            YName="Users">
        </AccumulationChartSeries>
    </AccumulationChartSeriesCollection>
</SfAccumulationChart>
@code{
    SfAccumulationChart AccChart;
    public class Statistics
    {
        public string Browser { get; set; }
        public double Users { get; set; }
    }
    public List<Statistics> StatisticsDetails = new List<Statistics>
    {
        new Statistics { Browser = "Chrome", Users = 37 },
        new Statistics { Browser = "UC Browser", Users = 17 },
        new Statistics { Browser = "iPhone", Users = 19 },
        new Statistics { Browser = "Others", Users = 4 },
        new Statistics { Browser = "Opera", Users = 11 },
        new Statistics { Browser = "Android", Users = 12 },
    }
}

```

```
};
public void Print()
{
    AccChart.Print();
}
public void PrintCompleteEvent()
{
    // Here you can customize your code
}
}
```

SizeChanged

[SizeChanged](#) event is triggered when the accumulation chart is resized.

Arguments

The following fields are available in the [AccumulationResizeEventArgs](#).

- [Chart](#) – Specifies the current accumulation chart instance.
- [CurrentSize](#) – Specifies the current size of the accumulation chart.
- [PreviousSize](#) – Specifies the previous size of the accumulation chart.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfAccumulationChart Title="Mobile Browser Statistics">
  <AccumulationChartEvents
    SizeChanged="SizeChangeEvent"></AccumulationChartEvents>
  <AccumulationChartSeriesCollection>
    <AccumulationChartSeries DataSource="@StatisticsDetails" XName="Browser"
      YName="Users">
    </AccumulationChartSeries>
  </AccumulationChartSeriesCollection>
</SfAccumulationChart>
@code{
    public class Statistics
    {
        public string Browser { get; set; }
        public double Users { get; set; }
    }
    public List<Statistics> StatisticsDetails = new List<Statistics>
    {
        new Statistics { Browser = "Chrome", Users = 37 },
        new Statistics { Browser = "UC Browser", Users = 17 },
        new Statistics { Browser = "iPhone", Users = 19 },
        new Statistics { Browser = "Others", Users = 4 },
        new Statistics { Browser = "Opera", Users = 11 },
        new Statistics { Browser = "Android", Users = 12 },
    };
    public void SizeChangeEvent(AccumulationResizeEventArgs args)
    {
        // Here you can customize your code
    }
}
```

Loaded

Loaded event triggers after accumulation chart is loaded.

ASPX-CS

```

@using Syncfusion.Blazor.Charts
<SfAccumulationChart Title="Mobile Browser Statistics">
  <AccumulationChartEvents Loaded="@LoadHandler"></AccumulationChartEvents>
  <AccumulationChartSeriesCollection>
    <AccumulationChartSeries DataSource="@StatisticsDetails" XName="Browser"
      YName="Users">
    </AccumulationChartSeries>
  </AccumulationChartSeriesCollection>
</SfAccumulationChart>
@code{
public class Statistics
{
public string Browser { get; set; }
public double Users { get; set; }
}
public List<Statistics> StatisticsDetails = new List<Statistics>
{
new Statistics { Browser = "Chrome", Users = 37 },
new Statistics { Browser = "UC Browser", Users = 17 },
new Statistics { Browser = "iPhone", Users = 19 },
new Statistics { Browser = "Others", Users = 4 },
new Statistics { Browser = "Opera", Users = 11 },
new Statistics { Browser = "Android", Users = 12 },
};
public void LoadHandler(AccumulationLoadedEventArgs args)
{
// Here you can customize your code
}
}

```

OnPointClick

OnPointClick event triggers on point click.

Arguments

The following fields are available in the [AccumulationPointEventArgs](#).

- [PageX](#) – Specifies the current window page x location.
- [PageY](#) – Specifies the current window page y location.
- [Point](#) – Specifies the current point which is clicked.
- [PointIndex](#) – Specifies the index of the current point.
- [SeriesIndex](#) – Specifies the current series index.
- [X](#) – Specifies the x coordinate of the current mouse click.
- [Y](#) – Specifies the y coordinate of the current mouse click.

ASPX-CS

```

@using Syncfusion.Blazor.Charts
<SfAccumulationChart Title="Mobile Browser Statistics">

```

```

<AccumulationChartEvents
OnPointClick="PointClick"></AccumulationChartEvents>
<AccumulationChartSeriesCollection>
<AccumulationChartSeries DataSource="@StatisticsDetails" XName="Browser"
YName="Users">
</AccumulationChartSeries>
</AccumulationChartSeriesCollection>
</SfAccumulationChart>
@code{
public class Statistics
{
public string Browser { get; set; }
public double Users { get; set; }
}
public List<Statistics> StatisticsDetails = new List<Statistics>
{
new Statistics { Browser = "Chrome", Users = 37 },
new Statistics { Browser = "UC Browser", Users = 17 },
new Statistics { Browser = "iPhone", Users = 19 },
new Statistics { Browser = "Others", Users = 4 },
new Statistics { Browser = "Opera", Users = 11 },
new Statistics { Browser = "Android", Users = 12 },
};
public void PointClick(AccumulationPointEventArgs args)
{
// Here you can customize your code
}
}

```

TooltipRender

[TooltipRender](#) event triggers, before the tooltip for series is rendered.

Arguments

The following property is available in the [TooltipRenderEventArgs](#).

- [HeaderText](#) – Specifies the header text for the tooltip.
- [Text](#) – Specifies the text for the tooltip.

ASPX-CS

```

@using Syncfusion.Blazor.Charts
<SfAccumulationChart Title="Mobile Browser Statistics">
<AccumulationChartEvents
TooltipRender="TooltipRenderEvent"></AccumulationChartEvents>
<AccumulationChartSeriesCollection>
<AccumulationChartSeries DataSource="@StatisticsDetails" XName="Browser"
YName="Users">
</AccumulationChartSeries>
</AccumulationChartSeriesCollection>
<AccumulationChartTooltipSettings
Enable="true"></AccumulationChartTooltipSettings>
</SfAccumulationChart>
@code{
public class Statistics
{

```

```

public string Browser { get; set; }
public double Users { get; set; }
}
public List<Statistics> StatisticsDetails = new List<Statistics>
{
    new Statistics { Browser = "Chrome", Users = 37 },
    new Statistics { Browser = "UC Browser", Users = 17 },
    new Statistics { Browser = "iPhone", Users = 19 },
    new Statistics { Browser = "Others", Users = 4 },
    new Statistics { Browser = "Opera", Users = 11 },
    new Statistics { Browser = "Android", Users = 12 },
};
public void TooltipRenderEvent(TooltipRenderEventArgs args)
{
    // Here you can customize your code
}
}

```

How To

Text placing center of the Blazor Doughnut Chart Component

The annotation is used to place text, shapes or images in the center of the doughnut chart.

The [AccumulationChartAnnotation](#) property allows to add annotations to the chart. Specify the content that needs to be displayed in the accumulation chart area by using the [ContentTemplate](#) property of the annotation.

Step 1:

Render a doughnut chart with the required series using the [ChartSeriesCollection](#).

ASPX-CS

```

<AccumulationChartSeriesCollection>
<AccumulationChartSeries InnerRadius="60%"
Name="@nameof(MyDataModel.XValue)" DataSource="@chartData"
YName="@nameof(MyDataModel.YValue)" XName="@nameof(MyDataModel.XValue)">
</AccumulationChartSeries>
</AccumulationChartSeriesCollection>

```

Step 2:

Create a div element inside the [ContentTemplate](#) to display the text placing the centre of the doughnut.

ASPX-CS

```

<AccumulationChartAnnotation>
<ContentTemplate>
<div class="donut-text">Chart Annotation</div>
</ContentTemplate>
</AccumulationChartAnnotation>
</AccumulationChartAnnotations>

```

Step 3:

Since the text need to be placed in the center of the doughnut chart the [Region](#) property need to be set to **Regions.Chart**. Specify the [CoordinateUnits](#) in [Pixel](#) and set the X and Y coordinate values in percentage as shown in the following.

ASPX-CS

```
<AccumulationChartAnnotations>
<AccumulationChartAnnotation X="50%" Y="50%" CoordinateUnits="Units.Pixel"
Region="Regions.Chart">
<ContentTemplate>
<div class="donut-text">Chart Annotation</div>
</ContentTemplate>
</AccumulationChartAnnotation>
</AccumulationChartAnnotations>
```

The complete code snippet for the preceding steps is as follows.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfAccumulationChart>
<AccumulationChartLegendSettings
Visible="false"></AccumulationChartLegendSettings>
<AccumulationChartAnnotations>
<AccumulationChartAnnotation X="50%" Y="50%" CoordinateUnits="Units.Pixel"
Region="Regions.Chart">
<ContentTemplate>
<div class="donut-text">Chart Annotation</div>
</ContentTemplate>
</AccumulationChartAnnotation>
</AccumulationChartAnnotations>
<AccumulationChartSeriesCollection>
<AccumulationChartSeries InnerRadius="60%"
Name="@nameof(MyDataModel.XValue)" DataSource="@chartData"
YName="@nameof(MyDataModel.YValue)"
XName="@nameof(MyDataModel.XValue)"></AccumulationChartSeries>
</AccumulationChartSeriesCollection>
</SfAccumulationChart>
<style>
.donut-text {
align-content: center;
}
</style>
@code {
public class MyDataModel
{
public int XValue { get; set; }
public int YValue { get; set; }
}
private Random rnd = new Random();
public List<MyDataModel> chartData = new List<MyDataModel>();
protected override async Task OnInitializedAsync()
{
for (int i = 0; i < 4; i++)
{
chartData.Add(new MyDataModel() { XValue = i, YValue = rnd.Next(10, 100) });
}
}
```

```
}
}
```

Refer to the [Blazor Charts](#) feature tour page for its groundbreaking feature representations and also explore the [Blazor Accumulation Chart Example](#) to know about the various features of accumulation charts and how it is used to represent numeric proportional data.

AutoComplete

Getting Started with Blazor AutoComplete Component

This section briefly explains how to include a [AutoComplete](#) Component in your Blazor client-side application. You can refer to the [Getting Started with Syncfusion Blazor for Client-side in Visual Studio 2019](#) page for introduction and configure the common specifications.

To get start quickly with Blazor AutoComplete component, you can check on this video.

{% youtube

"youtube:https://www.youtube.com/watch?v=DbVs6UPjQ-U"%}

Importing Syncfusion Blazor component in the application

- Install [Syncfusion.Blazor.DropDowns](#) NuGet package to the application by using the **NuGet Package Manager**.
- You can add the client-side resources through [CDN](#) or from [NuGet](#) package in the **HEAD** element of the `~/Pages/_Host.cshtml` page.

HTML

```
<head>
<link href="_content/Syncfusion.Blazor.Themes/bootstrap4.css"
rel="stylesheet" />
@*<link
href="https://cdn.syncfusion.com/blazor/{{version}}/styles/{{theme}}.css"
rel="stylesheet" />*@
</head>
```

For Internet Explorer 11 kindly refer the polyfills. Refer the [documentation](#) for more information.

HTML

```
<head>
<link href="_content/Syncfusion.Blazor.Themes/bootstrap4.css"
rel="stylesheet" />
<script
src="https://github.com/Daddoon/Blazor.Polyfill/releases/download/3.0.1/blaz
or.polyfill.min.js"></script>
</head>
```

Adding component package to the application

Open `~/_Imports.razor` file and import the `Syncfusion.Blazor.DropDowns` package.

ASPX-CS

```
@using Syncfusion.Blazor.DropDowns
```

Add SyncfusionBlazor service in Startup.cs

Open the **Startup.cs** file and add services required by Syncfusion components using **services.AddSyncfusionBlazor()** method. Add this method in the **ConfigureServices** function as follows.

C#

```
using Syncfusion.Blazor;
namespace BlazorApplication
{
    public class Startup
    {
        ....
        ....
        public void ConfigureServices(IServiceCollection services)
        {
            ....
            ....
            services.AddSyncfusionBlazor();
        }
    }
}
```

To enable custom client side resource loading from CRG or CDN. You need to disable resource loading by **AddSyncfusionBlazor(true)** and load the scripts in the **HEAD** element of the **~/Pages/_Host.cshtml** page.

HTML

```
<head>
<script src="https://cdn.syncfusion.com/blazor/{{ site.blazorversion
}}/syncfusion-blazor.min.js"></script>
</head>
```

Adding AutoComplete component to the application

To initialize the AutoComplete component add the below code to your **Index.razor** view page which is present under **~/Pages** folder.

The following code shows a basic AutoComplete component.

ASPX-CS

```
<SfAutoComplete TValue="string" TItem="Countries" Placeholder="e.g.
Australia" DataSource="@LocalData">
<AutoCompleteFieldSettings Value="Name"></AutoCompleteFieldSettings>
</SfAutoComplete>
@code {
    public class Countries
    {
        public string Name { get; set; }
        public string Code { get; set; }
    }
    List<Countries> LocalData = new List<Countries> {
```



```
new Countries() { Name = "Australia", Code = "AU" },
new Countries() { Name = "Bermuda", Code = "BM" },
new Countries() { Name = "Canada", Code = "CA" },
new Countries() { Name = "Cameroon", Code = "CM" },
new Countries() { Name = "Denmark", Code = "DK" }
};
}
```

Run the application

After successful compilation of your application, press **F5** to run the application.

The output will be as follows.

e.g. Australia

Binding data source

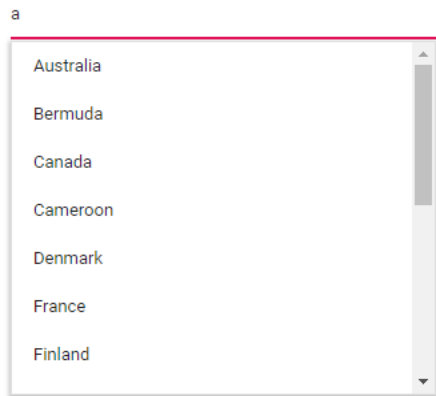
After initialization, populate the AutoComplete with data using the [DataSource](#) property. Here, an array of string values is passed to the [AutoComplete](#) component.

The following example illustrates the output in your browser.

ASPX-CS

```
<SfAutoComplete TValue="string" TItem="Countries" Placeholder="Select a
country" DataSource="@LocalData">
  <AutoCompleteFieldSettings Value="Name"></AutoCompleteFieldSettings>
</SfAutoComplete>
@code {
  public class Countries
  {
    public string Name { get; set; }
    public string Code { get; set; }
  }
  List<Countries> LocalData = new List<Countries> {
    new Countries() { Name = "Australia", Code = "AU" },
    new Countries() { Name = "Bermuda", Code = "BM" },
    new Countries() { Name = "Canada", Code = "CA" },
    new Countries() { Name = "Cameroon", Code = "CM" },
    new Countries() { Name = "Denmark", Code = "DK" },
    new Countries() { Name = "France", Code = "FR" },
    new Countries() { Name = "Finland", Code = "FI" },
    new Countries() { Name = "Germany", Code = "DE" },
    new Countries() { Name = "Greenland", Code = "GL" },
    new Countries() { Name = "Hong Kong", Code = "HK" },
    new Countries() { Name = "India", Code = "IN" },
    new Countries() { Name = "Italy", Code = "IT" },
    new Countries() { Name = "Japan", Code = "JP" },
    new Countries() { Name = "Mexico", Code = "MX" },
    new Countries() { Name = "Norway", Code = "NO" },
    new Countries() { Name = "Poland", Code = "PL" },
    new Countries() { Name = "Switzerland", Code = "CH" },
    new Countries() { Name = "United Kingdom", Code = "GB" },
    new Countries() { Name = "United States", Code = "US" },
  };
}
```

The output will be as follows.



Custom values

The AutoComplete allows the users to give input as custom value, which is not required to present in predefined set of values. By default, this support is enabled by the [AllowCustom](#) property. The custom value will be sent to post back handler when a form is about to be submitted.

ASPX-CS

```
<SfAutoComplete TValue="string" TItem="Countries" Placeholder="Select a
country" AllowCustom=true DataSource="@LocalData">
  <AutoCompleteFieldSettings Value="Name"></AutoCompleteFieldSettings>
</SfAutoComplete>
@code {
public class Countries
{
public string Name { get; set; }
public string Code { get; set; }
}
List<Countries> LocalData = new List<Countries> {
new Countries() { Name = "Australia", Code = "AU" },
new Countries() { Name = "Bermuda", Code = "BM" },
new Countries() { Name = "Canada", Code = "CA" },
new Countries() { Name = "Cameroon", Code = "CM" },
new Countries() { Name = "Denmark", Code = "DK" },
new Countries() { Name = "France", Code = "FR" },
new Countries() { Name = "Finland", Code = "FI" },
new Countries() { Name = "Germany", Code = "DE" },
new Countries() { Name = "Greenland", Code = "GL" },
new Countries() { Name = "Hong Kong", Code = "HK" },
new Countries() { Name = "India", Code = "IN" },
new Countries() { Name = "Italy", Code = "IT" },
new Countries() { Name = "Japan", Code = "JP" },
new Countries() { Name = "Mexico", Code = "MX" },
new Countries() { Name = "Norway", Code = "NO" },
new Countries() { Name = "Poland", Code = "PL" },
new Countries() { Name = "Switzerland", Code = "CH" },
new Countries() { Name = "United Kingdom", Code = "GB" },
new Countries() { Name = "United States", Code = "US" },
};
}
```

Configure the suggestion list

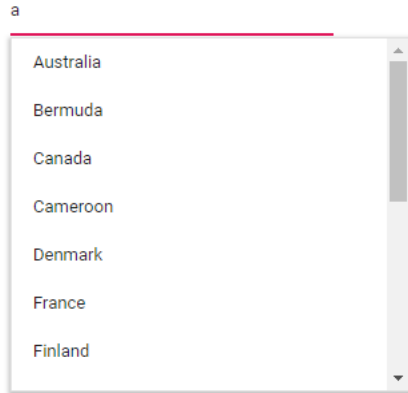
By default, suggestion list width automatically adjusts according to the AutoComplete input element's width, and the height of the suggestion list has 300px. The height and width of the popup list can also be customized using the [PopupHeight](#) and [PopupWidth](#) property respectively.

In the following sample, suggestion list's width and height are configured.

ASPX-CS

```
<SfAutoComplete TValue="string" TItem="Countries" Placeholder="Select a
country" DataSource="@LocalData" PopupHeight="300px" PopupWidth="300px">
<AutoCompleteFieldSettings Value="Name"></AutoCompleteFieldSettings>
</SfAutoComplete>
@code {
public class Countries
{
public string Name { get; set; }
public string Code { get; set; }
}
List<Countries> LocalData = new List<Countries> {
new Countries() { Name = "Australia", Code = "AU" },
new Countries() { Name = "Bermuda", Code = "BM" },
new Countries() { Name = "Canada", Code = "CA" },
new Countries() { Name = "Cameroon", Code = "CM" },
new Countries() { Name = "Denmark", Code = "DK" },
new Countries() { Name = "France", Code = "FR" },
new Countries() { Name = "Finland", Code = "FI" },
new Countries() { Name = "Germany", Code = "DE" },
new Countries() { Name = "Greenland", Code = "GL" },
new Countries() { Name = "Hong Kong", Code = "HK" },
new Countries() { Name = "India", Code = "IN" },
new Countries() { Name = "Italy", Code = "IT" },
new Countries() { Name = "Japan", Code = "JP" },
new Countries() { Name = "Mexico", Code = "MX" },
new Countries() { Name = "Norway", Code = "NO" },
new Countries() { Name = "Poland", Code = "PL" },
new Countries() { Name = "Switzerland", Code = "CH" },
new Countries() { Name = "United Kingdom", Code = "GB" },
new Countries() { Name = "United States", Code = "US" },
};
}
```

The output will be as follows.



See Also

- [Getting Started with Syncfusion Blazor for Client-Side in .NET Core CLI](#)
- [Getting Started with Syncfusion Blazor for Server-side in Visual Studio](#)
- [Getting Started with Syncfusion Blazor for Server-Side in .NET Core CLI](#)

Data Binding in Blazor AutoComplete Component

Data binding can be achieved by using the **bind-Value** attribute and it supports string, int, Enum, DateTime, and bool types. If component value has been changed, it will affect all places where you bind the variable for the **bind-value** attribute.

ASPX-CS

```
@using Syncfusion.Blazor.DropDowns
<p>AutoComplete value is: @AutoVal</p>
<SfAutoComplete TValue="string" TItem="Countries" Placeholder="e.g.
Australia" @bind-Value="@AutoVal" DataSource="@Country">
  <AutoCompleteFieldSettings Value="Name"></AutoCompleteFieldSettings>
</SfAutoComplete>
@code {
public string AutoVal;
public class Countries
{
public string Name { get; set; }
public string Code { get; set; }
}
List<Countries> Country = new List<Countries>
{
new Countries() { Name = "Australia", Code = "AU" },
new Countries() { Name = "Bermuda", Code = "BM" },
new Countries() { Name = "Canada", Code = "CA" },
new Countries() { Name = "Cameroon", Code = "CM" },
};
}
```

Data Source in Blazor AutoComplete 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 kinds of data services such as OData, OData V4, and Web API and data formats such as XML, JSON, JSONP with the help of DataManager Adaptors.

Fields	Type	Description
Value	int 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 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.

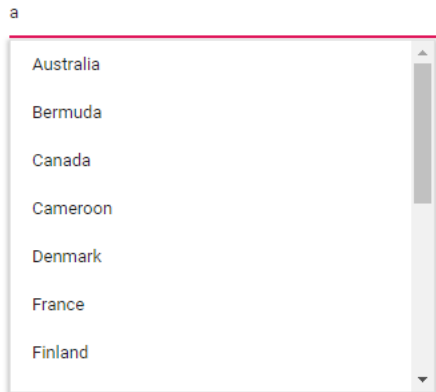
In the following example, **Name** column from complex data have been mapped to the **Value** field.

ASPX-CS

```
@using Syncfusion.Blazor.DropDowns
<SfAutoComplete TValue="string" TItem="Countries" Placeholder="e.g.
Australia" DataSource="@Country">
<AutoCompleteFieldSettings Value="Name"></AutoCompleteFieldSettings>
</SfAutoComplete>
@code {
public class Countries
{
public string Name { get; set; }
public string Code { get; set; }
}
List<Countries> Country = new List<Countries>
{
new Countries() { Name = "Australia", Code = "AU" },
new Countries() { Name = "Bermuda", Code = "BM" },
new Countries() { Name = "Canada", Code = "CA" },
new Countries() { Name = "Cameroon", Code = "CM" },
new Countries() { Name = "Denmark", Code = "DK" },
new Countries() { Name = "France", Code = "FR" },
new Countries() { Name = "Finland", Code = "FI" },
new Countries() { Name = "Germany", Code = "DE" },
new Countries() { Name = "Greenland", Code = "GL" },
new Countries() { Name = "Hong Kong", Code = "HK" },
new Countries() { Name = "India", Code = "IN" },
new Countries() { Name = "Italy", Code = "IT" },
new Countries() { Name = "Japan", Code = "JP" },
new Countries() { Name = "Mexico", Code = "MX" },
new Countries() { Name = "Norway", Code = "NO" },
new Countries() { Name = "Poland", Code = "PL" },
}
```

```
new Countries() { Name = "Switzerland", Code = "CH" },
new Countries() { Name = "United Kingdom", Code = "GB" },
new Countries() { Name = "United States", Code = "US" },
};
}
```

The output will be as follows.



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.CountryID` column from complex data have been mapped to the `Value` field.

ASPX-CS

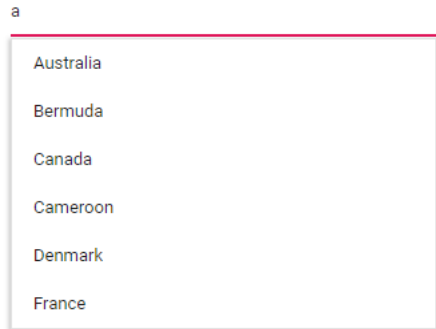
```
@using Syncfusion.Blazor.DropDowns
<SfAutoComplete TValue="string" TItem="Complex" Placeholder="e.g. Select a
country" DataSource="@LocalData">
<AutoCompleteFieldSettings Value="Country.CountryID"
></AutoCompleteFieldSettings>
</SfAutoComplete>
@code {
public IEnumerable<Complex> LocalData { get; set; } = new
Complex().GetData();
public class Code
{
public string ID { get; set; }
}
public class Country
{
public string CountryID { get; set; }
}
public class Complex
{
public Country Country { get; set; }
public Code Code { get; set; }
public List<Complex> GetData()
{
List<Complex> Data = new List<Complex>();
```

```

Data.Add(new Complex() { Country = new Country() { CountryID = "Australia"
}, Code = new Code() { ID = "AU" } });
Data.Add(new Complex() { Country = new Country() { CountryID = "Bermuda" },
Code = new Code() { ID = "BM" } });
Data.Add(new Complex() { Country = new Country() { CountryID = "Canada" },
Code = new Code() { ID = "CA" } });
Data.Add(new Complex() { Country = new Country() { CountryID = "Cameroon" },
Code = new Code() { ID = "CM" } });
Data.Add(new Complex() { Country = new Country() { CountryID = "Denmark" },
Code = new Code() { ID = "DK" } });
Data.Add(new Complex() { Country = new Country() { CountryID = "France" },
Code = new Code() { ID = "FR" } });
return Data;
}
}
}

```

The output will be as follows.



Bind to remote data

The AutoComplete supports retrieval of data from remote data services with the help of [DataManager](#) 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.

ASPX-CS

```

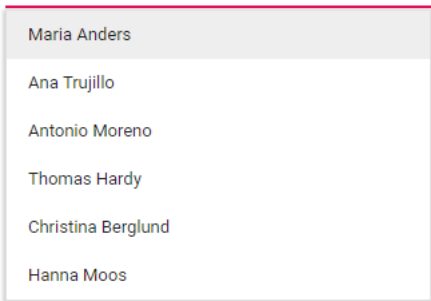
@using Syncfusion.Blazor
@using Syncfusion.Blazor.Data
@using Syncfusion.Blazor.DropDowns
<SfAutoComplete TValue="string" TItem="OrderDetails" Placeholder="Select a
name" Query="@RemoteDataQuery" Autofill="true">
<SfDataManager
Url="https://js.syncfusion.com/demos/ejServices/Wcf/Northwind.svc/Orders"
CrossDomain="true"
Adaptor="Syncfusion.Blazor.Adaptors.ODataAdaptor"></SfDataManager>
<AutoCompleteFieldSettings Value="CustomerID"></AutoCompleteFieldSettings>
</SfAutoComplete>
@code{
public Query RemoteDataQuery = new Query().Select(new List<string> {
"CustomerID" }).Take(6).RequiresCount();
public Syncfusion.Blazor.Lists.SortOrder Sort { get; set; } =
Syncfusion.Blazor.Lists.SortOrder.Ascending;
}

```

```
public class OrderDetails
{
    public int? OrderID { get; set; }
    public string CustomerID { get; set; }
    public int? EmployeeID { get; set; }
    public double? Freight { get; set; }
    public string ShipCity { get; set; }
    public bool Verified { get; set; }
    public DateTime? OrderDate { get; set; }
    public string ShipName { get; set; }
    public string ShipCountry { get; set; }
    public DateTime? ShippedDate { get; set; }
    public string ShipAddress { get; set; }
}
}
```

The output will be as follows.

a



Web API Adaptor

Use the **WebApiAdaptor** to bind autocomplete with Web API created using OData.

ASPX-CS

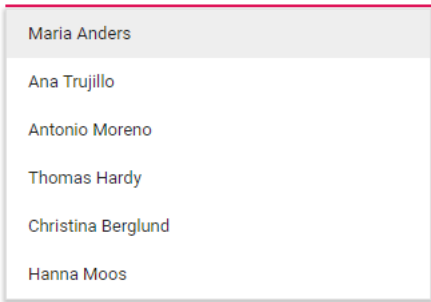
```
@using Syncfusion.Blazor.Data
@using Syncfusion.Blazor.DropDowns
<SfAutoComplete TValue="string" TItem="OrderDetails" Placeholder="Select a name" Query="@Query">
    <SfDataManager Url="https://ej2services.syncfusion.com/production/web-services/api/Orders" Adaptor="Syncfusion.Blazor.Adaptors.WebApiAdaptor" CrossDomain=true></SfDataManager>
    <AutoCompleteFieldSettings Value="CustomerID"></AutoCompleteFieldSettings>
</SfAutoComplete>
@code {
    public Query Query = new Query().Select(new List<string> { "CustomerID" }).Take(6).RequiresCount();
    public class OrderDetails
    {
        public int? OrderID { get; set; }
        public string CustomerID { get; set; }
        public int? EmployeeID { get; set; }
        public double? Freight { get; set; }
        public string ShipCity { get; set; }
        public bool Verified { get; set; }
        public DateTime? OrderDate { get; set; }
    }
}
```



```
public string ShipName { get; set; }
public string ShipCountry { get; set; }
public DateTime? ShippedDate { get; set; }
public string ShipAddress { get; set; }
}
```

The output will be as follows.

a



Custom Adaptor

The [SfDataManager](#) has custom adaptor support which allows you to perform manual operations on the data. This can be utilized for implementing custom data binding and editing operations in the AutoComplete component.

For implementing custom data binding in AutoComplete, the **DataAdaptor** class is used. This abstract class acts as a base class for the custom adaptor.

The **DataAdaptor** abstract class has both synchronous and asynchronous method signatures which can be overridden in the custom adaptor. Following are the method signatures present in this class,

C#

```
public abstract class DataAdaptor
{
    /// <summary>
    /// Performs data Read operation synchronously.
    /// </summary>
    public virtual object Read(DataManagerRequest dataManagerRequest, string key
    = null)
    {
        /// <summary>
        /// Performs data Read operation asynchronously.
        /// </summary>
        public virtual Task<object> ReadAsync(DataManagerRequest dataManagerRequest,
        string key = null)
    }
}
```

The custom data binding can be performed in the AutoComplete component by providing the custom adaptor class and overriding the Read or ReadAsync method of the DataAdaptor abstract class.

The following sample code demonstrates implementing custom data binding using custom adaptor,

ASPX-CS

```
<SfAutoComplete TValue="string" TItem="Orders">
```

```
<SfDataManager AdaptorInstance="@typeof(CustomAdaptor)"
Adaptor="Adaptors.CustomAdaptor"></SfDataManager>
<AutoCompleteFieldSettings Value="CustomerID"></AutoCompleteFieldSettings>
</SfAutoComplete>
@code{
public class Orders
{
public Orders() { }
public Orders(int OrderID, string CustomerID)
{
this.OrderID = OrderID;
this.CustomerID = CustomerID;
}
public int OrderID { get; set; }
public string CustomerID { get; set; }
}
public class CustomAdaptor : DataAdaptor
{
static readonly HttpClient client = new HttpClient();
public static List<OrdersDetails> order = OrdersDetails.GetAllRecords();
public override object Read(DataManagerRequest dm, string key = null)
{
IEnumerable<OrdersDetails> DataSource = order;
if (dm.Search != null && dm.Search.Count > 0)
{
DataSource = DataOperations.PerformSearching(DataSource, dm.Search);
//Search
}
if (dm.Sorted != null && dm.Sorted.Count > 0) //Sorting
{
DataSource = DataOperations.PerformSorting(DataSource, dm.Sorted);
}
if (dm.Where != null && dm.Where.Count > 0) //Filtering
{
DataSource = DataOperations.PerformFiltering(DataSource, dm.Where,
dm.Where[0].Operator);
}
int count = DataSource.Cast<OrdersDetails>().Count();
if (dm.Skip != 0)
{
DataSource = DataOperations.PerformSkip(DataSource, dm.Skip);
//Paging
}
if (dm.Take != 0)
{
DataSource = DataOperations.PerformTake(DataSource, dm.Take);
}
return dm.RequiresCounts ? new DataResult() { Result = DataSource, Count =
count } : (object)DataSource;
}
}
}
```

Offline mode

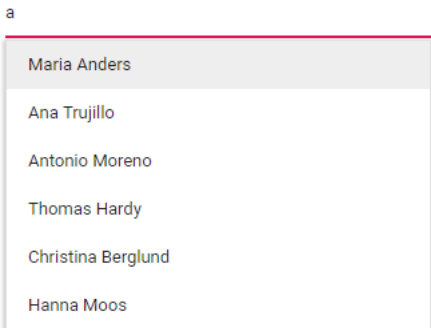
To avoid post back for every action, set the autocomplete to load all data on initialization and make the actions process in client-side. To enable this behaviour, use the `Offline` property of `DataManager`.

Refer to the following example for remote data binding and enabled offline mode.

ASPX-CS

```
@using Syncfusion.Blazor.Data
@using Syncfusion.Blazor.DropDowns
<SfAutoComplete TValue="string" TItem="EmployeeData" Placeholder="Select a
Employee" Query="@Query">
<SfDataManager Url="https://ej2services.syncfusion.com/production/web-
services/api/Employees" Offline=true Adaptor="Adaptors.WebApiAdaptor"
CrossDomain=true></SfDataManager>
<AutoCompleteFieldSettings Value="FirstName"></AutoCompleteFieldSettings>
</SfAutoComplete>
@code{
public Query Query = new Query().Select(new List<string> { "FirstName"
}).Take(6).RequiresCount();
public class EmployeeData
{
public int EmployeeID { get; set; }
public string FirstName { get; set; }
public string Designation { get; set; }
public string Country { get; set; }
}
}
```

The output will be as follows.



ValueTuple data binding

You can bind [ValueTuple](#) data to AutoComplete component. The following code helps you get a string value from the enumeration data by using [ValueTuple](#).

CSHARP

```
@using Syncfusion.Blazor.DropDowns;
<SfAutoComplete TItem="(DayOfWeek, string)" Width="250px" TValue="DayOfWeek"
DataSource="@ (Enum.GetValues<DayOfWeek>().Select(e => (e, e.ToString())))">
<AutoCompleteFieldSettings Value="Item1" />
</SfAutoComplete>
```

The output will shown as follows,

Sunday

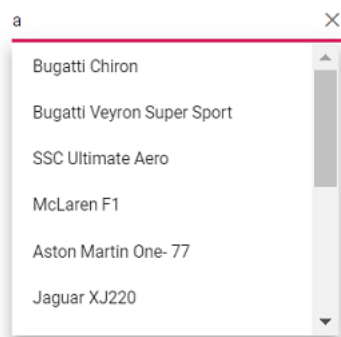
Binding ExpandoObject

You can bind [ExpandoObject](#) data to the AutoComplete component. The following example [ExpandoObject](#) is bound to the collection of vehicles data.

CSHARP

```
@using Syncfusion.Blazor.DropDowns
@using System.Dynamic
<SfAutoComplete TItem="ExpandoObject" TValue="string" PopupHeight="230px"
Placeholder="Select a vehicle" DataSource="@VehicleData">
<AutoCompleteFieldSettings Value="Text"></AutoCompleteFieldSettings>
</SfAutoComplete>
@code{
public List<ExpandoObject> VehicleData { get; set; } = new
List<ExpandoObject>();
protected override void OnInitialized()
{
VehicleData = Enumerable.Range(1, 15).Select((x) =>
{
dynamic d = new ExpandoObject();
d.ID = (1000 + x).ToString();
d.Text = (new string[] { "Hennessey Venom", "Bugatti Chiron", "Bugatti
Veyron Super Sport", "SSC Ultimate Aero", "Koenigsegg CCR", "McLaren F1",
"Aston Martin One- 77", "Jaguar XJ220", "McLaren P1", "Ferrari LaFerrari",
"Mahindra Jaguar", "Hyundai Toyota", "Jeep Volkswagen", "Tata Maruti
Suzuki", "Audi Mercedes Benz" }[x - 1]);
return d;
}).Cast<ExpandoObject>().ToList<ExpandoObject>();
}
}
```

The output will shown as follows,



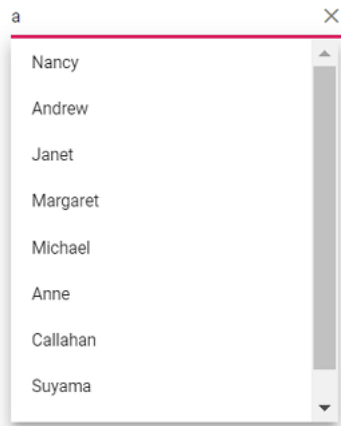
Binding DynamicObject

You can bind [DynamicObject](#) data to the AutoComplete component. The following example `DynamicObject` is bound to the collection of customers data.

CSHARP

```
@using Syncfusion.Blazor.DropDowns
@using System.Dynamic
<SfAutoComplete TValue="string" TItem="DynamicDictionary"
Placeholder="Select a name" DataSource="@Orders">
<AutoCompleteFieldSettings Text="CustomerName"
Value="CustomerName"></AutoCompleteFieldSettings>
</SfAutoComplete>
@code{
    public List<DynamicDictionary> Orders = new List<DynamicDictionary>() { };
    protected override void OnInitialized()
    {
        Orders = Enumerable.Range(1, 15).Select((x) =>
        {
            dynamic d = new DynamicDictionary();
            d.OrderID = 1000 + x;
            d.CustomerName = (new string[] { "Nancy", "Andrew", "Janet", "Margaret",
            "Steven", "Michael", "Robert", "Anne", "Nige", "Fuller", "Dodsworth",
            "Leverling", "Callahan", "Suyama", "Davolio" }[x - 1]);
            return d;
        }).Cast<DynamicDictionary>().ToList<DynamicDictionary>();
    }
    public class DynamicDictionary : System.Dynamic.DynamicObject
    {
        Dictionary<string, object> dictionary = new Dictionary<string, object>();
        public override bool TryGetMember(GetMemberBinder binder, out object result)
        {
            string name = binder.Name;
            return dictionary.TryGetValue(name, out result);
        }
        public override bool TrySetMember(SetMemberBinder binder, object value)
        {
            dictionary[binder.Name] = value;
            return true;
        }
        //The GetDynamicMemberNames method of DynamicObject class must be overridden
        and return the property names to perform data operation and editing while
        using DynamicObject.
        public override System.Collections.Generic.IEnumerable<string>
        GetDynamicMemberNames()
        {
            return this.dictionary?.Keys;
        }
    }
}
```

The output will shown as follows,



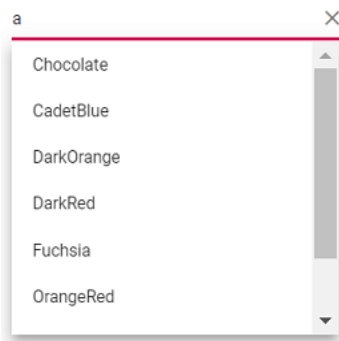
Binding ObservableCollection

You can bind [ObservableCollection](#) data to the AutoComplete component. The following example **Observable Data** is bound to a collection of colors data.

CSHARP

```
@using Syncfusion.Blazor.DropDowns
@using System.Collections.ObjectModel;
<SfAutoComplete TValue="string" TItem="Colors" PopupHeight="230px"
Placeholder="Select a color" DataSource="@ColorsData">
<AutoCompleteFieldSettings Value="Color"></AutoCompleteFieldSettings>
</SfAutoComplete>
@code {
    public class Colors
    {
        public string Code { get; set; }
        public string Color { get; set; }
    }
    private ObservableCollection<Colors> ColorsData = new
    ObservableCollection<Colors>()
    {
        new Colors() { Color = "Chocolate", Code = "#75523C" },
        new Colors() { Color = "CadetBlue", Code = "#3B8289" },
        new Colors() { Color = "DarkOrange", Code = "#FF843D" },
        new Colors() { Color = "DarkRed", Code = "#CA3832" },
        new Colors() { Color = "Fuchsia", Code = "#D44FA3" },
        new Colors() { Color = "HotPink", Code = "#F23F82" },
        new Colors() { Color = "Indigo", Code = "#2F5D81" },
        new Colors() { Color = "LimeGreen", Code = "#4CD242" },
        new Colors() { Color = "OrangeRed", Code = "#FE2A00" },
        new Colors() { Color = "Tomato", Code = "#FF745C" },
        new Colors() { Color = "Brown", Code = "#A52A2A" },
        new Colors() { Color = "Maroon", Code = "#800000" },
        new Colors() { Color = "Green", Code = "#008000" },
        new Colors() { Color = "Pink", Code = "#FFC0CB" },
        new Colors() { Color = "Purple", Code = "#800080" }
    };
}
```

The output will shown as follows,



Entity Framework

You need to follow the below steps to consume data from the **Entity Framework** in the AutoComplete component.

Create DbContext class

The first step is to create a DbContext class called **OrderContext** to connect to a Microsoft SQL Server database.

C#

```
using Microsoft.EntityFrameworkCore;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using EFDropDown.Shared.Models;
namespace EFDropDown.Shared.DataAccess
{
    public class OrderContext : DbContext
    {
        public virtual DbSet<Shared.Models.Order> Orders { get; set; }
        protected override void OnConfiguring(DbContextOptionsBuilder
optionsBuilder)
        {
            if (!optionsBuilder.IsConfigured)
            {
                optionsBuilder.UseSqlServer(@"Data
Source=(LocalDB)\MSSQLLocalDB;AttachDbFilename=D:\Blazor\DropDownList\EFDrop
Down\Shared\App_Data\NORTHWND.MDF;Integrated Security=True;Connect
Timeout=30");
            }
        }
    }
}
```

Create data access layer to perform data operation

Now you need to create a class named **OrderDataAccessLayer**, which act as data access layer for retrieving the records from the database table.

C#

```
using Microsoft.EntityFrameworkCore;
using System;
using System.Collections.Generic;
```

```
using System.Linq;
using System.Threading.Tasks;
using EFDropDown.Shared.Models;
namespace EFDropDown.Shared.DataAccess
{
    public class OrderDataAccessLayer
    {
        OrderContext db = new OrderContext();
        //To Get all Orders details
        public DbSet<Order> GetAllOrders()
        {
            try
            {
                return db.Orders;
            }
            catch
            {
                throw;
            }
        }
    }
}
```

Creating Web API Controller

A Web API Controller has to be created which allows AutoComplete directly to consume data from the Entity framework.

C#

```
using EFDropDown.Shared.DataAccess;
using EFDropDown.Shared.Models;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Primitives;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using System.Web;
using Microsoft.AspNetCore.Http;
namespace EFDropDown.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    //TreeGrid
    public class DefaultController : ControllerBase
    {
        OrderDataAccessLayer db = new OrderDataAccessLayer();
        [HttpGet]
        public object Get()
        {
            IQueryable<Order> data = db.GetAllOrders().AsQueryable();
            var count = data.Count();
            var queryString = Request.Query;
            if (queryString.Keys.Contains("$inlinecount"))
            {
                StringValues Skip;
            }
        }
    }
}
```



```
StringValues Take;
int skip = (queryString.TryGetValue("$skip", out Skip)) ?
Convert.ToInt32(Skip[0]) : 0;
int top = (queryString.TryGetValue("$top", out Take)) ?
Convert.ToInt32(Take[0]) : data.Count();
return new { Items = data.Skip(skip).Take(top), Count = count };
}
else
{
return data;
}
}
}
```

Configure AutoComplete component using Web API adaptor

Now you can configure the AutoComplete using the '**SfDataManager**' to interact with the created Web API and consume the data appropriately. To interact with web api, you need to use WebApiAdaptor.

ASPX-CS

```
@using Syncfusion.Blazor.Data
@using Syncfusion.Blazor.DropDowns
<SfAutoComplete TValue="string" TItem="Order" Placeholder="Select a
Country">
<SfDataManager Url="api/Default" Adaptor="Adaptors.WebApiAdaptor"
CrossDomain="true"></SfDataManager>
<AutoCompleteFieldSettings Value="ShipCountry"></AutoCompleteFieldSettings>
</SfAutoComplete>
@code{
public class Order
{
public string ShipCountry { get; set; }
}
}
```

Grouping in Blazor AutoComplete 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.

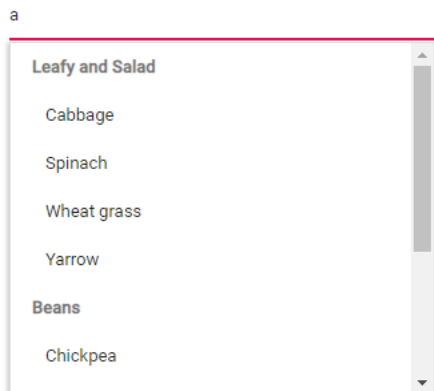
In the following sample, vegetables are grouped according on its category using **GroupBy** field.

ASPX-CS

```
@using Syncfusion.Blazor.DropDowns
<SfAutoComplete TValue="string" TItem="Vegetables" Placeholder="e.g. Select
a vegetable" DataSource="@LocalData">
<AutoCompleteFieldSettings GroupBy="Category"
Value="Vegetable"></AutoCompleteFieldSettings>
</SfAutoComplete>
@code {
public IEnumerable<Vegetables> LocalData { get; set; } = new
Vegetables().VegetablesList();
}
```

```
public class Vegetables
{
    public string Vegetable { get; set; }
    public string Category { get; set; }
    public string ID { get; set; }
    public List<Vegetables> VegetablesList()
    {
        List<Vegetables> Veg = new List<Vegetables>();
        Veg.Add(new Vegetables { Vegetable = "Cabbage", Category = "Leafy and Salad", ID = "item1" });
        Veg.Add(new Vegetables { Vegetable = "Chickpea", Category = "Beans", ID = "item2" });
        Veg.Add(new Vegetables { Vegetable = "Garlic", Category = "Bulb and Stem", ID = "item3" });
        Veg.Add(new Vegetables { Vegetable = "Green bean", Category = "Beans", ID = "item4" });
        Veg.Add(new Vegetables { Vegetable = "Horse gram", Category = "Beans", ID = "item5" });
        Veg.Add(new Vegetables { Vegetable = "Nopal", Category = "Bulb and Stem", ID = "item6" });
        Veg.Add(new Vegetables { Vegetable = "Onion", Category = "Bulb and Stem", ID = "item7" });
        Veg.Add(new Vegetables { Vegetable = "Pumpkins", Category = "Leafy and Salad", ID = "item8" });
        Veg.Add(new Vegetables { Vegetable = "Spinach", Category = "Leafy and Salad", ID = "item9" });
        Veg.Add(new Vegetables { Vegetable = "Wheat grass", Category = "Leafy and Salad", ID = "item10" });
        Veg.Add(new Vegetables { Vegetable = "Yarrow", Category = "Leafy and Salad", ID = "item11" });
        return Veg;
    }
}
```

The output will be as follows.



Filtering in Blazor AutoComplete 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:

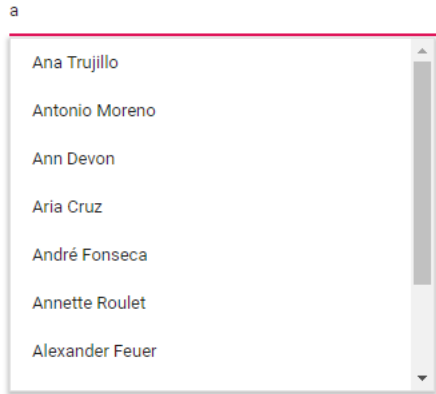
Filter Type	Description	Supported Types
-----	-----	-----
StartsWith	Checks whether a value begins with the specified value.	String
EndsWith	Checks whether a value ends with specified value.	String
Contains	Checks whether a value contains with specified value.	String

The following examples shows data filtering is done with the `StartsWith` type.

ASPX-CS

```
@using Syncfusion.Blazor.Data
@using Syncfusion.Blazor.DropDowns
<SfAutoComplete TValue="string" TItem="OrderDetails" Placeholder="Select a
customerID" Query="@RemoteDataQuery"
FilterType="Syncfusion.Blazor.DropDowns.FilterType.StartsWith">
<SfDataManager
Url="https://js.syncfusion.com/demos/ejServices/Wcf/Northwind.svc/Orders"
Adaptor="Adaptors.ODataAdaptor" CrossDomain=true></SfDataManager>
<AutoCompleteFieldSettings Value="CustomerID"></AutoCompleteFieldSettings>
</SfAutoComplete>
@code {
public Query RemoteDataQuery = new Query().Select(new List<string> {
"CustomerID" }).Take(6).RequiresCount();
public class OrderDetails
{
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public int? EmployeeID { get; set; }
public double? Freight { get; set; }
public string ShipCity { get; set; }
public bool Verified { get; set; }
public DateTime? OrderDate { get; set; }
public string ShipName { get; set; }
public string ShipCountry { get; set; }
public DateTime? ShippedDate { get; set; }
public string ShipAddress { get; set; }
}
}
```

The output will be as follows.



Filter item count

You can specify the filter suggestion item count using the [SuggestionCount](#) property of AutoComplete.

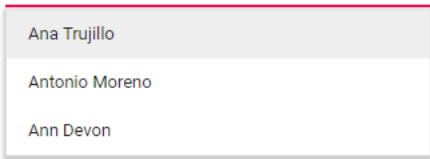
Refer to the following example to restrict the suggestion list item counts as 3.

ASPX-CS

```
@using Syncfusion.Blazor.Data
<SfAutoComplete TValue="string" TItem="OrderDetails" Placeholder="Select a
customerID" SuggestionCount=3 Query="@RemoteDataQuery"
FilterType="Syncfusion.Blazor.DropDowns.FilterType.StartsWith">
<SfDataManager
Url="https://js.syncfusion.com/demos/ejServices/Wcf/Northwind.svc/Orders"
Adaptor="Adaptors.ODataAdaptor" CrossDomain=true></SfDataManager>
<AutoCompleteFieldSettings Value="CustomerID"></AutoCompleteFieldSettings>
</SfAutoComplete>
@code {
public Query RemoteDataQuery = new Query().Select(new List<string> {
"CustomerID" }).RequiresCount();
public class OrderDetails
{
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public int? EmployeeID { get; set; }
public double? Freight { get; set; }
public string ShipCity { get; set; }
public bool Verified { get; set; }
public DateTime? OrderDate { get; set; }
public string ShipName { get; set; }
public string ShipCountry { get; set; }
public DateTime? ShippedDate { get; set; }
public string ShipAddress { get; set; }
}
}
```

The output will be as follows.

a



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 setting 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.

ASPX-CS

```
@using Syncfusion.Blazor.Data
@using Syncfusion.Blazor.DropDowns
<SfAutoComplete TValue="string" TItem="OrderDetails" Placeholder="Select a
customerID" MinLength=3 Query="@RemoteDataQuery"
FilterType="Syncfusion.Blazor.DropDowns.FilterType.StartsWith">
<SfDataManager
Url="https://js.syncfusion.com/demos/ejServices/Wcf/Northwind.svc/Orders"
Adaptor="Adaptors.ODataAdaptor" CrossDomain=true></SfDataManager>
<AutoCompleteFieldSettings Value="CustomerID"></AutoCompleteFieldSettings>
</SfAutoComplete>
@code {
public Query RemoteDataQuery = new Query().Select(new List<string> {
"CustomerID" }).RequiresCount();
public class OrderDetails
{
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public int? EmployeeID { get; set; }
public double? Freight { get; set; }
public string ShipCity { get; set; }
public bool Verified { get; set; }
public DateTime? OrderDate { get; set; }
public string ShipName { get; set; }
public string ShipCountry { get; set; }
public DateTime? ShippedDate { get; set; }
public string ShipAddress { get; set; }
}
}
```

The output will be as follows.

ann



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.

ASPX-CS

```
@using Syncfusion.Blazor.Data
@using Syncfusion.Blazor.DropDowns
<SfAutoComplete TValue="string" TItem="Countries" Placeholder="Select a
country" IgnoreCase=false DataSource="@LocalData">
<AutoCompleteFieldSettings Value="Name"></AutoCompleteFieldSettings>
</SfAutoComplete>
@code {
public class Countries
{
public string Name { get; set; }
public string Code { get; set; }
}
List<Countries> LocalData = new List<Countries> {
new Countries() { Name = "Australia", Code = "AU" },
new Countries() { Name = "Bermuda", Code = "BM" },
new Countries() { Name = "Canada", Code = "CA" },
new Countries() { Name = "Cameroon", Code = "CM" },
new Countries() { Name = "Denmark", Code = "DK" },
new Countries() { Name = "France", Code = "FR" },
new Countries() { Name = "Finland", Code = "FI" },
new Countries() { Name = "Germany", Code = "DE" },
new Countries() { Name = "Greenland", Code = "GL" },
new Countries() { Name = "Hong Kong", Code = "HK" },
};
}
```

Custom Filtering

The AutoComplete component filter queries can be customized. You can also use your own filter libraries to filter data like Fuzzy search.

ASPX-CS

```
@using Syncfusion.Blazor.Data
<SfAutoComplete TValue="string" @ref="autoObj" TItem="Countries"
Placeholder="e.g. Australia" AllowFiltering="true">
<AutoCompleteFieldSettings Text="Name"
Value="Code"></AutoCompleteFieldSettings>
<AutoCompleteEvents TValue="string" TItem="Countries"
Filtering="OnFilter"></AutoCompleteEvents>
</SfAutoComplete>
@code {
SfAutoComplete<string, Countries> autoObj { get; set; }
public class Countries
{
public string Name { get; set; }
public string Code { get; set; }
}
List<Countries> Country = new List<Countries>
```

```

{
new Countries() { Name = "Australia", Code = "AU" },
new Countries() { Name = "Bermuda", Code = "BM" },
new Countries() { Name = "Canada", Code = "CA" },
new Countries() { Name = "Cameroon", Code = "CM" },
new Countries() { Name = "Denmark", Code = "DK" }
};
private async Task OnFilter(FilteringEventArgs args)
{
args.PreventDefaultAction = true;
var query = new Query().Where(new WhereFilter() { Field = "Name", Operator =
"contains", value = args.Text, IgnoreCase = true });
query = !string.IsNullOrEmpty(args.Text) ? query : new Query();
await autoObj.FilterAsync(Country, query);
}
}

```

Templates in Blazor AutoComplete 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.

ASPX-CS

```

@using Syncfusion.Blazor.Data
@using Syncfusion.Blazor.DropDowns
<SfAutoComplete TValue="string" TItem="EmployeeData" Placeholder="Select a
customer" DataSource="@Data">
<AutoCompleteTemplates TItem="EmployeeData">
<ItemTemplate>
<span><span class='name'>@((context as EmployeeData).FirstName)</span><span
class='country'>@((context as EmployeeData).Country)</span></span>
</ItemTemplate>
</AutoCompleteTemplates>
<AutoCompleteFieldSettings Value="FirstName"></AutoCompleteFieldSettings>
</SfAutoComplete>
@code {
public class EmployeeData
{
public string FirstName { get; set; }
public string Country { get; set; }
}
List<EmployeeData> Data = new List<EmployeeData>
{
new EmployeeData() { FirstName = "Andrew Fuller", Country = "England" },
new EmployeeData() { FirstName = "Anne Dodsworth", Country = "USA" },
new EmployeeData() { FirstName = "Janet Leverling", Country = "USA" },
new EmployeeData() { FirstName = "Laura Callahan", Country = "USA"},
new EmployeeData() { FirstName = "Margaret Peacock", Country = "USA"},
new EmployeeData() { FirstName = "Michael Suyama", Country = "USA", },
new EmployeeData() { FirstName = "Nancy Davolio", Country = "USA"},

```

```

new EmployeeData() { FirstName = "Robert King", Country = "England"},
new EmployeeData() { FirstName = "Steven Buchanan", Country = "England"},
};
}
<style>
.country {
right: 15px;
position: absolute;
}
</style>

```

The output will be as follows.

a

Andrew Fuller	England
Anne Dodsworth	USA
Janet Leverling	USA
Laura Callahan	USA
Margaret Peacock	USA
Michael Suyama	USA

Group template

The group header title under which appropriate sub-items are categorized can also be customized with the help of [GroupTemplate](#) property. This template is common for both inline and floating group header templates.

In the following sample, employees are grouped according to their country.

ASPX-CS

```

@using Syncfusion.Blazor.Data
@using Syncfusion.Blazor.DropDowns
<SfAutoComplete TValue="string" TItem="EmployeeData" Placeholder="Select a customer" DataSource="@Data">
<AutoCompleteTemplates TItem="EmployeeData">
<GroupTemplate>
<span class='country'>@(context.Text)</span>
</GroupTemplate>
</AutoCompleteTemplates>
<AutoCompleteFieldSettings Value="FirstName"
GroupBy="Country"></AutoCompleteFieldSettings>
</SfAutoComplete>
@code {
public class EmployeeData
{
public string FirstName { get; set; }
public string Country { get; set; }
}
List<EmployeeData> Data = new List<EmployeeData>
{
new EmployeeData() { FirstName = "Andrew Fuller", Country = "England" },
new EmployeeData() { FirstName = "Anne Dodsworth", Country = "USA" },
new EmployeeData() { FirstName = "Janet Leverling", Country = "USA" },
new EmployeeData() { FirstName = "Laura Callahan", Country = "USA"},

```

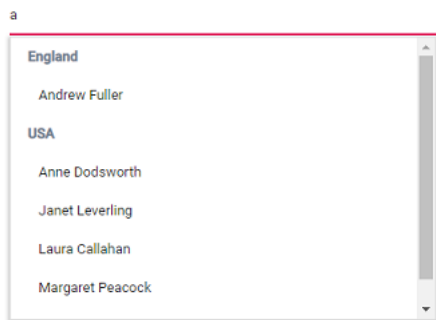


```

new EmployeeData() { FirstName = "Margaret Peacock", Country = "USA"},
new EmployeeData() { FirstName = "Michael Suyama", Country = "USA", },
new EmployeeData() { FirstName = "Nancy Davolio", Country = "USA"},
new EmployeeData() { FirstName = "Robert King", Country = "England"},
new EmployeeData() { FirstName = "Steven Buchanan", Country = "England"},
};
}
<style>
.group {
color: slategrey;
}
</style>

```

The output will be as follows.



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.

ASPX-CS

```

@using Syncfusion.Blazor.Data
@using Syncfusion.Blazor.DropDowns
<SfAutoComplete TValue="string" TItem="EmployeeData" Placeholder="Select a customer" DataSource="@Data">
<AutoCompleteTemplates TItem="EmployeeData">
<ItemTemplate>
<span class='item'><span class='name'>@((context as EmployeeData).FirstName)</span><span class='city'>@((context as EmployeeData).Country)</span></span></ItemTemplate>
<HeaderTemplate>
<span class='head'><span class='name'>Name</span><span class='city'>Country</span></span></HeaderTemplate>
</AutoCompleteTemplates>
<AutoCompleteFieldSettings Value="FirstName"></AutoCompleteFieldSettings>
</SfAutoComplete>
@code {
public class EmployeeData
{
public string FirstName { get; set; }
}
}

```

```
public string Country { get; set; }
}
List<EmployeeData> Data = new List<EmployeeData>
{
    new EmployeeData() { FirstName = "Andrew Fuller", Country = "England" },
    new EmployeeData() { FirstName = "Anne Dodsworth", Country = "USA" },
    new EmployeeData() { FirstName = "Janet Leverling", Country = "USA" },
    new EmployeeData() { FirstName = "Laura Callahan", Country = "USA"},
    new EmployeeData() { FirstName = "Margaret Peacock", Country = "USA"},
    new EmployeeData() { FirstName = "Michael Suyama", Country = "USA", },
    new EmployeeData() { FirstName = "Nancy Davolio", Country = "USA"},
    new EmployeeData() { FirstName = "Robert King", Country = "England"},
    new EmployeeData() { FirstName = "Steven Buchanan", Country = "England"},
};
}
<style>
.head, .item {
display: table;
width: 100%;
margin: auto;
}
.head {
height: 40px;
font-size: 15px;
font-weight: 600;
}
.name, .city {
display: table-cell;
vertical-align: middle;
width: 50%;
}
.head .name {
text-indent: 16px;
}
</style>
```

The output will be as follows.

a

Name	Country
Andrew Fuller	England
Anne Dodsworth	USA
Janet Leverling	USA
Laura Callahan	USA
Margaret Peacock	USA
Michael Suyama	USA

Footer template

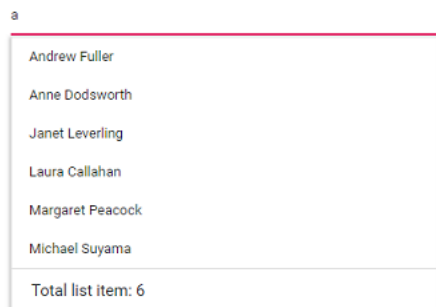
The AutoComplete has options to show a footer element at the bottom of the list items in the suggestion list. Here, you can place any custom element as a footer element using `FooterTemplate` property.

In the following sample, footer element displays the total number of list items present in the AutoComplete.

ASPX-CS

```
@using Syncfusion.Blazor.Data
@using Syncfusion.Blazor.DropDowns
<SfAutoComplete TValue="string" TItem="EmployeeData" DataSource="@Data"
Placeholder="Select a customer">
  <AutoCompleteTemplates TItem="EmployeeData">
    <FooterTemplate>
      <span class='footer'>Total list item: 6</span>
    </FooterTemplate>
  </AutoCompleteTemplates>
  <AutoCompleteFieldSettings Value="FirstName"></AutoCompleteFieldSettings>
</SfAutoComplete>
@code {
public class EmployeeData
{
public string FirstName { get; set; }
public string Country { get; set; }
}
List<EmployeeData> Data = new List<EmployeeData>
{
new EmployeeData() { FirstName = "Andrew Fuller", Country = "England" },
new EmployeeData() { FirstName = "Anne Dodsworth", Country = "USA" },
new EmployeeData() { FirstName = "Janet Leverling", Country = "USA" },
new EmployeeData() { FirstName = "Laura Callahan", Country = "USA"},
new EmployeeData() { FirstName = "Margaret Peacock", Country = "USA"},
new EmployeeData() { FirstName = "Michael Suyama", Country = "USA", },
};
}
<style>
.footer {
text-indent: 1.2em;
display: block;
font-size: 15px;
line-height: 40px;
border-top: 1px solid #e0e0e0;
}
</style>
```

The output will be as follows.



No records template

The AutoComplete is provided with support to custom design the suggestion list content when no data is found and no matches found on search with the help of the [NoRecordsTemplate](#) property.

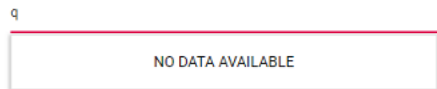
In the following sample, suggestion list content displays the notification of no data available.

ASPX-CS

```
@using Syncfusion.Blazor.DropDowns
<SfAutoComplete TValue="string" TItem="Countries" Placeholder="Select a customer" DataSource="@Country">
  <AutoCompleteTemplates TItem="Countries">
    <NoRecordsTemplate>
      <span class='norecord'> NO DATA AVAILABLE</span>
    </NoRecordsTemplate>
  </AutoCompleteTemplates>
  <AutoCompleteFieldSettings Value="Name"></AutoCompleteFieldSettings>
</SfAutoComplete>

@code {
public class EmployeeData
{
public string Name { get; set; }
}
public EmployeeData Data = new EmployeeData();
public class Countries
{
public string Name { get; set; }
public string Code { get; set; }
}
List<Countries> Country = new List<Countries>
{
new Countries() { Name = "Australia", Code = "AU" },
new Countries() { Name = "Bermuda", Code = "BM" },
new Countries() { Name = "Canada", Code = "CA" },
new Countries() { Name = "Cameroon", Code = "CM" },
new Countries() { Name = "Denmark", Code = "DK" },
};
}
```

The output will be as follows.



Action failure template

There is also an option to custom design the suggestion 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.

ASPX-CS

```
@using Syncfusion.Blazor.Data
@using Syncfusion.Blazor.DropDowns
<SfAutoComplete TValue="string" TItem="OrderDetails" Placeholder="Select a name" Query="@RemoteDataQuery">
```

```

<AutoCompleteTemplates TItem="OrderDetails">
  <ActionFailureTemplate>
    <span class='norecord'>Data fetch get fails </span>
  </ActionFailureTemplate>
</AutoCompleteTemplates>
<SfDataManager
  Url="https://js.syncfusion.com/demos/ejServices/Wcf/Northwind.svc/Orders"
  CrossDomain="true"
  Adaptor="Syncfusion.Blazor.Adaptors.ODataAdaptor"></SfDataManager>
<AutoCompleteFieldSettings Value="CustomerID"></AutoCompleteFieldSettings>
</SfAutoComplete>
@code {
  public Query RemoteDataQuery = new Query().Select(new List<string> {
    "CustomerID" }).Take(6).RequiresCount();
  public Syncfusion.Blazor.Lists.SortOrder Sort { get; set; } =
    Syncfusion.Blazor.Lists.SortOrder.Ascending;
  public class OrderDetails
  {
    public int? OrderID { get; set; }
    public string CustomerID { get; set; }
    public int? EmployeeID { get; set; }
    public double? Freight { get; set; }
    public string ShipCity { get; set; }
    public bool Verified { get; set; }
    public DateTime? OrderDate { get; set; }
    public string ShipName { get; set; }
    public string ShipCountry { get; set; }
    public DateTime? ShippedDate { get; set; }
    public string ShipAddress { get; set; }
  }
}

```

Localization in Blazor AutoComplete Component

Blazor server side

Add `UseRequestLocalization` middle-ware in Configure method in **Startup.cs** file to get browser Culture Info.

Refer the following code to add configuration in Startup.cs file

C# SHARP

```

using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Localization;
namespace BlazorApplication
{
  public class Startup
  {
    ....
    ....
    public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
    {
      app.UseRequestLocalization();
      ....
      ....
    }
  }
}

```

```
}
```

The **Localization** library allows you to localize default text content. The AutoComplete component has static text that can be changed to other cultures (Arabic, Deutsch, French, etc.).

In the following examples, demonstrate how to enable **Localization** for AutoComplete in server side Blazor samples. Here, we have used Resource file to translate the static text.

The Resource file is an XML file which contains the strings(key and value pairs) that you want to translate into different language. You can also refer Localization [link](#) to know more about how to configure and use localization in the ASP.NET Core application framework.

- Open the **Startup.cs** file and add the below configuration in the **ConfigureServices** function as follows.

CSHARP

```
using Syncfusion.Blazor;
using System.Globalization;
using Microsoft.AspNetCore.Localization;
namespace BlazorApplication
{
    public class Startup
    {
        ....
        ....
        public void ConfigureServices(IServiceCollection services)
        {
            ....
            ....
            services.AddSyncfusionBlazor();
            services.AddLocalization(options => options.ResourcesPath = "Resources");
            services.Configure<RequestLocalizationOptions>(options =>
            {
                // define the list of cultures your app will support
                var supportedCultures = new List<CultureInfo>()
                {
                    new CultureInfo("de")
                };
                // set the default culture
                options.DefaultRequestCulture = new RequestCulture("de");
                options.SupportedCultures = supportedCultures;
                options.SupportedUICultures = supportedCultures;
                options.RequestCultureProviders = new List<IRequestCultureProvider>() {
                    new QueryStringRequestCultureProvider() // Here, You can also use other
                    localization provider
                };
            });
            services.AddSingleton(typeof(ISyncfusionStringLocalizer),
                typeof(SampleLocalizer));
        }
    }
}
```

- Then, write a **class** by inheriting **ISyncfusionStringLocalizer** interface and override the Manager property to get the resource file details from the application end.

CSHARP

```
using Syncfusion.Blazor;
namespace blazorDropdowns
{
    public class SampleLocalizer : ISyncfusionStringLocalizer
    {
        public string Get(string key)
        {
            return this.Manager.GetString(key);
        }
        public System.Resources.ResourceManager Manager
        {
            get
            {
                return blazorDropdowns.Resources.SyncfusionBlazorLocale.ResourceManager;
            }
        }
    }
}
```

- Add **.resx** file to Resource folder and enter the key value (Locale Keywords) in the **Name** column and the translated string in the Value column as follows.

Name	Value (in Deutsch culture)
---	---
AutoComplete_ActionFailureTemplate	Die Anfrage ist fehlgeschlagen
AutoComplete_NoRecordsTemplate	Keine Aufzeichnungen gefunden

- Finally, Specify the culture for AutoComplete using [locale](#) property.

ASPX-CS

```
@using Syncfusion.Blazor.DropDowns
<SfAutoComplete TValue="string" TItem="Countries" Placeholder="e.g.
Australia" Locale="de" DataSource="@LocalData">
<AutoCompleteFieldSettings Value="Name"></AutoCompleteFieldSettings>
</SfAutoComplete>
@code {
    public class Countries
    {
        public string Name { get; set; }
        public string Code { get; set; }
    }
    List<Countries> LocalData = new List<Countries> {
        new Countries() { Name = "Australia", Code = "AU" },
        new Countries() { Name = "Bermuda", Code = "BM" },
        new Countries() { Name = "Canada", Code = "CA" },
    }
```

```
new Countries() { Name = "Cameroon", Code = "CM" },
new Countries() { Name = "Denmark", Code = "DK" }
};
}
```

Blazor WebAssembly

The Localization library allows you to localize static text content of the `NoRecordsTemplate` and `ActionFailureTemplate` properties according to the culture currently assigned to the AutoComplete.

| Locale key | en-US (default)

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

| NoRecordsTemplate | No Records Found

| ActionFailureTemplate | The Request Failed

The following steps explain how to render the AutoComplete in French culture (fr) in Blazor Web Assembly application.

- Open the `program.cs` file and add the below configuration in the **Builder ConfigureServices** function as follows.

CSHARP

```
using Syncfusion.Blazor;
using Microsoft.AspNetCore.Builder;
namespace WebAssemblyLocale
{
    public class Program
    {
        public static async Task Main(string[] args)
        {
            ....
            ....
            builder.Services.Configure<RequestLocalizationOptions>(options =>
            {
                // Define the list of cultures your app will support
                var supportedCultures = new List<System.Globalization.CultureInfo>()
                {
                    new System.Globalization.CultureInfo("en-US"),
                    new System.Globalization.CultureInfo("fr"),
                };
                // Set the default culture
                options.DefaultRequestCulture = new
                Microsoft.AspNetCore.Localization.RequestCulture("fr");
                options.SupportedCultures = supportedCultures;
                options.SupportedUICultures = supportedCultures;
                options.RequestCultureProviders = new
                List<Microsoft.AspNetCore.Localization.IRequestCultureProvider>() {
                    new Microsoft.AspNetCore.Localization.QueryStringRequestCultureProvider()
                };
            });
            ....
            ....
        }
    }
}
```



```
}
}
```

- To download the locale definition of Blazor components, use this [link](#).
- After downloading the blazor-locale package, copy the blazor-locale folder with required local definition file into wwwroot folder.
- By default, the blazor-locale package contains the localized text for static text present in components like button text, placeholder, tooltip, and more.
- Set the culture by using the SetCulture method.

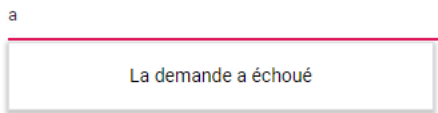
In the following sample, French culture is set to the AutoComplete and no data is loaded. Hence, the [NoRecordsTemplate](#) property displays its text in French culture initially and if the sample is run offline, the [ActionFailureTemplate](#) property displays its text appropriately.

ASPX-CS

```
@using Syncfusion.Blazor.Data
@using Syncfusion.Blazor.DropDowns
@inject HttpClient Http;
<SfAutoComplete TValue="string" TItem="EmployeeData" Query="@Query"
Placeholder="Select a customer" Locale="fr">
<SfDataManager Url="https://ej2services.syncfusion.com/production/web-
services/api/Employees" Adaptor="Adaptors.WebApiAdaptor"
CrossDomain=true></SfDataManager>
<AutoCompleteFieldSettings Value="FirstName"></AutoCompleteFieldSettings>
</SfAutoComplete>
@code {
[Inject]
protected IJSRuntime JsRuntime { get; set; }
protected override async Task OnInitializedAsync()
{
this.JsRuntime.Sf().LoadLocaleData(await Http.GetJsonAsync<object>("blazor-
locale/src/fr.json")).SetCulture("fr");
}
public Query Query = new Query();
public class EmployeeData
{
public int EmployeeID { get; set; }
public string FirstName { get; set; }
public string Designation { get; set; }
public string Country { get; set; }
}
}
```

The output will be as follows.

a



Style and appearance in Blazor AutoComplete 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-icon.e-ddl-icon.e-icons, .e-ddl .e-input-group-icon.e-ddl-icon.e-icons: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-lib.e-input-group.e-control-wrapper.e-control-container.e-float-  
input.e-input-focus .e-float-text.e-label-top {  
  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 placeholder to add mandatory indicator(*)

Use the following CSS to add the mandatory indicator * to the float label element.

CSS

```
.e-input-group.e-control-wrapper.e-control-container.e-float-input .e-float-  
text::after {  
  content: "*";  
  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 search icon in the Blazor AutoComplete component.

You can add the search icon to the AutoComplete component by overriding the content of the existing icon. The following code demonstrates how to add a search icon to the AutoComplete component.

ASPX-CS

```
@using Syncfusion.Blazor.DropDowns
<SfAutoComplete TValue="string" TItem="GameFields" Width="300px"
ShowPopupButton="true" Placeholder="e.g. Basketball" DataSource="@Games">
<AutoCompleteFieldSettings Value="Text"></AutoCompleteFieldSettings>
</SfAutoComplete>
<style>
.e-ddl.e-input-group.e-control-wrapper .e-ddl-icon::before {
content: '\e724';
font-family: 'e-icons';
font-size: 16px;
opacity: 0.4;
}
</style>
@code{
public class GameFields
{
public string ID { get; set; }
public string Text { get; set; }
}
public List<GameFields> Games = new List<GameFields>()
{
new GameFields(){ ID= "Game1", Text= "American Football" },
new GameFields(){ ID= "Game2", Text= "Badminton" },
new GameFields(){ ID= "Game3", Text= "Basketball" },
}
```

```
new GameFields() { ID= "Game4", Text= "Cricket" },
new GameFields() { ID= "Game5", Text= "Football" },
};
}
```

The output will be as follows.

e.g. Basketball



Virtualization in Blazor AutoComplete Component

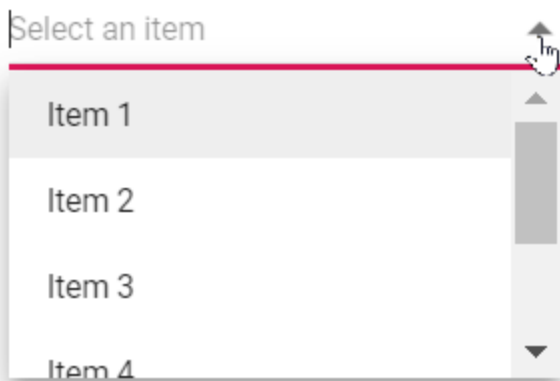
The AutoComplete has been provided virtualization to improve the UI performance for a large amount of data when [EnableVirtualization](#) is true. This feature doesn't render out the entire data source on initial component rendering. It loads the N number of items in the popup on initial rendering and the remaining set number of items will load on each scrolling action in the popup. It can work with both local and remote data.

In the following code 150 items bound to the component, but only 6 items will load to the popup when you open the popup. Remaining set number of items will load on each scrolling action in the popup.

ASPX-CS

```
@using Syncfusion.Blazor.DropDowns
@using Syncfusion.Blazor.Data
<SfAutoComplete TValue="string" TItem="Record" Placeholder="Select an item"
DataSource="@Records" Query="@LocalDataQuery" PopupHeight="130px"
EnableVirtualization="true" ShowPopupButton="true">
<AutoCompleteFieldSettings Value="Text"></AutoCompleteFieldSettings>
</SfAutoComplete>
@code{
public Query LocalDataQuery = new Query().Take(6);
public class Record
{
public string ID { get; set; }
public string Text { get; set; }
}
public List<Record> Records { get; set; }
protected override void OnInitialized()
{
this.Records = Enumerable.Range(1, 150).Select(i => new Record()
{
ID = i.ToString(),
Text = "Item " + i,
}).ToList();
}
}
```

The output will shown as follows,



Native Events in Blazor AutoComplete Component

The following section explains the steps to include native events and pass data to event handler in the [AutoComplete](#) component.

Bind native events to AutoComplete

You can access any native event by using on `<event>` attribute with a component. The attribute's value is treated as an event handler.

In the following example, the `keyPressed` method is called every time the key is pressed on input.

ASPX-CS

```
<SfAutoComplete TValue="string" TItem="Countries" @onkeypress="@KeyPressed">
  <AutoCompleteFieldSettings Value="Name"></AutoCompleteFieldSettings>
</SfAutoComplete>
@code {
  public void KeyPressed()
  {
    Console.WriteLine("Key Pressed!");
  }
  public class Countries
  {
    public string Name { get; set; }
    public string Code { get; set; }
  }
  List<Countries> LocalData = new List<Countries> {
    new Countries() { Name = "Australia", Code = "AU" },
    new Countries() { Name = "Bermuda", Code = "BM" },
    new Countries() { Name = "Canada", Code = "CA" },
    new Countries() { Name = "Cameroon", Code = "CM" },
    new Countries() { Name = "Denmark", Code = "DK" },
    new Countries() { Name = "France", Code = "FR" },
  };
}
```

Also, you can rewrite the above example code as follows using Lambda expressions.

ASPX-CS

```
<SfAutoComplete TValue="string" @onkeypress="@(() => Console.WriteLine("Key Pressed!"))"></SfAutoComplete>
```

Pass event data to event handler

Blazor provides set of argument types to map to native events. The list of event types and event arguments are:

- Focus Events - FocusEventArgs
- Mouse Events - MouseEventArgs
- Keyboard Events - KeyboardEventArgs
- Input Events - ChangeEventArgs/EventArgs
- Touch Events – TouchEventArgs
- Pointer Events – PointerEventArgs

In the following example, the on keypress method is called every time any key is pressed inside input. But the message will print when you press "a" key.

ASPX-CS

```
<SfAutoComplete TValue="string" TItem="Countries" @onkeypress="@ (e => KeyPressed(e)) " >
<AutoCompleteFieldSettings Value="Name"></AutoCompleteFieldSettings>
</SfAutoComplete>
@code {
public void KeyPressed(KeyboardEventArgs args) {
if (args.Key == "a") {
Console.WriteLine("A was pressed");
}
}
public class Countries
{
public string Name { get; set; }
public string Code { get; set; }
}
List<Countries> LocalData = new List<Countries> {
new Countries() { Name = "Australia", Code = "AU" },
new Countries() { Name = "Bermuda", Code = "BM" },
new Countries() { Name = "Canada", Code = "CA" },
new Countries() { Name = "Cameroon", Code = "CM" },
new Countries() { Name = "Denmark", Code = "DK" },
new Countries() { Name = "France", Code = "FR" },
};
}
```

Using Lambda expression also, you can pass the event data to the event handler.

List of Native events supported

| List of Native events | | |

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

| onclick | onblur | onfocus | onfocusout |

| onmousemove | onmouseover | onmouseout | onmousedown | onmouseup |

| ondblclick | onkeydown | onkeyup | onkeypress |
 | ontouchend | onfocusin | onmouseup | ontouchstart |

Accessibility in Blazor AutoComplete 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.

ARIA attributes

The AutoComplete component uses the [combobox](#) role and each list item has an [option](#) role. The following [ARIA Attributes](#) denotes 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 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 selects the first item when popup list opens. |

| Enter | Selects the focused item and set to AutoComplete component. |

| Tab | Focuses the next tab indexed element when the popup is closed. Otherwise, closes the popup list and remains the focus in component if it is in 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 if it is in 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 open state then removes the selection. |

| Home | Cursor moves to before of the first character in input. |

| End | Cursor moves to next of the last character in input. |

In the following sample, disable the AutoComplete component using t keys.

ASPX-CS

```
@using Syncfusion.Blazor.DropDowns
<SfAutoComplete TValue="string" TItem="Countries" @ref="AutoObj"
Placeholder="Select a country" Enabled="@enable" @onkeypress="@ (e =>
KeyPressed(e))" DataSource="@LocalData">
<AutoCompleteFieldSettings Value="Name"></AutoCompleteFieldSettings>
</SfAutoComplete>
@code {
public SfAutoComplete<string, Countries> AutoObj;
public bool enable { get; set; } = true ;
public class Countries
{
public string Name { get; set; }
public string Code { get; set; }
}
List<Countries> LocalData = new List<Countries> {
new Countries() { Name = "Australia", Code = "AU" },
new Countries() { Name = "Bermuda", Code = "BM" },
new Countries() { Name = "Canada", Code = "CA" },
new Countries() { Name = "Cameroon", Code = "CM" },
new Countries() { Name = "Denmark", Code = "DK" },
new Countries() { Name = "France", Code = "FR" },
new Countries() { Name = "Finland", Code = "FI" },
new Countries() { Name = "Germany", Code = "DE" },
new Countries() { Name = "Greenland", Code = "GL" },
new Countries() { Name = "Hong Kong", Code = "HK" },
};
public void KeyPressed(KeyboardEventArgs args)
{
if (args.Key == "t")
{
enable = false;
}
}
}
```

The output will be as follows.

t
.....

Events in Blazor AutoComplete Component

This section explains the list of events of the AutoComplete component which will be triggered for appropriate AutoComplete actions.

Blur

Blur event triggers when the input loses focus.

ASPX-CS

```
@using Syncfusion.Blazor.DropDowns
<SfAutoComplete TItem="GameFields" TValue="string" DataSource="@Games">
  <AutoCompleteEvents TItem="GameFields" TValue="string"
  Blur="@BlurHandler"></AutoCompleteEvents>
  <AutoCompleteFieldSettings Text="Text"
  Value="ID"></AutoCompleteFieldSettings>
</SfAutoComplete>
@code {
public class GameFields
{
public string ID { get; set; }
public string Text { get; set; }
}
private List<GameFields> Games = new List<GameFields>() {
new GameFields(){ ID= "Game1", Text= "American Football" },
new GameFields(){ ID= "Game2", Text= "Badminton" },
new GameFields(){ ID= "Game3", Text= "Basketball" },
new GameFields(){ ID= "Game4", Text= "Cricket" },
};
private void BlurHandler(Object args)
{
// Here you can customize your code
}
}
```

ValueChange

ValueChange event triggers when the AutoComplete value is changed.

ASPX-CS

```
@using Syncfusion.Blazor.DropDowns
<SfAutoComplete TItem="GameFields" TValue="string" DataSource="@Games">
  <AutoCompleteEvents TItem="GameFields" TValue="string"
  ValueChange="@ValueChangeHandler"></AutoCompleteEvents>
  <AutoCompleteFieldSettings Text="Text"
  Value="ID"></AutoCompleteFieldSettings>
</SfAutoComplete>
@code {
public class GameFields
{
public string ID { get; set; }
public string Text { get; set; }
}
private List<GameFields> Games = new List<GameFields>() {
new GameFields(){ ID= "Game1", Text= "American Football" },
new GameFields(){ ID= "Game2", Text= "Badminton" },
new GameFields(){ ID= "Game3", Text= "Basketball" },
}
```

```
new GameFields(){ ID= "Game4", Text= "Cricket" },
};
private void ValueChangeHandler(ChangeEventArgs<string, GameFields> args)
{
// Here you can customize your code
}
}
```

Closed

Closed event triggers after the popup has been closed.

ASPX-CS

```
@using Syncfusion.Blazor.DropDowns
<SfAutoComplete TItem="GameFields" TValue="string" DataSource="@Games">
<AutoCompleteEvents TItem="GameFields" TValue="string"
Closed="@CloseHandler"></AutoCompleteEvents>
<AutoCompleteFieldSettings Text="Text"
Value="ID"></AutoCompleteFieldSettings>
</SfAutoComplete>
@code {
public class GameFields
{
public string ID { get; set; }
public string Text { get; set; }
}
private List<GameFields> Games = new List<GameFields>() {
new GameFields(){ ID= "Game1", Text= "American Football" },
new GameFields(){ ID= "Game2", Text= "Badminton" },
new GameFields(){ ID= "Game3", Text= "Basketball" },
new GameFields(){ ID= "Game4", Text= "Cricket" },
};
private void CloseHandler(ClosedEventArgs args)
{
// Here you can customize your code
}
}
```

Created

Created event triggers when the component is created.

ASPX-CS

```
@using Syncfusion.Blazor.DropDowns
<SfAutoComplete TItem="GameFields" TValue="string" DataSource="@Games">
<AutoCompleteEvents TItem="GameFields" TValue="string"
Created="@CreatedHandler"></AutoCompleteEvents>
<AutoCompleteFieldSettings Text="Text"
Value="ID"></AutoCompleteFieldSettings>
</SfAutoComplete>
@code {
public class GameFields
{
public string ID { get; set; }
public string Text { get; set; }
}
```

```

}
private List<GameFields> Games = new List<GameFields>() {
new GameFields(){ ID= "Game1", Text= "American Football" },
new GameFields(){ ID= "Game2", Text= "Badminton" },
new GameFields(){ ID= "Game3", Text= "Basketball" },
new GameFields(){ ID= "Game4", Text= "Cricket" },
};
private void CreatedHandler(Object args)
{
// Here you can customize your code
}
}

```

Destroyed

Destroyed event triggers when the component is destroyed.

ASPX-CS

```

@using Syncfusion.Blazor.DropDowns
<SfAutoComplete TItem="GameFields" TValue="string" DataSource="@Games">
<AutoCompleteEvents TItem="GameFields" TValue="string"
Destroyed="@DestroyedHandler"></AutoCompleteEvents>
<AutoCompleteFieldSettings Text="Text"
Value="ID"></AutoCompleteFieldSettings>
</SfAutoComplete>
@code {
public class GameFields
{
public string ID { get; set; }
public string Text { get; set; }
}
private List<GameFields> Games = new List<GameFields>() {
new GameFields(){ ID= "Game1", Text= "American Football" },
new GameFields(){ ID= "Game2", Text= "Badminton" },
new GameFields(){ ID= "Game3", Text= "Basketball" },
new GameFields(){ ID= "Game4", Text= "Cricket" },
};
private void DestroyedHandler(Object args)
{
// Here you can customize your code
}
}

```

Focus

Focus event triggers when the input gets focus.

ASPX-CS

```

@using Syncfusion.Blazor.DropDowns
<SfAutoComplete TItem="GameFields" TValue="string" DataSource="@Games">
<AutoCompleteEvents TItem="GameFields" TValue="string"
Focus="@FocusHandler"></AutoCompleteEvents>
<AutoCompleteFieldSettings Text="Text"
Value="ID"></AutoCompleteFieldSettings>
</SfAutoComplete>

```

```
@code {
public class GameFields
{
public string ID { get; set; }
public string Text { get; set; }
}
private List<GameFields> Games = new List<GameFields>() {
new GameFields(){ ID= "Game1", Text= "American Football" },
new GameFields(){ ID= "Game2", Text= "Badminton" },
new GameFields(){ ID= "Game3", Text= "Basketball" },
new GameFields(){ ID= "Game4", Text= "Cricket" },
};
private void FocusHandler(Object args)
{
// Here you can customize your code
}
}
```

OnOpen

OnOpen event triggers when the popup is opened. If you cancel this event, the popup remains closed.

ASPX-CS

```
@using Syncfusion.Blazor.DropDowns
<SfAutoComplete TItem="GameFields" TValue="string" DataSource="@Games">
  <AutoCompleteEvents TItem="GameFields" TValue="string"
    OnOpen="@OnOpenHandler"></AutoCompleteEvents>
  <AutoCompleteFieldSettings Text="Text"
    Value="ID"></AutoCompleteFieldSettings>
</SfAutoComplete>
@code {
public class GameFields
{
public string ID { get; set; }
public string Text { get; set; }
}
private List<GameFields> Games = new List<GameFields>() {
new GameFields(){ ID= "Game1", Text= "American Football" },
new GameFields(){ ID= "Game2", Text= "Badminton" },
new GameFields(){ ID= "Game3", Text= "Basketball" },
new GameFields(){ ID= "Game4", Text= "Cricket" },
};
private void OnOpenHandler(BeforeOpenEventArgs args)
{
// Here you can customize your code
}
}
```

OnClose

OnClose event triggers before the popup is closed. If you cancel this event, the popup will remain open.

ASPX-CS

```
@using Syncfusion.Blazor.DropDowns
<SfAutoComplete TItem="GameFields" TValue="string" DataSource="@Games">
```

```

<AutoCompleteEvents TItem="GameFields" TValue="string"
OnClose="@OnCloseHandler"></AutoCompleteEvents>
<AutoCompleteFieldSettings Text="Text"
Value="ID"></AutoCompleteFieldSettings>
</SfAutoComplete>
@code {
public class GameFields
{
public string ID { get; set; }
public string Text { get; set; }
}
private List<GameFields> Games = new List<GameFields>() {
new GameFields(){ ID= "Game1", Text= "American Football" },
new GameFields(){ ID= "Game2", Text= "Badminton" },
new GameFields(){ ID= "Game3", Text= "Basketball" },
new GameFields(){ ID= "Game4", Text= "Cricket" },
};
private void OnCloseHandler(PopupEventArgs args)
{
// Here you can customize your code
}
}

```

DataBound

DataBound event triggers when the data source is populated in the popup list.

ASPX-CS

```

@using Syncfusion.Blazor.DropDowns
<SfAutoComplete TItem="GameFields" TValue="string" DataSource="@Games">
<AutoCompleteEvents TItem="GameFields" TValue="string"
DataBound="@DataBoundHandler"></AutoCompleteEvents>
<AutoCompleteFieldSettings Text="Text"
Value="ID"></AutoCompleteFieldSettings>
</SfAutoComplete>
@code {
public class GameFields
{
public string ID { get; set; }
public string Text { get; set; }
}
private List<GameFields> Games = new List<GameFields>() {
new GameFields(){ ID= "Game1", Text= "American Football" },
new GameFields(){ ID= "Game2", Text= "Badminton" },
new GameFields(){ ID= "Game3", Text= "Basketball" },
new GameFields(){ ID= "Game4", Text= "Cricket" },
};
private void DataBoundHandler(DataBoundEventArgs args)
{
// Here you can customize your code
}
}

```

Filtering

Filtering event triggers on typing a character in the filter bar when the AllowFiltering is enabled.

ASPX-CS

```

@using Syncfusion.Blazor.DropDowns
<SfAutoComplete TItem="GameFields" TValue="string" AllowFiltering="true"
DataSource="@Games">
<AutoCompleteEvents TItem="GameFields" TValue="string"
Filtering="@Filteringhandler"></AutoCompleteEvents>
<AutoCompleteFieldSettings Text="Text"
Value="ID"></AutoCompleteFieldSettings>
</SfAutoComplete>
@code {
public class GameFields
{
public string ID { get; set; }
public string Text { get; set; }
}
private List<GameFields> Games = new List<GameFields>() {
new GameFields(){ ID= "Game1", Text= "American Football" },
new GameFields(){ ID= "Game2", Text= "Badminton" },
new GameFields(){ ID= "Game3", Text= "Basketball" },
new GameFields(){ ID= "Game4", Text= "Cricket" },
};
private void Filteringhandler(FilteringEventArgs args)
{
// Here you can customize your code
}
}

```

OnActionBegin

OnActionBegin event triggers before fetching data from the remote server.

ASPX-CS

```

@using Syncfusion.Blazor.DropDowns
@using Syncfusion.Blazor.Data
<SfAutoComplete TValue="string" TItem="OrderDetails"
Query="@RemoteDataQuery">
<SfDataManager
Url="https://js.syncfusion.com/demos/ejServices/Wcf/Northwind.svc/Orders"
CrossDomain="true"
Adaptor="Syncfusion.Blazor.Adaptors.ODataAdaptor"></SfDataManager>
<AutoCompleteEvents TValue="string" TItem="OrderDetails"
OnActionBegin="@OnActionBeginhandler"></AutoCompleteEvents>
<AutoCompleteFieldSettings Text="CustomerID"
Value="CustomerID"></AutoCompleteFieldSettings>
</SfAutoComplete>
@code {
public Query RemoteDataQuery = new Query().Select(new List<string> {
"CustomerID" }).Take(6).RequiresCount();
public class OrderDetails
{
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public int? EmployeeID { get; set; }
public double? Freight { get; set; }
public string ShipCity { get; set; }
}
}

```

```

public bool Verified { get; set; }
public DateTime? OrderDate { get; set; }
public string ShipName { get; set; }
public string ShipCountry { get; set; }
public DateTime? ShippedDate { get; set; }
public string ShipAddress { get; set; }
}
private void OnActionBeginhandler(ActionBeginEventArgs args)
{
    // Here you can customize your code
}
}

```

OnActionFailure

OnActionFailure event triggers when the data fetch request from the remote server fails.

ASPX-CS

```

@using Syncfusion.Blazor.DropDowns
@using Syncfusion.Blazor.Data
<SfAutoComplete TValue="string" TItem="OrderDetails"
Query="@RemoteDataQuery">
<SfDataManager
Url="https://js.syncfusion.com/demos/ejServices/Wcf/Northwind.svc/Orders"
CrossDomain="true"
Adaptor="Syncfusion.Blazor.Adaptors.ODataAdaptor"></SfDataManager>
<AutoCompleteEvents TValue="string" TItem="OrderDetails"
OnActionFailure="@OnActionFailurehandler"></AutoCompleteEvents>
<AutoCompleteFieldSettings Text="CustomerID"
Value="CustomerID"></AutoCompleteFieldSettings>
</SfAutoComplete>
@code {
public Query RemoteDataQuery = new Query().Select(new List<string> {
"CustomerID" }).Take(6).RequiresCount();
public class OrderDetails
{
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public int? EmployeeID { get; set; }
public double? Freight { get; set; }
public string ShipCity { get; set; }
public bool Verified { get; set; }
public DateTime? OrderDate { get; set; }
public string ShipName { get; set; }
public string ShipCountry { get; set; }
public DateTime? ShippedDate { get; set; }
public string ShipAddress { get; set; }
}
private void OnActionFailurehandler(Exception args)
{
    // Here you can customize your code
}
}

```


OnValueSelect

OnValueSelect event triggers when a user selects an item in the popup using the mouse/tap or keyboard navigation.

ASPX-CS

```
@using Syncfusion.Blazor.DropDowns
<SfAutoComplete TItem="GameFields" TValue="string" AllowFiltering="true"
DataSource="@Games">
  <AutoCompleteEvents TItem="GameFields" TValue="string"
OnValueSelect="@OnValueSelecthandler"></AutoCompleteEvents>
  <AutoCompleteFieldSettings Text="Text"
Value="ID"></AutoCompleteFieldSettings>
</SfAutoComplete>
@code {
public class GameFields
{
public string ID { get; set; }
public string Text { get; set; }
}
private List<GameFields> Games = new List<GameFields>() {
new GameFields(){ ID= "Game1", Text= "American Football" },
new GameFields(){ ID= "Game2", Text= "Badminton" },
new GameFields(){ ID= "Game3", Text= "Basketball" },
new GameFields(){ ID= "Game4", Text= "Cricket" },
};
private void OnValueSelecthandler(SelectEventArgs<GameFields> args)
{
// Here you can customize your code
}
}
```

Opened

Opened event triggers when the popup opens.

ASPX-CS

```
@using Syncfusion.Blazor.DropDowns
<SfAutoComplete TItem="GameFields" TValue="string" AllowFiltering="true"
DataSource="@Games">
  <AutoCompleteEvents TItem="GameFields" TValue="string"
Opened="@Openedhandler"></AutoCompleteEvents>
  <AutoCompleteFieldSettings Text="Text"
Value="ID"></AutoCompleteFieldSettings>
</SfAutoComplete>
@code {
public class GameFields
{
public string ID { get; set; }
public string Text { get; set; }
}
private List<GameFields> Games = new List<GameFields>() {
new GameFields(){ ID= "Game1", Text= "American Football" },
new GameFields(){ ID= "Game2", Text= "Badminton" },
new GameFields(){ ID= "Game3", Text= "Basketball" },
new GameFields(){ ID= "Game4", Text= "Cricket" },
}
```

```
};  
private void Openedhandler(PopupEventArgs args)  
{  
    // Here you can customize your code  
}  
}
```

AutoComplete is limited with these events and new events will be added in the future based on the user requests. If the event you are looking for is not on the list, then please request [here](#).

Avatar

<!-- markdownlint-disable MD024 -->

Getting Started with Blazor Avatar Component

This section briefly explains about how to include a **Avatar** in your Blazor application. You can refer [Getting Started with Syncfusion Blazor for Server-Side in Visual Studio](#) page for the introduction and configuring the common specifications.

Importing Syncfusion Blazor component in the application

- Install [Syncfusion.Blazor](#) NuGet package to the application by using the **NuGet Package Manager**.
- You can add the client-side style resources through [CDN](#) or from [NuGet](#) package to the element of the `~/wwwroot/index.html` page in Blazor WebAssembly app or `~/Pages/_Host.cshtml` page in Blazor app.

HTML

```
<head>  
<link href="_content/Syncfusion.Blazor/styles/bootstrap4.css"  
rel="stylesheet" />  
</head>
```

HTML

```
<head>  
<link href="https://cdn.syncfusion.com/blazor/{{ site.blazorversion  
}}/styles/bootstrap4.css" rel="stylesheet" />  
</head>
```

- For **Internet Explorer 11** kindly refer the polyfills. Refer the [documentation](#) for more information.

HTML

```
<head>  
<link href="_content/Syncfusion.Blazor/styles/bootstrap4.css"  
rel="stylesheet" />
```

```
<script  
src="https://github.com/Daddoon/Blazor.Polyfill/releases/download/3.0.1/blaz  
or.polyfill.min.js"></script>  
</head>
```

Adding Avatar component to the application

Now, add the Syncfusion Avatar component in any web page `razor` in the `Pages` folder. For example, the Avatar component is added in the `~/Pages/Index.razor` page.

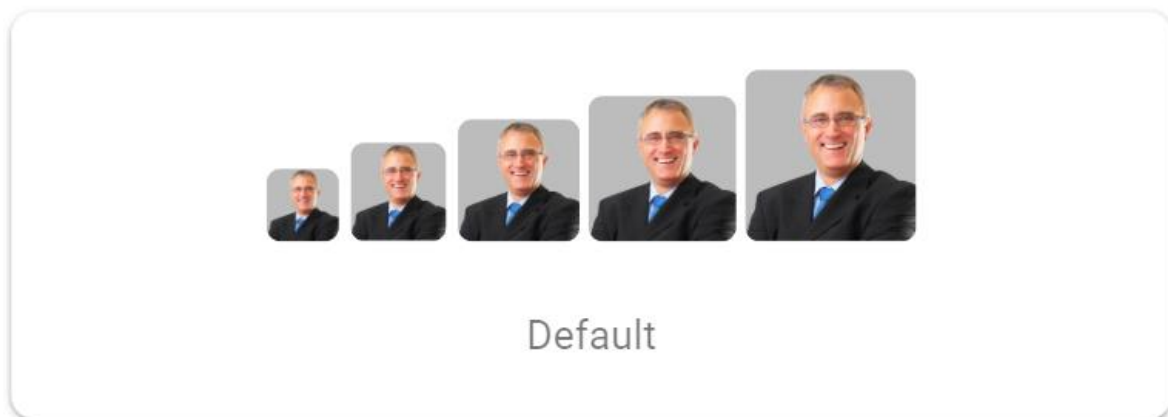
HTML

```
<!-- xSmall Avatar-->  
<div class="e-avatar e-avatar-xsmall image"></div>  
<!-- Small Avatar-->  
<div class="e-avatar e-avatar-small image"></div>  
<!-- Avatar-->  
<div class="e-avatar image"></div>  
<!-- Large Avatar-->  
<div class="e-avatar e-avatar-large image"></div>  
<!-- xLarge Avatar-->  
<div class="e-avatar e-avatar-xlarge image"></div>
```

Run the application

After successful compilation of your application, simply press `F5` to run the application.

Output be like the below.



Badge

<!-- markdownlint-disable MD024 -->

Getting Started with Blazor Badge Component

This section briefly explains about how to include a `Badge` in your Blazor server-side application. You can refer [Getting Started with Syncfusion Blazor for Server-Side in Visual Studio](#) page for the introduction and configuring the common specifications.

Importing Syncfusion Blazor component in the application

- Install [Syncfusion.Blazor](#) NuGet package to the application by using the [NuGet Package Manager](#).
- You can add the client-side style resources through [CDN](#) or from [NuGet](#) package to the element of the `~/wwwroot/index.html` page in Blazor WebAssembly app or `~/Pages/_Host.cshtml` page in Blazor Server app.

HTML

```
<head>
<link href="_content/Syncfusion.Blazor/styles/bootstrap4.css"
rel="stylesheet" />
</head>
```

HTML

```
<head>
<link href="https://cdn.syncfusion.com/blazor/{{ site.blazorversion
}}/styles/bootstrap4.css" rel="stylesheet" />
</head>
```

- For Internet Explorer 11 kindly refer the polyfills. Refer the [documentation](#) for more information.

HTML

```
<head>
<link href="_content/Syncfusion.Blazor/styles/bootstrap4.css"
rel="stylesheet" />
<script
src="https://github.com/Daddoon/Blazor.Polyfill/releases/download/3.0.1/blaz
or.polyfill.min.js"></script>
</head>
```

Adding Badge component to the application

Now, add the Syncfusion Blazor Badge component in any web page `razor` in the `Pages` folder. For example, the Badge component is added in the `~/Pages/Index.razor` page.

ASPX-CS

```
<h1>Badge Component <span class="e-badge">New</span></h1>
```

Run the application

After successful compilation of your application, simply press `F5` to run the application.

Output be like the below.

Badge Component New

Barcode

Getting Started with Blazor Barcode Component

This section briefly explains about how to include a BarcodeGenerator in your Blazor Server-Side application. You can refer [Getting Started with Syncfusion Blazor for Server-Side in Visual Studio](#) page for the introduction and configuring the common specifications.

Importing Syncfusion Blazor component in the application

1. Install the [Syncfusion.Blazor.BarcodeGenerator](#) NuGet package to the application by using the [NuGet Package Manager](#).
2. You can add the client-side resources through [CDN](#) or from [NuGet](#) package in the **HEAD** element of the `~/Pages/_Host.cshtml` page.

ASPX-CS

```
<head>
<environment include="Development">
<link href="_content/Syncfusion.Blazor.Themes/bootstrap4.css"
rel="stylesheet" />
</environment>
</head>
```

For Internet Explorer 11 kindly refer the polyfills. Refer the [documentation](#) for more information.

ASPX-CS

```
<head>
<environment include="Development">
<link href="_content/Syncfusion.Blazor.Themes/bootstrap4.css"
rel="stylesheet" />
<script
src="https://github.com/Daddoon/Blazor.Polyfill/releases/download/3.0.1/blaz
or.polyfill.min.js"></script>
</environment>
</head>
```

Adding component package to the application

Open `~/_Imports.Blazor` file and import the `Syncfusion.Blazor.BarcodeGenerator` packages.

ASPX-CS

```
@using Syncfusion.Blazor
@using Syncfusion.Blazor.BarcodeGenerator
```

Add SyncfusionBlazor service in Startup.cs

Open the **Startup.cs** file and add services required by Syncfusion components using **services.AddSyncfusionBlazor()** method. Add this method in the **ConfigureServices** function as follows.

ASPX-CS

```
@using Syncfusion.Blazor;
namespace BlazorApplication
{
    public class Startup
    {
        ....
        ....
        public void ConfigureServices(IServiceCollection services)
        {
            ....
            ....
            services.AddSyncfusionBlazor();
        }
    }
}
```

Note: To enable custom client side resource loading from CRG or CDN. You need to disable resource loading by **AddSyncfusionBlazor(true)** and load the scripts in the **HEAD** element of the **~/Pages/_Host.cshtml** page.

ASPX-CS

```
<head>
<environment include="Development">
<script src="https://cdn.syncfusion.com/blazor/{ site.blazorversion
  }/syncfusion-blazor.min.js"></script>
</environment>
</head>
```

Adding BarcodeGenerator component to the Application

BarcodeGenerator component can be rendered by using the [SfBarcodeGenerator](#) tag helper in ASP.NET Core Blazor application. Add the BarcodeGenerator component in any web page Blazor in the Pages folder. For example, the BarcodeGenerator component is added in the **~/Pages/Index.Blazor** page.

The following example shows a basic BarcodeGenerator component.

ASPX-CS

```
<SfBarcodeGenerator Width="200px" Height="150px" Mode="@RenderingMode.SVG"
  Type="@BarcodeType.Codabar" Value="123456789"></SfBarcodeGenerator>
```



Running the above code will display the barcode generator component on the browser.

Adding QR Generator control

You can add the QR code in our barcode generator component.

ASPX-CS

```
<SfQRCodeGenerator Width="200px" Height="150px"  
Value="Syncfusion"></SfQRCodeGenerator>
```



Syncfusion

Adding Data Matrix Generator control

You can add the Data Matrix code in our barcode generator component.

ASPX-CS

```
<SfDataMatrixGenerator Width="200" Height="150" Mode="@RenderingMode.SVG"  
Value="SYNCFUSION"></SfDataMatrixGenerator>
```



SYNCFUSION

See Also

- [Getting Started with Syncfusion Blazor for Client-Side in .NET Core CLI](#)
- [Getting Started with Syncfusion Blazor for Server-Side in Visual Studio](#)
- [Getting Started with Syncfusion Blazor for Server-Side in .NET Core CLI](#)

Barcode Generator in Blazor 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.

ASPX-CS

```
@using Syncfusion.Blazor.BarcodeGenerator
<SfBarcodeGenerator Width="200px" Height="150px" Type="@BarcodeType.Code39"
Value="SYNCFUSION"></SfBarcodeGenerator>
```



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.

ASPX-CS

```
@using Syncfusion.Blazor.BarcodeGenerator
<SfBarcodeGenerator Width="200px" Height="150px"
Type="@BarcodeType.Code39Extension" Value="SYNCFUSION"></SfBarcodeGenerator>
```




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.

ASPX-CS

```
@using Syncfusion.Blazor.BarcodeGenerator
<SfBarcodeGenerator Width="200px" Height="150px" Type="@BarcodeType.Code11"
Value="112"></SfBarcodeGenerator>
```



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.

ASPX-CS

```
@using Syncfusion.Blazor.BarcodeGenerator
<SfBarcodeGenerator Width="200px" Height="150px" Type="@BarcodeType.Codabar"
Value="123456789"></SfBarcodeGenerator>
```



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.

ASPX-CS

```
@using Syncfusion.Blazor.BarcodeGenerator
<SfBarcodeGenerator Width="200px" Height="150px" Type="@BarcodeType.Code32"
Value="01234567"></SfBarcodeGenerator>
```



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 +.

ASPX-CS

```
@using Syncfusion.Blazor.BarcodeGenerator
<SfBarcodeGenerator Width="200px" Height="150px" Type="@BarcodeType.Code93"
Value="01234567"></SfBarcodeGenerator>
```



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.

ASPX-CS

```
@using Syncfusion.Blazor.BarcodeGenerator
<SfBarcodeGenerator Width="200px" Height="150px" Type="@BarcodeType.Code128"
Value="SYNCFUSION"></SfBarcodeGenerator>
```



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 .

ASPX-CS

```
@using Syncfusion.Blazor.BarcodeGenerator
<SfBarcodeGenerator Width="200px" Height="150px" Type="@BarcodeType.Code128"
foreColor="red" Value="SYNCFUSION"></SfBarcodeGenerator>
```



Customizing the Barcode dimension

The dimension of the barcode can be changed using the [Height](#) and [Width](#) property of the barcode generator.

ASPX-CS

```
@using Syncfusion.Blazor.BarcodeGenerator
<SfBarcodeGenerator Width="300px" Height="300px" Type="@BarcodeType.Code128"
Value="SYNCFUSION"></SfBarcodeGenerator>
```



Customizing the text

In barcode generators you can customize the barcode text by using display [Text](#) property .

ASPX-CS

```
@using Syncfusion.Blazor.BarcodeGenerator
<SfBarcodeGenerator Width="300px" Height="300px" Type="@BarcodeType.Code128"
Value="SYNCFUSION">
<BarcodeGeneratorDisplayText text="Text"></BarcodeGeneratorDisplayText>
</SfBarcodeGenerator>
```



Event

[OnValidationFailed](#) event in the [SfBarcodeGenerator](#) is used to trigger when the input is an invalid string.

ASPX-CS

```
@using Syncfusion.Blazor.BarcodeGenerator
<SfBarcodeGenerator Width="200px" Height="150px" Type="@BarcodeType.Code128"
Value="SYNCFUSION"
OnValidationFailed="@OnValidationFailed"></SfBarcodeGenerator>
@code
{
    public void OnValidationFailed(ValidationFailedEventArgs args)
    {
    }
}
```

QR Code generator in Blazor 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.

ASPX-CS

```
@using Syncfusion.Blazor.BarcodeGenerator
<SfQRCodeGenerator Width="200px" Height="150px"
Value="Syncfusion"></SfQRCodeGenerator>
```



Syncfusion

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 .

ASPX-CS

```
@using Syncfusion.Blazor.BarcodeGenerator
<SfQRCodeGenerator Width="200px" Height="150px" ForeColor="red"
Value="Syncfusion"></SfQRCodeGenerator>
```



Syncfusion

Customizing the Barcode dimension

The dimension of the barcode can be changed using the [Height](#) and [Width](#) properties of the barcode generator.

ASPX-CS

```
@using Syncfusion.Blazor.BarcodeGenerator
<SfQRCodeGenerator Width="200px" Height="150px"
Value="Syncfusion"></SfQRCodeGenerator>
```

Customizing the text

In barcode generators You can customize the barcode text by using display [Text](#) property .

ASPX-CS

```
@using Syncfusion.Blazor.BarcodeGenerator
```

```
<SfQRCodeGenerator Width="200px" Height="150px" Value="Syncfusion">
  <QRCodeGeneratorDisplayText text="Text"></QRCodeGeneratorDisplayText>
</SfQRCodeGenerator>
```



Text

Event

[OnValidationFailed](#) event in the [SfQRCodeGenerator](#) is used to trigger when the input is an invalid string.

ASPX-CS

```
@using Syncfusion.Blazor.BarcodeGenerator
<SfQRCodeGenerator Width="200px" Height="150px" Value="Syncfusion"
OnValidationFailed="@OnValidationFailed"></SfQRCodeGenerator>
@code
{
    public void OnValidationFailed(ValidationFailedEventArgs args)
    {
    }
}
```

Data Matrix generator in Blazor 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.

ASPX-CS

```
@using Syncfusion.Blazor.BarcodeGenerator
<SfDataMatrixGenerator Width="200" Height="150"
Value="SYNCFUSION"></SfDataMatrixGenerator>
```




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.

ASPX-CS

```
@using Syncfusion.Blazor.BarcodeGenerator
<SfDataMatrixGenerator Width="200" ForeColor='red' Height="150"
Value="SYNCFUSION"></SfDataMatrixGenerator>
```



Customizing the Barcode dimension

The dimension of the barcode can be changed using the [Height](#) and [Width](#) property of the barcode generator.

ASPX-CS

```
@using Syncfusion.Blazor.BarcodeGenerator
<SfDataMatrixGenerator Width="200" Height="150"
Value="SYNCFUSION"></SfDataMatrixGenerator>
```

Customizing the text

In barcode generators you can customize the barcode text by using the display [Text](#) property.

ASPX-CS

```
@using Syncfusion.Blazor.BarcodeGenerator
<SfDataMatrixGenerator Width="200" Height="150" Value="SYNCFUSION">
<DataMatrixGeneratorDisplayText
Text="Text"></DataMatrixGeneratorDisplayText>
</SfDataMatrixGenerator>
```



Text

Event

[OnValidationFailed](#) event in the [SfDataMatrixGenerator](#) is used to trigger when the input is an invalid string.

ASPX-CS

```
@using Syncfusion.Blazor.BarcodeGenerator
<SfDataMatrixGenerator Width="200px" Height="150px" Value="SYNCFUSION"
OnValidationFailed="@OnValidationFailed"></SfDataMatrixGenerator>
@code
{
public void OnValidationFailed(ValidationFailedEventArgs args)
{
}
}
```

Export in Blazor Barcode Component

Export

Barcode provides support to export its content as an image in the specified image type and downloads it in the browser.

ASPX-CS

```
@using Syncfusion.Blazor.BarcodeGenerator
<input type="button" value="Export" @onclick="@OnExport" />
<SfQRCodeGenerator Width="200px" Height="150px" Value="Syncfusion"
@ref="@QRcode" >
<QRCodeGeneratorDisplayText text="Text"></QRCodeGeneratorDisplayText>
</SfQRCodeGenerator>
@code{
SfQRCodeGenerator QRcode;
private void OnExport() {
QRcode.Export("fileName", BarcodeExportType.JPG);
}
}
```

ExportAsBaseImage

Barcode provides support to export as an image in the specified image type and returns it as base64 string.

ASPX-CS

```
@using Syncfusion.Blazor.BarcodeGenerator
<input type="button" value="Export" @onclick="@OnExport" />
<SfQRCodeGenerator Width="200px" Height="150px" Value="Syncfusion"
@ref="@QRcode">
<QRCodeGeneratorDisplayText text="Text"></QRCodeGeneratorDisplayText>
</SfQRCodeGenerator>
@code
{
    SfQRCodeGenerator QRcode;
    private async void OnExport()
    {
        await QRcode.ExportAsBase64Image(BarcodeExportType.JPG);
    }
}
```

Format is to specify the type/format of the exported file. You can export Barcode to the following formats:

* JPG.

* PNG.

Breadcrumb

Getting Started with Blazor Breadcrumb Component

This section briefly explains about how to include [Blazor Breadcrumb](#) component in your Blazor Server App and Blazor WebAssembly App using Visual Studio.

Prerequisites

- [System requirements for Blazor components](#)

Create a new Blazor App in Visual Studio

You can create **Blazor Server App** or **Blazor WebAssembly App** using Visual Studio in one of the following ways,

- [Create a Project using Microsoft Templates](#)
- [Create a Project using Syncfusion Blazor Extension](#)

Install Syncfusion Blazor Navigations NuGet in the App

Syncfusion Blazor components are available in nuget.org. In order to use Syncfusion Blazor components in the application, add reference to the corresponding NuGet. Refer to [NuGet packages topic](#) for available NuGet packages list with component details.

To add Blazor Breadcrumb component in the app, open the NuGet package manager in Visual Studio (*Tools* → *NuGet Package Manager* → *Manage NuGet Packages for Solution*), search for [Syncfusion.Blazor.Navigations](#) and then install it.

Add Style Sheet

Checkout the [Blazor Themes topic](#) to learn different ways to refer themes in Blazor application, and to have the expected appearance for Syncfusion Blazor components. Here, the theme is referred using [Static Web Assets](#).

To add theme to the app, open the NuGet package manager in Visual Studio (*Tools* → *NuGet Package Manager* → *Manage NuGet Packages for Solution*), search for [Syncfusion.Blazor.Themes](#) and then install it. Then, the theme style sheet from NuGet can be referred as follows,

Blazor Server App

- For .NET 6 app, add the Syncfusion bootstrap5 theme in the `element` of the `~/Pages/_Layout.cshtml` page.

ASPX-CS

```
<head>
....
<link href="_content/Syncfusion.Blazor.Themes/bootstrap5.css"
rel="stylesheet" />
</head>
```

- For .NET 5 and .NET 3.X app, add the Syncfusion bootstrap5 theme in the `element` of the `~/Pages/_Host.cshtml` page.

ASPX-CS

```
<head>
....
<link href="_content/Syncfusion.Blazor.Themes/bootstrap5.css"
rel="stylesheet" />
</head>
```

Blazor WebAssembly App

The theme style sheet from NuGet can be referred inside the `<head>` element of `wwwroot/index.html` file of client web app.

HTML

```
<head>
...
<link href="_content/Syncfusion.Blazor.Themes/bootstrap5.css"
rel="stylesheet" />
</head>
```

Add Script Reference

Checkout [Adding Script Reference topic](#) to learn different ways to add script reference in Blazor Application. In this getting started walk-through, the required scripts are referenced automatically via javascript script isolation approach.

Syncfusion recommends to reference scripts using [Static Web Assets](#), [CDN](#) and [CRG](#) by [disabling JavaScript isolation](#) for better loading performance of the Blazor application.

Register Syncfusion Blazor Service

Open `~/_Imports.razor` file and import the `Syncfusion.Blazor` namespace.

C#

```
@using Syncfusion.Blazor
```

Blazor Server App

Now, register the Syncfusion Blazor Service in the Blazor Server App.

- For .NET 6 app, open the **~/Program.cs** file and register the Syncfusion Blazor Service.

C#

```
using Microsoft.AspNetCore.Components;  
using Microsoft.AspNetCore.Components.Web;  
using Syncfusion.Blazor;  
var builder = WebApplication.CreateBuilder(args);  
// Add services to the container.  
builder.Services.AddRazorPages();  
builder.Services.AddServerSideBlazor();  
builder.Services.AddSyncfusionBlazor();  
var app = builder.Build();  
....
```

- For .NET 5 and .NET 3.X app, open the **~/Startup.cs** file and register the Syncfusion Blazor Service.

C#

```
using Syncfusion.Blazor;  
namespace BlazorApplication  
{  
    public class Startup  
    {  
        ...  
        public void ConfigureServices(IServiceCollection services)  
        {  
            services.AddRazorPages();  
            services.AddServerSideBlazor();  
            services.AddSyncfusionBlazor();  
        }  
        ...  
    }  
}
```

Blazor WebAssembly App

Open **~/Program.cs** file and register the Syncfusion Blazor Service in the client web app.

- For .NET 6 app,

C#

```
using Microsoft.AspNetCore.Components.Web;  
using Microsoft.AspNetCore.Components.WebAssembly.Hosting;  
using Syncfusion.Blazor;  
var builder = WebAssemblyHostBuilder.CreateDefault(args);
```

```
builder.RootComponents.Add<App>("#app");
builder.RootComponents.Add<HeadOutlet>("head::after");
builder.Services.AddScoped(sp => new HttpClient { BaseAddress = new
Uri(builder.HostEnvironment.BaseAddress) });
builder.Services.AddSyncfusionBlazor();
await builder.Build().RunAsync();
....
```

- For .NET 5 and .NET 3.X app

C#

```
using Syncfusion.Blazor;
namespace WebApplication1
{
    public class Program
    {
        public static async Task Main(string[] args)
        {
            ....
            builder.Services.AddSyncfusionBlazor();
            await builder.Build().RunAsync();
        }
    }
}
```

Add Syncfusion Blazor Breadcrumb component

- Open ~/_Imports.razor file or any razor page under the ~/Pages folder where the component is to be added and import the Syncfusion.Blazor.Navigations namespace.

C#

```
@using Syncfusion.Blazor
@using Syncfusion.Blazor.Navigations
```

- Now, add the Syncfusion Breadcrumb component in razor file. Here, the Breadcrumb component is added in the ~/Pages/Index.razor page under the ~/Pages folder.

C#

```
<SfBreadcrumb></SfBreadcrumb>
```

 / [breadcrumb](#) / bind-to-location?theme=bootstrap5

The Breadcrumb component will render based on the current URL, when the Breadcrumb items are not specified.

Add items to the Breadcrumb component

To render Breadcrumb component with items use [BreadcrumbItem](#) tag directive as like below code example.

C#

```
@using Syncfusion.Blazor.Navigations
<SfBreadcrumb>
<BreadcrumbItems>
<BreadcrumbItem IconCss="e-icons e-home"
Url="https://blazor.syncfusion.com/demos/" />
<BreadcrumbItem Text="Components"
Url="https://blazor.syncfusion.com/demos/datagrid/overview" />
<BreadcrumbItem Text="Navigations"
Url="https://blazor.syncfusion.com/demos/menu-bar/default-functionalities" />
<BreadcrumbItem Text="Breadcrumb" Url="./breadcrumb/default-
functionalities" />
</BreadcrumbItems>
</SfBreadcrumb>
```

Place list of [BreadcrumbItem](#) within [BreadcrumbItems](#) tag directive.

 / [Components](#) / [Navigations](#) / Breadcrumb

Enable or disable navigation

Breadcrumb component enables or disables built-in URL navigation based on the [EnableNavigation](#) property. By default, the navigation will be enabled when setting the [Url](#) property. To prevent Breadcrumb item navigation, set the [EnableNavigation](#) property as `false` in Breadcrumb.

See Also

- [Getting Started with Syncfusion Blazor for client side in .NET Core CLI](#)
- [Getting Started with Syncfusion Blazor for server side in Visual Studio](#)
- [Getting Started with Syncfusion Blazor for server side in .NET Core CLI](#)

Breadcrumb items in Blazor Breadcrumb component

The Breadcrumb supports to generate items based on the current URL by default. You can set the [BreadcrumbItem](#) tag directive or [Url](#) property to generate the items.

[BreadcrumbItem](#) has following properties for navigation and its customizations.

- [Url](#) - Sets the URL of the Breadcrumb item and it will navigate when click.
- [IconCss](#) - Sets CSS class string to include an icon for the Breadcrumb item.
- [Text](#) - Sets the text content of the Breadcrumb item.

ASPX-CS

```
@using Syncfusion.Blazor.Navigations
```

```
<SfBreadcrumb>
  <BreadcrumbItems>
    <BreadcrumbItem Text="Home"
      Url="https://blazor.syncfusion.com/demos/"></BreadcrumbItem>
    <BreadcrumbItem Text="Components"
      Url="https://blazor.syncfusion.com/demos/datagrid/overview"></BreadcrumbItem>
  >
    <BreadcrumbItem Text="Navigations"
      Url="https://blazor.syncfusion.com/demos/breadcrumb/default-
      functionalities"></BreadcrumbItem>
    <BreadcrumbItem Text="Breadcrumb" Url="./breadcrumb/default-
      functionalities"></BreadcrumbItem>
  </BreadcrumbItems>
</SfBreadcrumb>
```

[Home](#) / [Components](#) / [Navigations](#) / Breadcrumb

Items based on current URL

The Breadcrumb items can be generated based on the current URL of the page when the user does not specify the Breadcrumb items using the [BreadcrumbItem](#) tag directive. The following example shows the Breadcrumb items that are auto generated based on the current URL.

ASPX-CS

```
@using Syncfusion.Blazor.Navigations
<SfBreadcrumb></SfBreadcrumb>
```

[🏠](#) / [breadcrumb](#) / bind-to-location?theme=bootstrap5

This output screenshot shows the [Bind to Location](#) sample.

Absolute URL

You can generate the Breadcrumb items by providing the [Url](#) property in the component as like the below example.

ASPX-CS

```
@using Syncfusion.Blazor.Navigations
<SfBreadcrumb
  Url="https://blazor.syncfusion.com/demos/breadcrumb/navigation">
</SfBreadcrumb>
```

[🏠](#) / [demos](#) / [breadcrumb](#) / navigation

Icons in Blazor Breadcrumb component

The Breadcrumb component contains an icon, image, and SVG to provide a visual representation of an item.

Breadcrumb with font icon

To place the font icon on the Breadcrumb item, set the [IconCss](#) property to `e-icons` with the required icon's class name.

ASPX-CS

```
@using Syncfusion.Blazor.Navigations
<SfBreadcrumb>
  <BreadcrumbItems>
    <BreadcrumbItem IconCss="e-icons e-home"
      Url="https://blazor.syncfusion.com/demos/"></BreadcrumbItem>
    <BreadcrumbItem Text="Components"
      Url="https://blazor.syncfusion.com/demos/datagrid/overview"></BreadcrumbItem>
  >
  <BreadcrumbItem Text="Navigations"
    Url="https://blazor.syncfusion.com/demos/menu-bar/default-
    functionalities"></BreadcrumbItem>
  <BreadcrumbItem Text="Breadcrumb" Url="./breadcrumb/default-
    functionalities"></BreadcrumbItem>
</BreadcrumbItems>
</SfBreadcrumb>
```

 / [Components](#) / [Navigations](#) / Breadcrumb

By default, the icon is positioned to the left side of the item.


Breadcrumb with image

In the Breadcrumb component, we can add images to the items using [IconCss](#) property. In the following example, an image is added to the Breadcrumb item with height and width by using `e-image-home` class.

ASPX-CS

```
@using Syncfusion.Blazor.Navigations
<SfBreadcrumb>
  <BreadcrumbItems>
    <BreadcrumbItem IconCss="e-image-home"
      Url="https://blazor.syncfusion.com/demos/"></BreadcrumbItem>
    <BreadcrumbItem Text="Components"
      Url="https://blazor.syncfusion.com/demos/datagrid/overview"></BreadcrumbItem>
  >
  <BreadcrumbItem Text="Navigations"
    Url="https://blazor.syncfusion.com/demos/menu-bar/default-
    functionalities"></BreadcrumbItem>
  <BreadcrumbItem Text="Breadcrumb" Url="./breadcrumb/default-
    functionalities"></BreadcrumbItem>
</BreadcrumbItems>
</SfBreadcrumb>
<style>
```

```
.e-image-home {  
background-image: url(/home.png);  
height: 20px;  
width: 20px;  
}  
</style>
```

 / Components / Navigations / Breadcrumb

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 is added to the Breadcrumb item with height and width by using `e-svg-home` class.

ASPX-CS

```
@using Syncfusion.Blazor.Navigations  
<SfBreadcrumb>  
<BreadcrumbItems>  
<BreadcrumbItem IconCss="e-svg-home"  
Url="https://blazor.syncfusion.com/demos/"></BreadcrumbItem>  
<BreadcrumbItem Text="Components"  
Url="https://blazor.syncfusion.com/demos/datagrid/overview"></BreadcrumbItem  
>  
<BreadcrumbItem Text="Navigations"  
Url="https://blazor.syncfusion.com/demos/menu-bar/default-  
functionalities"></BreadcrumbItem>  
<BreadcrumbItem Text="Breadcrumb" Url="./breadcrumb/default-  
functionalities"></BreadcrumbItem>  
</BreadcrumbItems>  
</SfBreadcrumb>  
<style>  
.e-svg-home {  
background-image: url('/home.svg');  
height: 20px;  
width: 20px;  
}  
</style>
```

 / Components / Navigations / Breadcrumb

Icon only

Use [IconCss](#) property to display icon for an item.

In the following example, Breadcrumb items are demonstrated with only icons by providing the [IconCss](#) property.

ASPX-CS

```
@using Syncfusion.Blazor.Navigations
<SfBreadcrumb>
<BreadcrumbItems>
<BreadcrumbItem IconCss="e-icons e-home"></BreadcrumbItem>
<BreadcrumbItem IconCss="e-icons e-folder-open"></BreadcrumbItem>
<BreadcrumbItem IconCss="e-icons e-file-new"></BreadcrumbItem>
</BreadcrumbItems>
</SfBreadcrumb>
```



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 is provided only for the first item by setting the [IconCss](#) property.

ASPX-CS

```
@using Syncfusion.Blazor.Navigations
<SfBreadcrumb>
<BreadcrumbItems>
<BreadcrumbItem IconCss="e-icons e-home"
Url="https://blazor.syncfusion.com/demos/"></BreadcrumbItem>
<BreadcrumbItem Text="Components"
Url="https://blazor.syncfusion.com/demos/datagrid/overview"></BreadcrumbItem>
<BreadcrumbItem Text="Navigations"
Url="https://blazor.syncfusion.com/demos/menu-bar/default-
functionalities"></BreadcrumbItem>
<BreadcrumbItem Text="Breadcrumb" Url="./breadcrumb/default-
functionalities"></BreadcrumbItem>
</BreadcrumbItems>
</SfBreadcrumb>
```



Navigation in Blazor Breadcrumb component

By default, Breadcrumb items support navigation for relative or absolute URL. You can handle the custom navigation by setting [EnableNavigation](#) property as `false`.

Relative URL

You can specify relative URL in the [Url](#) property of the [BreadcrumbItem](#). In the following example, the items contain the relative URL.

ASPX-CS

```
@using Syncfusion.Blazor.Navigations
<SfBreadcrumb>
<BreadcrumbItems>
<BreadcrumbItem Text="Home" Url=".."></BreadcrumbItem>
<BreadcrumbItem Text="Breadcrumb" Url="./breadcrumb/getting-
started"></BreadcrumbItem>
<BreadcrumbItem Text="Default" Url=".."></BreadcrumbItem>
<BreadcrumbItem Text="Icons" Url="./breadcrumb/icons"></BreadcrumbItem>
<BreadcrumbItem Text="Navigation"
Url="./breadcrumb/navigation"></BreadcrumbItem>
<BreadcrumbItem Text="Overflow"
Url="./breadcrumb/overflow"></BreadcrumbItem>
</BreadcrumbItems>
</SfBreadcrumb>
```

[Home](#) / [Breadcrumb](#) / [Default](#) / [Icons](#) / [Navigation](#) / Overflow

Absolute URL

You can specify absolute URL in the [Url](#) property of the [BreadcrumbItem](#). In the following example, the items contains the absolute URL.

ASPX-CS

```
@using Syncfusion.Blazor.Navigations
<SfBreadcrumb>
<BreadcrumbItems>
<BreadcrumbItem Text="Home"
Url="https://blazor.syncfusion.com/documentation/breadcrumb/introduction"></
BreadcrumbItem>
<BreadcrumbItem Text="Getting"
Url="https://blazor.syncfusion.com/documentation/breadcrumb/getting-
started"></BreadcrumbItem>
<BreadcrumbItem Text="Icons"
Url="https://blazor.syncfusion.com/documentation/breadcrumb/icons"></BreadcrumbItem>
<BreadcrumbItem Text="Navigation"
Url="https://blazor.syncfusion.com/documentation/breadcrumb/navigation"></BreadcrumbItem>
<BreadcrumbItem Text="Overflow"
Url="https://blazor.syncfusion.com/documentation/breadcrumb/overflow"></BreadcrumbItem>
</BreadcrumbItems>
</SfBreadcrumb>
```

[Home](#) / [Getting](#) / [Icons](#) / [Navigation](#) / Overflow

Enable navigation for last Breadcrumb item

Breadcrumb enables the navigation for the last item by setting the [EnableActiveItemNavigation](#) property as `true`. In the following example, the navigation enabled for last Breadcrumb item.

ASPX-CS

```
@using Syncfusion.Blazor.Navigations
<SfBreadcrumb EnableActiveItemNavigation="true">
  <BreadcrumbItems>
    <BreadcrumbItem Text="Home"
      Url="https://blazor.syncfusion.com/documentation/breadcrumb/introduction"></BreadcrumbItem>
    <BreadcrumbItem Text="Getting"
      Url="https://blazor.syncfusion.com/documentation/breadcrumb/getting-started"></BreadcrumbItem>
    <BreadcrumbItem Text="Icons"
      Url="https://blazor.syncfusion.com/documentation/breadcrumb/icons"></BreadcrumbItem>
    <BreadcrumbItem Text="Navigation"
      Url="https://blazor.syncfusion.com/documentation/breadcrumb/navigation"></BreadcrumbItem>
    <BreadcrumbItem Text="Overflow"
      Url="https://blazor.syncfusion.com/documentation/breadcrumb/overflow"></BreadcrumbItem>
  </BreadcrumbItems>
</SfBreadcrumb>
```

[Home](#) / [Getting](#) / [Icons](#) / [Navigation](#) / [Overflow](#)

Overflow mode in Blazor Breadcrumb component

In the Breadcrumb component, [MaxItems](#) and [OverflowMode](#) properties were used to limit the number of Breadcrumb items to be displayed.

The following overflow modes are available in the Breadcrumb component.

[Default](#) - Specified [MaxItems](#) count will be visible and the remaining items will be hidden. While clicking on the previous item, the hidden item will become visible.

[Collapsed](#) - Only the first and last items will be visible, and the remaining items will be hidden with the collapsed icon. When the collapsed icon is clicked, all items will become visible.

In the following example, the [MaxItems](#) is set as 3 with [OverflowMode](#) as [Default](#). To prevent Breadcrumb item navigation, the [EnableNavigation](#) property is set as `false`.

ASPX-CS

```
@using Syncfusion.Blazor.Navigations
<SfBreadcrumb MaxItems="3" EnableNavigation="false">
  <BreadcrumbItems>
    <BreadcrumbItem Text="Home"
      Url="https://blazor.syncfusion.com/documentation/breadcrumb/introduction"></BreadcrumbItem>
```

```

<BreadcrumbItem Text="Getting"
Url="https://blazor.syncfusion.com/documentation/breadcrumb/getting-
started"></BreadcrumbItem>
<BreadcrumbItem Text="Icons"
Url="https://blazor.syncfusion.com/documentation/breadcrumb/icons"></BreadcrumbItem>
<BreadcrumbItem Text="Navigation"
Url="https://blazor.syncfusion.com/documentation/breadcrumb/navigation"></BreadcrumbItem>
<BreadcrumbItem Text="Overflow"
Url="https://blazor.syncfusion.com/documentation/breadcrumb/overflow"></BreadcrumbItem>
</BreadcrumbItems>
<BreadcrumbTemplates>
<SeparatorTemplate>
<span class="e-icons e-arrow"></span>
</SeparatorTemplate>
</BreadcrumbTemplates>
</SfBreadcrumb>
<style>
.e-arrow:before {
content: "\e748";
}
</style>

```

[Home](#) > > [Navigation](#) > Overflow

Templates in Blazor Breadcrumb component

Blazor has templated components which accepts one or more UI segments as input that can be rendered as part of the component during component rendering. Breadcrumb is a templated blazor component, that allow you to customize various part of the UI using template parameters. It allows you to render custom components or content based on your own logic.

The available template options in Breadcrumb are as follows,

[Item template](#) - Used to customize the items.

[Separator template](#)- Used to customize the separators.

Template context

The templates used by Breadcrumb are of type `RenderFragment` and they will be passed with parameters. You can access the parameters passed to the templates using implicit parameter named `context`. You can also change this implicit parameter name using `Context` attribute.

Item template

In the following example, shopping cart details are used as Breadcrumb items and each item is rendered as Chip component using [ItemTemplate](#) tag directive. You can get the current item in `context` property.

ASPX-CS

```

@using Syncfusion.Blazor.Navigations
@using Syncfusion.Blazor.Buttons
<SfBreadcrumb class="e-breadcrumb-chips">
<BreadcrumbItems>

```

```

<BreadcrumbItem Text="Cart"></BreadcrumbItem>
<BreadcrumbItem Text="Billing"></BreadcrumbItem>
<BreadcrumbItem Text="Shipping"></BreadcrumbItem>
<BreadcrumbItem Text="Payment"></BreadcrumbItem>
</BreadcrumbItems>
<BreadcrumbTemplates>
<ItemTemplate>
<SfChip>
<ChipItems>
<ChipItem Text="@context.Text" Enabled="true"></ChipItem>
</ChipItems>
</SfChip>
</ItemTemplate>
</BreadcrumbTemplates>
</SfBreadcrumb>

```

Cart / Billing / Shipping / Payment

Separator template

In the following example, the separators are customized with icons using [SeparatorTemplate](#) tag directive. You can get the previous and next item in `context` property.

ASPX-CS

```

@using Syncfusion.Blazor.Navigations
<SfBreadcrumb>
<BreadcrumbItems>
<BreadcrumbItem Text="Cart"></BreadcrumbItem>
<BreadcrumbItem Text="Billing"></BreadcrumbItem>
<BreadcrumbItem Text="Shipping"></BreadcrumbItem>
<BreadcrumbItem Text="Payment"></BreadcrumbItem>
</BreadcrumbItems>
<BreadcrumbTemplates>
<SeparatorTemplate>
<span class="e-icons e-bullet-arrow"></span>
</SeparatorTemplate>
</BreadcrumbTemplates>
</SfBreadcrumb>
<style>
.e-bullet-arrow::before {
content: '\e763';
font-size: 12px;
}
</style>

```

Cart ➤ Billing ➤ Shipping ➤ Payment

Bullet Chart

Getting Started with Blazor Bullet Chart Component

This section briefly explains how to include a Bullet Chart component in the Blazor server-side application. Refer to [Getting Started with Syncfusion Blazor for server-side in Visual Studio](#) page for introduction and configuring common specifications.

Importing Syncfusion Blazor Bullet Chart component in the application

1. Install [Syncfusion.Blazor](#) NuGet package to the application by using the [NuGet Package Manager](#).
2. Add the client-side resources through [CDN](#) or from [NuGet](#) package in the **HEAD** element of the `~/Pages/_Host.cshtml` page.

HTML

```
<head>
<link href="_content/Syncfusion.Blazor.Themes/bootstrap4.css"
rel="stylesheet" />
<!--CDN-->
@*<link href="https://cdn.syncfusion.com/blazor/{{ site.blazorversion
}}/styles/bootstrap4.css" rel="stylesheet" />*@
</head>
```

For Internet Explorer 11, kindly refer to the polyfills. Refer to the [documentation](#) for more information.

HTML

```
<head>
<link href="https://cdn.syncfusion.com/blazor/{{ site.blazorversion
}}/styles/bootstrap4.css" rel="stylesheet" />
<script
src="https://github.com/Daddoon/Blazor.Polyfill/releases/download/3.0.1/blaz
or.polyfill.min.js"></script>
</head>
```

Adding component package to the application

Open the `~/_Imports.razor` file and include the `Syncfusion.Blazor.Charts` namespace.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
```

Adding SyncfusionBlazor service in Startup.cs

Open the **Startup.cs** file and add services required by Syncfusion components using **services.AddSyncfusionBlazor()** method. Add this method in the **ConfigureServices** function as follows.

CSHARP

```
using Syncfusion.Blazor;
namespace BlazorApplication
{
public class Startup
```



```
{
    ....
    ....
    public void ConfigureServices(IServiceCollection services)
    {
        ....
        ....
        services.AddSyncfusionBlazor();
    }
}
```

To enable the custom client-side source loading from CRG or CDN, please refer to the section about [custom resources in Blazor application](#).

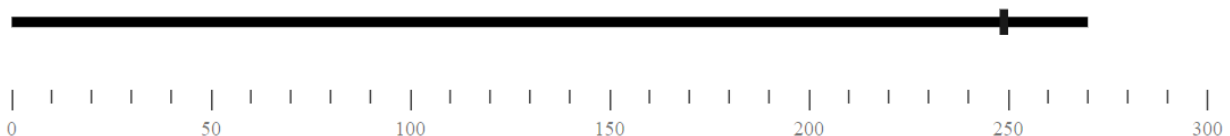
Adding Bullet Chart component

To initialize the Bullet Chart component, add the below code to the **Index.razor** view page under **~/Pages** folder. In a new application, if **Index.razor** page has any default content template, then those content can be completely removed, and following code can be added.

ASPX-CS

```
@page "/"
<SfBulletChart DataSource="@BulletChartData" ValueField="FieldValue"
TargetField="TargetValue" Minimum="0" Maximum="300" Interval="50">
</SfBulletChart>
@code{
    public class ChartData
    {
        public double FieldValue { get; set; }
        public double TargetValue { get; set; }
    }
    public List<ChartData> BulletChartData = new List<ChartData>
    {
        new ChartData { FieldValue = 270, TargetValue = 250 }
    };
}
```

On successful compilation of the application, the Syncfusion Blazor Bullet Chart component will render in the web browser as follows.



Adding Title

Add a title by using the [Title](#) property in the Bullet Chart, to provide quick information to the user about the data plotted in the component.

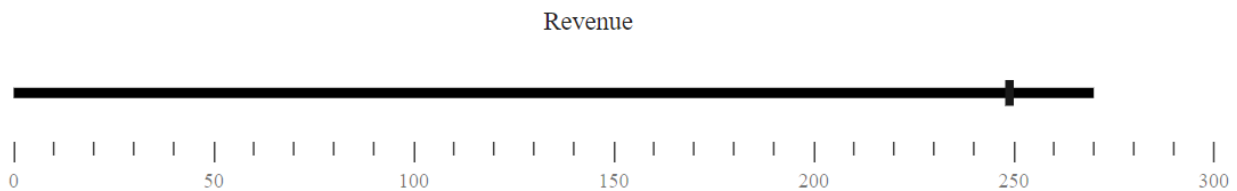
ASPX-CS

```
@page "/"
```

```

<SfBulletChart DataSource="@BulletChartData" ValueField="FieldValue"
TargetField="TargetValue" Minimum="0" Maximum="300" Interval="50"
Title="Revenue">
</SfBulletChart>
@code{
public class ChartData
{
public double FieldValue { get; set; }
public double TargetValue { get; set; }
}
public List<ChartData> BulletChartData = new List<ChartData>
{
new ChartData { FieldValue = 270, TargetValue = 250 }
};
}

```



Adding Ranges

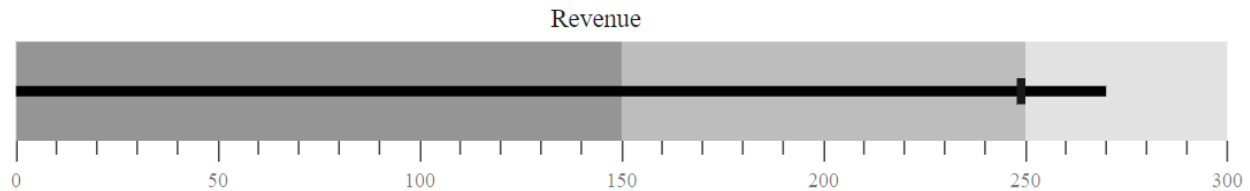
Add ranges by using the [BulletChartRangeCollection](#) to measure the qualitative state by observing the distance between each range.

ASPX-CS

```

@page "/"
<SfBulletChart DataSource="@BulletChartData" ValueField="FieldValue"
TargetField="TargetValue" Minimum="0" Maximum="300" Interval="50"
Title="Revenue">
<BulletChartRangeCollection>
<BulletChartRange End=150> </BulletChartRange>
<BulletChartRange End=250></BulletChartRange>
<BulletChartRange End=300></BulletChartRange>
</BulletChartRangeCollection>
</SfBulletChart>
@code{
public class ChartData
{
public double FieldValue { get; set; }
public double TargetValue { get; set; }
}
public List<ChartData> BulletChartData = new List<ChartData>
{
new ChartData { FieldValue = 270, TargetValue = 250 }
};
}

```



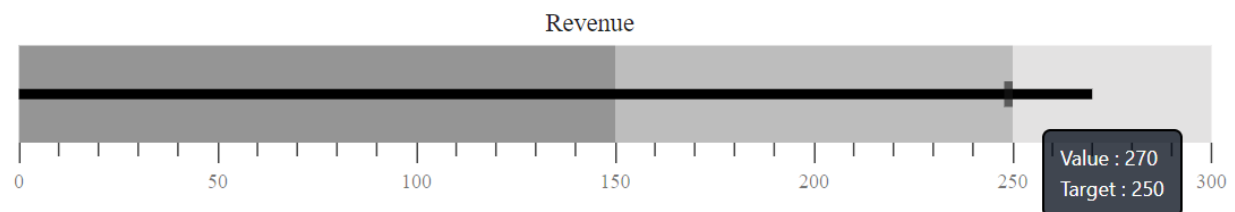
Adding Tooltip

Use the tooltip to show the measured values by setting the [Enable](#) property to **true** in the [BulletChartTooltip](#).

ASPX-CS

```
@page "/"
<SfBulletChart DataSource="@BulletChartData" ValueField="FieldValue"
TargetField="TargetValue" Minimum="0" Maximum="300" Interval="50"
Title="Revenue">
  <BulletChartTooltip TValue="ChartData" Enable="true"></BulletChartTooltip>
  <BulletChartRangeCollection>
    <BulletChartRange End=150> </BulletChartRange>
    <BulletChartRange End=250></BulletChartRange>
    <BulletChartRange End=300></BulletChartRange>
  </BulletChartRangeCollection>
</SfBulletChart>

@code{
public class ChartData
{
    public double FieldValue { get; set; }
    public double TargetValue { get; set; }
}
public List<ChartData> BulletChartData = new List<ChartData>
{
    new ChartData { FieldValue = 270, TargetValue = 250 }
};
}
```



Bullet Chart Dimensions in Blazor 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.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<div style="width:650px; height:100px;">
```

```

<SfBulletChart DataSource="@BulletChartData" ValueField="FieldValue"
TargetField="Target" Minimum="0" Maximum="300" Interval="50"
Title="Revenue">
  <BulletChartTooltip TValue="ChartData" Enable="true"></BulletChartTooltip>
  <BulletChartRangeCollection>
    <BulletChartRange End=150> </BulletChartRange>
    <BulletChartRange End=250></BulletChartRange>
    <BulletChartRange End=300></BulletChartRange>
  </BulletChartRangeCollection>
</SfBulletChart>
</div>
@code{
public class ChartData
{
public double FieldValue { get; set; }
public double Target { get; set; }
}
public List<ChartData> BulletChartData = new List<ChartData>
{
new ChartData { FieldValue = 270, Target = 250 }
};
}

```

Size for Bullet Chart

The [Width](#) and the [Height](#) properties are used to adjust the size of the Bullet Chart. Both the pixel and the percentage values are accepted. If the value is expressed as a percentage, the dimension of the Bullet Chart is determined by its container.

If the size is not specified, the Bullet Chart will be rendered with a height of **126px** and a width of the window.

ASPX-CS

```

@using Syncfusion.Blazor.Charts
<div style="width:1000px; height:150px;">
  <SfBulletChart DataSource="@BulletChartData" Height="70%" Width="50%"
ValueField="FieldValue" TargetField="Target" Minimum="0" Maximum="300"
Interval="50" Title="Revenue">
    <BulletChartTooltip TValue="ChartData" Enable="true"></BulletChartTooltip>
    <BulletChartRangeCollection>
      <BulletChartRange End=150> </BulletChartRange>
      <BulletChartRange End=250></BulletChartRange>
      <BulletChartRange End=300></BulletChartRange>
    </BulletChartRangeCollection>
  </SfBulletChart>
</div>

```

Refer to the [code block](#) to know about the property value of **BulletChartData**.

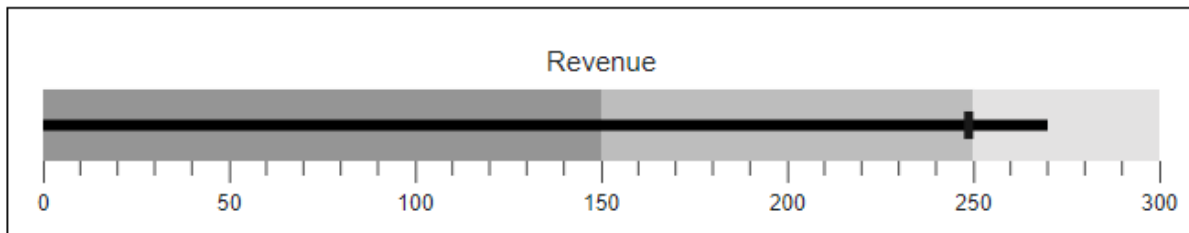
Margin

The [BulletChartMargin](#) is used to customize the bottom, the left, the right, and the top margins of the Bullet Chart.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<div style="width:650px; height:100px;">
<SfBulletChart DataSource="@BulletChartData" ValueField="FieldValue"
TargetField="Target" Minimum="0" Maximum="300" Interval="50"
Title="Revenue">
<BulletChartMargin Bottom="20" Left="20" Right="20"
Top="20"></BulletChartMargin>
<BulletChartBorder Color="#000000" Width="2"></BulletChartBorder>
<BulletChartTooltip TValue="ChartData" Enable="true"></BulletChartTooltip>
<BulletChartRangeCollection>
<BulletChartRange End=150></BulletChartRange>
<BulletChartRange End=250></BulletChartRange>
<BulletChartRange End=300></BulletChartRange>
</BulletChartRangeCollection>
</SfBulletChart>
</div>
```

Refer to the [code block](#) to know about the property value of **BulletChartData**.



Axis Customization in Blazor Bullet Chart Component

MajorTickLines and MinorTickLines Customization

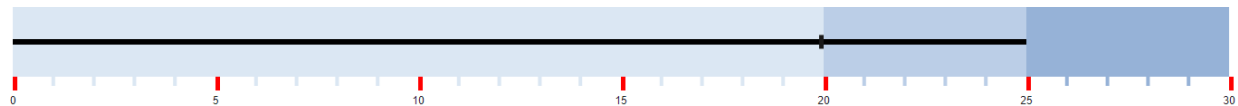
The following properties can be used to customize [MajorTicklines](#) and [MinorTicklines](#).

- **Width** - Specifies the width of ticklines.
- **Height** - Specifies the height of ticklines.
- **Color** - Specifies the color of ticklines.
- **EnableRangeColor** - Specifies the color of ticklines and represents the color from corresponding range colors.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfBulletChart DataSource="@BulletChartData" ValueField="FieldValue"
TargetField="Target" Minimum="0" Maximum="30" Interval="5">
<BulletChartMinorTickLines Width="4" Height="10"
EnableRangeColor="true"></BulletChartMinorTickLines>
<BulletChartMajorTickLines Width="5" Height="15"
Color="red"></BulletChartMajorTickLines>
<BulletChartRangeCollection>
<BulletChartRange End=20 Color="#DBE7F3"></BulletChartRange>
<BulletChartRange End=25 Color="#BCEE7"></BulletChartRange>
<BulletChartRange End=30 Color="#96B2D7"></BulletChartRange>
</BulletChartRangeCollection>
</SfBulletChart>
```

```
@code{
public class ChartData
{
    public double FieldValue { get; set; }
    public double Target { get; set; }
}
public List<ChartData> BulletChartData = new List<ChartData>
{
    new ChartData { FieldValue = 25, Target = 20 }
};
}
```



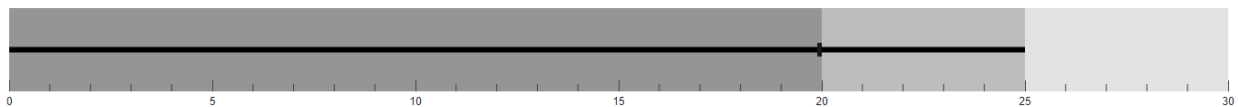
Tick Placement

The major and the minor ticks can be placed [Inside](#) or [Outside](#) the ranges using the [TickPosition](#) property.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfBulletChart DataSource="@BulletChartData"
    TickPosition="TickPosition.Inside" ValueField="FieldValue"
    TargetField="Target" Minimum="0" Maximum="30" Interval="5">
    <BulletChartRangeCollection>
    <BulletChartRange End=20> </BulletChartRange>
    <BulletChartRange End=25></BulletChartRange>
    <BulletChartRange End=30></BulletChartRange>
    </BulletChartRangeCollection>
</SfBulletChart>
```

Refer to the [code block](#) to know about the property value of **BulletChartData**.



Label Format

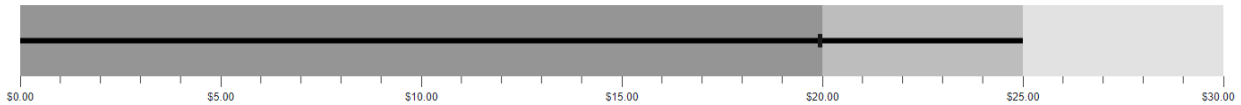
The axis labels support all the globalize formats and can be changed using the [Format](#) property. The following code shows the axis label in the currency format.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfBulletChart DataSource="@BulletChartData" Format="C"
    ValueField="FieldValue" TargetField="Target" Minimum="0" Maximum="30"
    Interval="5">
    <BulletChartRangeCollection>
    <BulletChartRange End=20> </BulletChartRange>
    <BulletChartRange End=25></BulletChartRange>
    <BulletChartRange End=30></BulletChartRange>
    </BulletChartRangeCollection>
```

```
</SfBulletChart>
```

Refer to the [code block](#) to know about the property value of **BulletChartData**.



The following table describes the result of applying some commonly used formats to numeric axis values.

<!-- markdownlint-disable MD033 -->

Label Value	Label Format property value	Result	Description
1000	n1	1000.0	The result is rounded to 1 decimal place.
1000	n2	1000.00	The result is rounded to 2 decimal places.
1000	n3	1000.000	The result is rounded to 3 decimal places.
0.01	p1	1.0%	The result is converted to percentage with 1 decimal place.
0.01	p2	1.00%	The result is converted to percentage with 2 decimal places.
0.01	p3	1.000%	The result is converted to percentage with 3 decimal places.
1000	c1	\$1000.0	The currency symbol is appended to the result and it is rounded to 1 decimal place.
1000	c2	\$1000.00	The currency symbol is appended to the result and it is rounded to 2 decimal places.

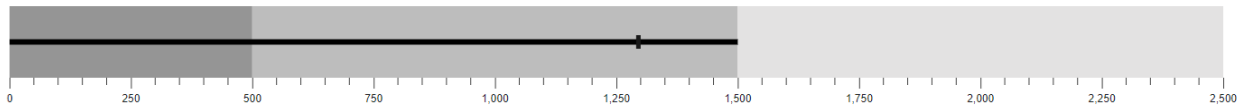
Grouping Separator

To separate the groups of thousands, set the [EnableGroupSeparator](#) property to **true**, and specify the numeric axis label by setting the [Format](#) property to **N0**.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfBulletChart DataSource="@BulletChartData" EnableGroupSeparator="true"
Format="N0" ValueField="FieldValue" TargetField="Target" Minimum="0"
Maximum="2500" Interval="250">
  <BulletChartRangeCollection>
    <BulletChartRange End=500> </BulletChartRange>
    <BulletChartRange End=1500></BulletChartRange>
    <BulletChartRange End=2500></BulletChartRange>
  </BulletChartRangeCollection>
</SfBulletChart>
@code{
public class ChartData
```

```
{
    public double FieldValue { get; set; }
    public double Target { get; set; }
}
public List<ChartData> BulletChartData = new List<ChartData>
{
    new ChartData { FieldValue = 1500, Target = 1300 }
};
}
```



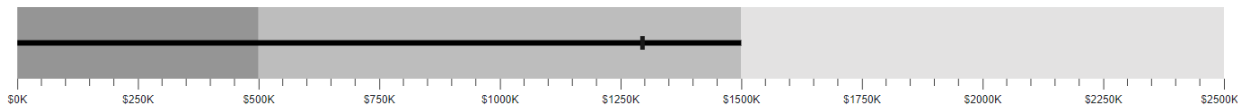
Custom Label Format

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.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfBulletChart DataSource="@BulletChartData" LabelFormat="${value}K"
ValueField="FieldValue" TargetField="Target" Minimum="0" Maximum="2500"
Interval="250">
    <BulletChartRangeCollection>
        <BulletChartRange End=500> </BulletChartRange>
        <BulletChartRange End=1500> </BulletChartRange>
        <BulletChartRange End=2500> </BulletChartRange>
    </BulletChartRangeCollection>
</SfBulletChart>
```

Refer to the [code block](#) to know about the property value of **BulletChartData**.



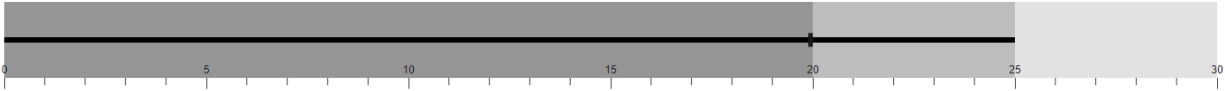
Label Placement

Label can be placed [Inside](#) or [Outside](#) of the ranges using the [LabelPosition](#) property.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfBulletChart DataSource="@BulletChartData"
LabelPosition="LabelsPlacement.Inside" ValueField="FieldValue"
TargetField="Target" Minimum="0" Maximum="30" Interval="5">
    <BulletChartRangeCollection>
        <BulletChartRange End=20> </BulletChartRange>
        <BulletChartRange End=25> </BulletChartRange>
        <BulletChartRange End=30> </BulletChartRange>
    </BulletChartRangeCollection>
</SfBulletChart>
```

Refer to the [code block](#) to know about the property value of **BulletChartData**.



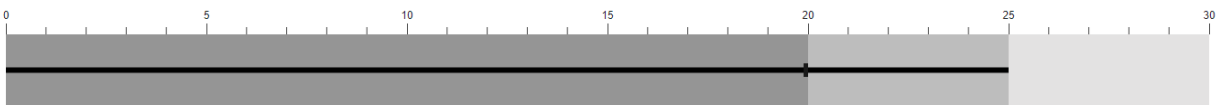
Opposed Position

To place an axis opposite to its original position, set the [OpposedPosition](#) property to **true**.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfBulletChart DataSource="@BulletChartData" OpposedPosition="true"
ValueField="FieldValue" TargetField="Target" Minimum="0" Maximum="30"
Interval="5">
  <BulletChartRangeCollection>
    <BulletChartRange End=20></BulletChartRange>
    <BulletChartRange End=25></BulletChartRange>
    <BulletChartRange End=30></BulletChartRange>
  </BulletChartRangeCollection>
</SfBulletChart>
```

Refer to the [code block](#) to know about the property value of **BulletChartData**.

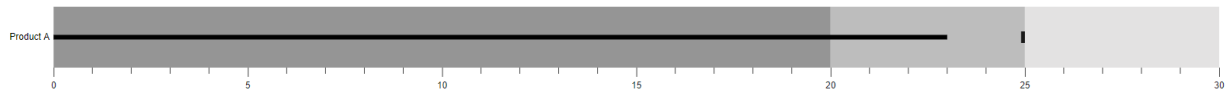


Category Label

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.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfBulletChart DataSource="@BulletChartData" CategoryField="Category"
ValueField="FieldValue" TargetField="Target" Minimum="0" Maximum="30"
Interval="5">
  <BulletChartRangeCollection>
    <BulletChartRange End=20></BulletChartRange>
    <BulletChartRange End=25></BulletChartRange>
    <BulletChartRange End=30></BulletChartRange>
  </BulletChartRangeCollection>
</SfBulletChart>
@code{
public class ChartData
{
    public double FieldValue { get; set; }
    public double Target { get; set; }
    public string Category { get; set; }
}
public List<ChartData> BulletChartData = new List<ChartData>
{
    new ChartData { FieldValue = 23, Target = 25, Category="Product A" }
};
}
```



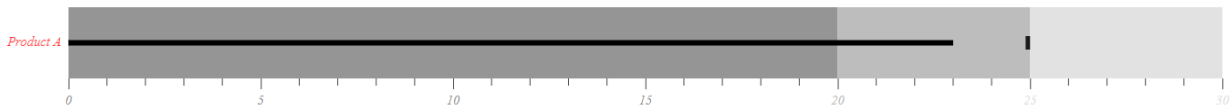
Axis Label and Category Label Customization

The label color, opacity, font size, font family, font weight, and font style can be customized by using the [BulletChartCategoryLabelStyle](#) setting for category and the [BulletChartLabelStyle](#) setting for axis label. The [EnableRangeColor](#) property specifies the color of the axis label and represents the color from the corresponding range colors.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfBulletChart DataSource="@BulletChartData" CategoryField="Category"
ValueField="FieldValue" TargetField="Target" Minimum="0" Maximum="30"
Interval="5">
  <BulletChartCategoryLabelStyle Color="red" Opacity="0.8" Size="15px"
FontStyle="italic"></BulletChartCategoryLabelStyle>
  <BulletChartLabelStyle Color="red" Opacity="1" Size="15px"
FontStyle="italic" EnableRangeColor="true"></BulletChartLabelStyle>
  <BulletChartRangeCollection>
    <BulletChartRange End=20 Color="#959595"></BulletChartRange>
    <BulletChartRange End=25 Color="#BDBDBD"></BulletChartRange>
    <BulletChartRange End=30 Color="#E3E2E2"></BulletChartRange>
  </BulletChartRangeCollection>
</SfBulletChart>
```

Refer to the [code block](#) to know about the property value of **BulletChartData**.



Working with Data in Blazor Bullet Chart Component

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.

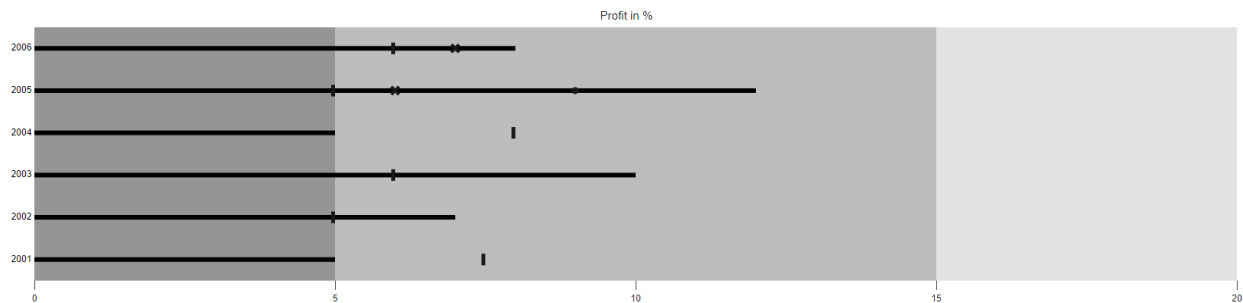
ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfBulletChart DataSource="@InputData" ValueField="FieldValue"
TargetField="ComparativeMeasureValue" CategoryField="Category" Height="400"
Minimum="0" Maximum="20" Interval="5" Title="Profit in %">
  <BulletChartMinorTickLines Width="0"></BulletChartMinorTickLines>
  <BulletChartRangeCollection>
    <BulletChartRange End=5> </BulletChartRange>
    <BulletChartRange End=15></BulletChartRange>
    <BulletChartRange End=20></BulletChartRange>
  </BulletChartRangeCollection>
</SfBulletChart>
@code{
public class BulletChartData
{
public double FieldValue { get; set; }
}
```

```

public double[] ComparativeMeasureValue { get; set; }
public string Category { get; set; }
}
public List<BulletChartData> InputData = new List<BulletChartData>
{
    new BulletChartData { FieldValue = 5, ComparativeMeasureValue = new double[]
    { 7.5 }, Category = "2001" },
    new BulletChartData { FieldValue = 7, ComparativeMeasureValue = new double[]
    { 5 }, Category = "2002" },
    new BulletChartData { FieldValue = 10, ComparativeMeasureValue = new
    double[] { 6 }, Category = "2003" },
    new BulletChartData { FieldValue = 5, ComparativeMeasureValue = new double[]
    { 8 }, Category = "2004" },
    new BulletChartData { FieldValue = 12, ComparativeMeasureValue = new
    double[] { 5, 6, 9 }, Category = "2005" },
    new BulletChartData { FieldValue = 8, ComparativeMeasureValue = new double[]
    { 6, 7 }, Category = "2006" }
};
}

```



Ranges in Blazor 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. The minimum value of a quantitative scale is considered the starting point of the first range or the previous range end point.

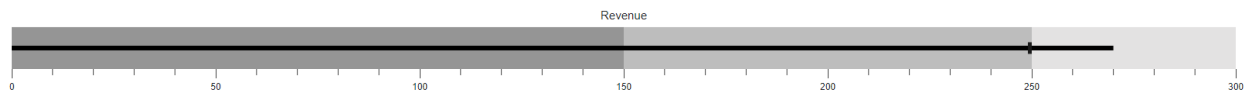
ASPX-CS

```

@using Syncfusion.Blazor.Charts
<SfBulletChart DataSource="@BulletChartData" ValueField="FieldValue"
TargetField="TargetValue" Minimum="0" Maximum="300" Interval="50"
Title="Revenue">
    <BulletChartRangeCollection>
        <BulletChartRange End=150> </BulletChartRange>
        <BulletChartRange End=250> </BulletChartRange>
        <BulletChartRange End=300> </BulletChartRange>
    </BulletChartRangeCollection>
</SfBulletChart>
@code{
    public class ChartData
    {
        public double FieldValue { get; set; }
        public double TargetValue { get; set; }
    }
    public List<ChartData> BulletChartData = new List<ChartData>
    {

```

```
new ChartData { FieldValue = 270, TargetValue = 250 }
};
}
```

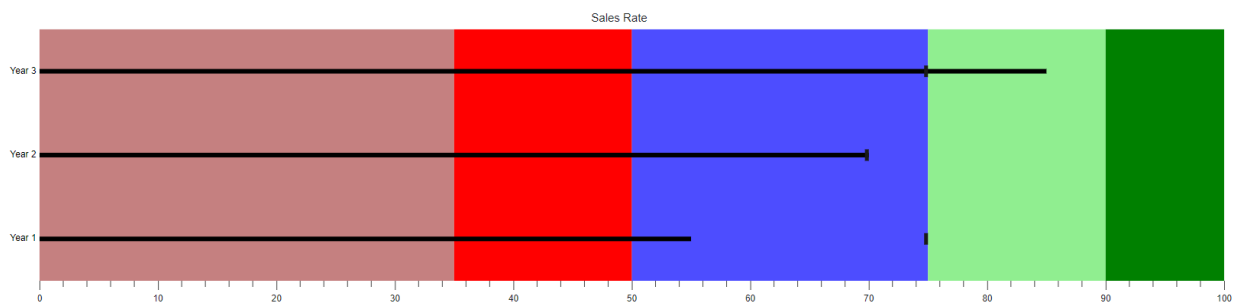


Color Customization

Enhance the readability of ranges with color and opacity. It can be applied using the [Color](#) and [Opacity](#) properties respectively.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfBulletChart DataSource="@BulletChartData" CategoryField="CategoryValue"
ValueField="FieldValue" TargetField="TargetValue" Minimum="0" Maximum="100"
Interval="10" Title="Sales Rate" Height="400">
  <BulletChartRangeCollection>
    <BulletChartRange End=35 Color="darkred" Opacity="0.5"></BulletChartRange>
    <BulletChartRange End=50 Color="red" Opacity="1"></BulletChartRange>
    <BulletChartRange End=75 Color="blue" Opacity="0.7"></BulletChartRange>
    <BulletChartRange End=90 Color="lightgreen" Opacity="1"></BulletChartRange>
    <BulletChartRange End=100 Color="green" Opacity="1"></BulletChartRange>
  </BulletChartRangeCollection>
</SfBulletChart>
@code{
public class ChartData
{
    public double FieldValue { get; set; }
    public double TargetValue { get; set; }
    public string CategoryValue { get; set; }
}
public List<ChartData> BulletChartData = new List<ChartData>
{
    new ChartData { FieldValue = 55, TargetValue = 75, CategoryValue = "Year 1" },
    new ChartData { FieldValue = 70, TargetValue = 70, CategoryValue = "Year 2" },
    new ChartData { FieldValue = 85, TargetValue = 75, CategoryValue = "Year 3" }
};
}
```

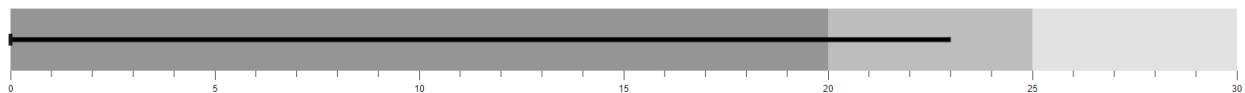


Actual Bar in Blazor Bullet Chart Component

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.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfBulletChart DataSource="@BulletChartData" ValueField="FieldValue"
Minimum="0" Maximum="30" Interval="5">
  <BulletChartRangeCollection>
    <BulletChartRange End=20> </BulletChartRange>
    <BulletChartRange End=25></BulletChartRange>
    <BulletChartRange End=30></BulletChartRange>
  </BulletChartRangeCollection>
</SfBulletChart>
@code{
public class ChartData
{
public double FieldValue { get; set; }
}
public List<ChartData> BulletChartData = new List<ChartData>
{
new ChartData { FieldValue = 23}
};
}
```

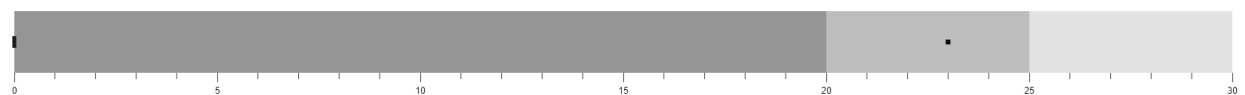


Types of Actual Bar

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](#).

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfBulletChart DataSource="@BulletChartData" Type="FeatureType.Dot"
ValueField="FieldValue" Minimum="0" Maximum="30" Interval="5">
  <BulletChartRangeCollection>
    <BulletChartRange End=20> </BulletChartRange>
    <BulletChartRange End=25></BulletChartRange>
    <BulletChartRange End=30></BulletChartRange>
  </BulletChartRangeCollection>
</SfBulletChart>
```



Refer to the [code block](#) to know more about the property value of the **BulletChartData**.

Actual Bar Customization

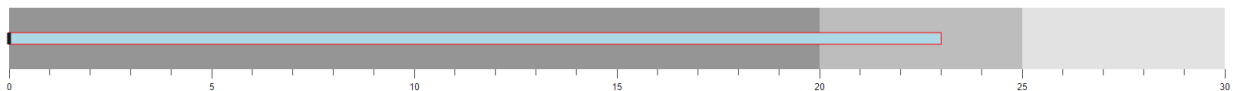
The following properties will be used to customize the actual bar.

- [ValueFill](#) - Specifies the fill color of the actual bar.
- [ValueHeight](#) - Specifies the width of the actual bar.
- [BulletChartValueBorder](#) - Specifies the border color and the border width of the actual bar.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfBulletChart DataSource="@BulletChartData" ValueFill="lightblue"
ValueHeight="15" ValueField="FieldValue" Minimum="0" Maximum="30"
Interval="5">
  <BulletChartValueBorder Color="red" Width="1"></BulletChartValueBorder>
  <BulletChartRangeCollection>
    <BulletChartRange End=20></BulletChartRange>
    <BulletChartRange End=25></BulletChartRange>
    <BulletChartRange End=30></BulletChartRange>
  </BulletChartRangeCollection>
</SfBulletChart>
```

Refer to the [code block](#) to know more about the property value of the **BulletChartData**.



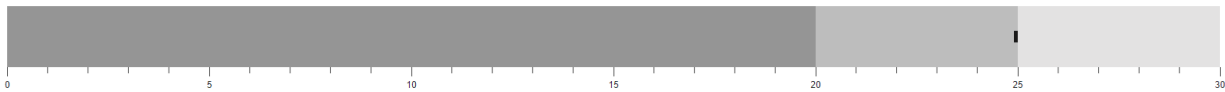
Target Bar in Blazor Bullet Chart Component

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.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfBulletChart DataSource="@BulletChartData" TargetField="Target"
Minimum="0" Maximum="30" Interval="5">
  <BulletChartRangeCollection>
    <BulletChartRange End=20> </BulletChartRange>
    <BulletChartRange End=25></BulletChartRange>
    <BulletChartRange End=30></BulletChartRange>
  </BulletChartRangeCollection>
</SfBulletChart>

@code{
public class ChartData
{
    public double Target { get; set; }
}
public List<ChartData> BulletChartData = new List<ChartData>
{
    new ChartData { Target = 25 }
};
}
```



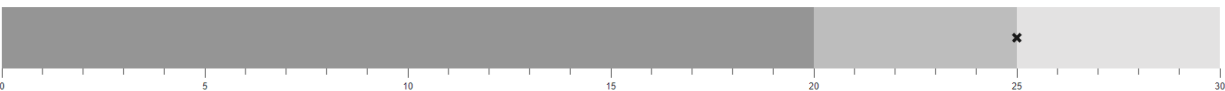
Types of Target Bar

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](#).

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfBulletChart DataSource="@BulletChartData" TargetField="Target"
Minimum="0" Maximum="30" Interval="5" TargetTypes="new List<TargetType>() {
TargetType.Cross }">
<BulletChartRangeCollection>
<BulletChartRange End=20> </BulletChartRange>
<BulletChartRange End=25></BulletChartRange>
<BulletChartRange End=30></BulletChartRange>
</BulletChartRangeCollection>
</SfBulletChart>
```

Refer to the [code block](#) to know about the property value of the **BulletChartData**.



Target Bar Customization

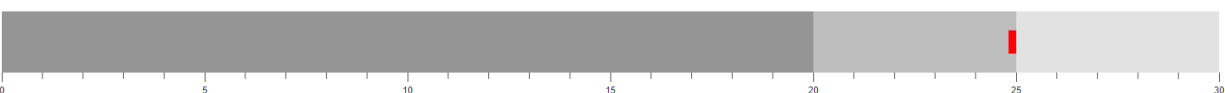
The following properties can be used to customize the Target Bar.

- [TargetColor](#) - Specifies the fill color of Target Bar.
- [TargetWidth](#) - Specifies the width of Target Bar.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfBulletChart DataSource="@BulletChartData" TargetField="Target"
Minimum="0" Maximum="30" Interval="5" TargetColor="red" TargetWidth="10">
<BulletChartRangeCollection>
<BulletChartRange End=20> </BulletChartRange>
<BulletChartRange End=25></BulletChartRange>
<BulletChartRange End=30></BulletChartRange>
</BulletChartRangeCollection>
</SfBulletChart>
```

Refer to the [code block](#) to know about the property value of the **BulletChartData**.



Title and Subtitle in Blazor Bullet Chart Component

Title

The title of the Bullet Chart displays the information about the data plotted by specifying it in the [Title](#) property.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfBulletChart DataSource="@BulletChartData" Title="Sales Rate"
ValueField="FieldValue" TargetField="TargetValue" Minimum="0" Maximum="100"
Interval="20">
<BulletChartRangeCollection>
<BulletChartRange End=35> </BulletChartRange>
<BulletChartRange End=50></BulletChartRange>
<BulletChartRange End=100></BulletChartRange>
</BulletChartRangeCollection>
</SfBulletChart>
@code{
public class ChartData
{
public double FieldValue { get; set; }
public double TargetValue { get; set; }
}
public List<ChartData> BulletChartData = new List<ChartData>
{
new ChartData { FieldValue = 55, TargetValue = 75 }
};
}
```



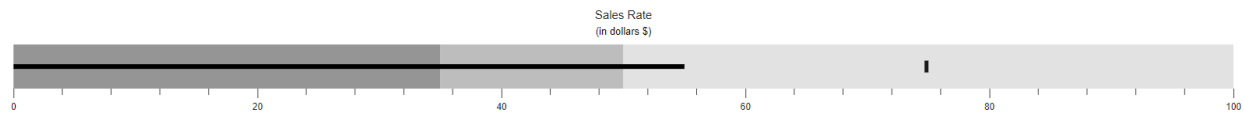
Subtitle

To show additional information about the data plotted, the Bullet Chart can also be given a subtitle using the [Subtitle](#) property.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfBulletChart DataSource="@BulletChartData" Height="150px" Subtitle="(in
dollars $)" Title="Sales Rate" ValueField="FieldValue"
TargetField="TargetValue" Minimum="0" Maximum="100" Interval="20">
<BulletChartRangeCollection>
<BulletChartRange End=35> </BulletChartRange>
<BulletChartRange End=50></BulletChartRange>
<BulletChartRange End=100></BulletChartRange>
</BulletChartRangeCollection>
</SfBulletChart>
```

Refer to the [code block](#) to know about the property value of the **BulletChartData**.



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](#).

ASPX-CS

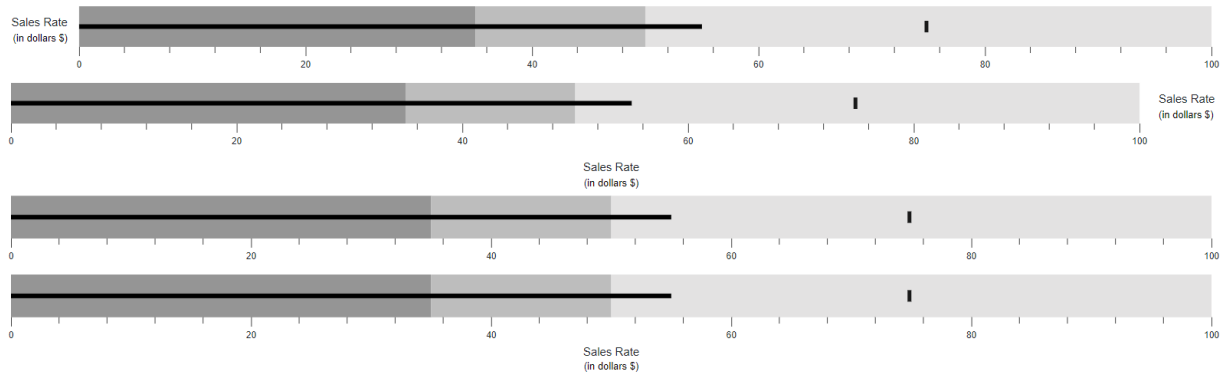
```
@using Syncfusion.Blazor.Charts
<SfBulletChart DataSource="@BulletChartData"
TitlePosition="TextPosition.Left" Height="100px" Subtitle="(in dollars $)"
Title="Sales Rate" ValueField="FieldValue" TargetField="TargetValue"
Minimum="0" Maximum="100" Interval="20">
  <BulletChartRangeCollection>
    <BulletChartRange End=35> </BulletChartRange>
    <BulletChartRange End=50></BulletChartRange>
    <BulletChartRange End=100></BulletChartRange>
  </BulletChartRangeCollection>
</SfBulletChart>

<SfBulletChart DataSource="@BulletChartData"
TitlePosition="TextPosition.Right" Height="100px" Subtitle="(in dollars $)"
Title="Sales Rate" ValueField="FieldValue" TargetField="TargetValue"
Minimum="0" Maximum="100" Interval="20">
  <BulletChartRangeCollection>
    <BulletChartRange End=35> </BulletChartRange>
    <BulletChartRange End=50></BulletChartRange>
    <BulletChartRange End=100></BulletChartRange>
  </BulletChartRangeCollection>
</SfBulletChart>

<SfBulletChart DataSource="@BulletChartData"
TitlePosition="TextPosition.Top" Height="150px" Subtitle="(in dollars $)"
Title="Sales Rate" ValueField="FieldValue" TargetField="TargetValue"
Minimum="0" Maximum="100" Interval="20">
  <BulletChartRangeCollection>
    <BulletChartRange End=35> </BulletChartRange>
    <BulletChartRange End=50></BulletChartRange>
    <BulletChartRange End=100></BulletChartRange>
  </BulletChartRangeCollection>
</SfBulletChart>

<SfBulletChart DataSource="@BulletChartData"
TitlePosition="TextPosition.Bottom" Height="150px" Subtitle="(in dollars $)"
Title="Sales Rate" ValueField="FieldValue" TargetField="TargetValue"
Minimum="0" Maximum="100" Interval="20">
  <BulletChartRangeCollection>
    <BulletChartRange End=35> </BulletChartRange>
    <BulletChartRange End=50></BulletChartRange>
    <BulletChartRange End=100></BulletChartRange>
  </BulletChartRangeCollection>
</SfBulletChart>
```

Refer to the [code block](#) to know about the property value of the **BulletChartData**.



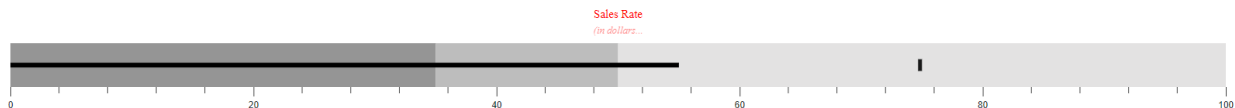
Title and Subtitle Customization

The title and the subtitle - color, opacity, font size, font family, font weight, and font style can be customized using the [BulletChartTitleStyle](#) and the [BulletChartSubTitleStyle](#) settings. The [MaximumTitleWidth](#) property determinate the maximum width of the text, and the [TextOverflow](#) property is used to [Wrap](#) or [Trim](#) the title and the subtitle, when the text size exceeds the [MaximumTitleWidth](#). The default value of the [TextOverflow](#) is [None](#).

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfBulletChart DataSource="@BulletChartData" Height="150px" Title="Sales
Rate" Subtitle="(in dollars $)" ValueField="FieldValue"
TargetField="TargetValue" Minimum="0" Maximum="100" Interval="20">
<BulletChartTitleStyle Color="red" Opacity="1"
Size="15px"></BulletChartTitleStyle>
<BulletChartSubTitleStyle Color="red" Opacity="0.5" Size="13px"
FontStyle="italic" MaximumTitleWidth="70"
TextOverflow="TextOverflow.Trim"></BulletChartSubTitleStyle>
<BulletChartRangeCollection>
<BulletChartRange End=35> </BulletChartRange>
<BulletChartRange End=50></BulletChartRange>
<BulletChartRange End=100></BulletChartRange>
</BulletChartRangeCollection>
</SfBulletChart>
```

Refer to the [code block](#) to know about the property value of the **BulletChartData**.



Customization in Blazor 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.

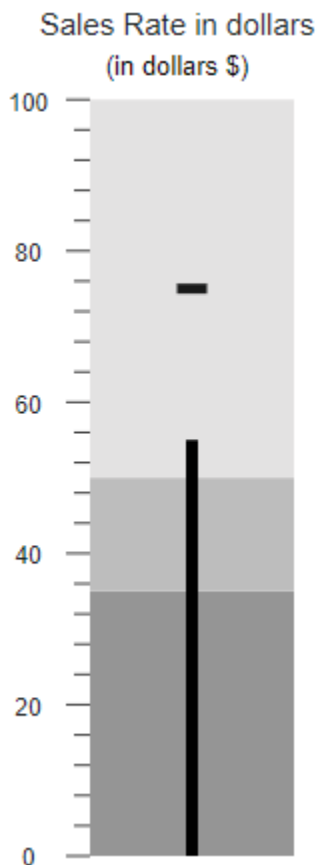
ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfBulletChart DataSource="@BulletChartData"
Orientation="OrientationType.Vertical" Width="20%" Title="Sales Rate in
```

```

dollars" Subtitle="(in dollars $)" ValueField="FieldValue"
TargetField="TargetValue" Minimum="0" Maximum="100" Interval="20">
<BulletChartRangeCollection>
<BulletChartRange End=35></BulletChartRange>
<BulletChartRange End=50></BulletChartRange>
<BulletChartRange End=100></BulletChartRange>
</BulletChartRangeCollection>
</SfBulletChart>
@code{
public class ChartData
{
public double FieldValue { get; set; }
public double TargetValue { get; set; }
}
public List<ChartData> BulletChartData = new List<ChartData>
{
new ChartData { FieldValue = 55, TargetValue = 75 }
};
}

```



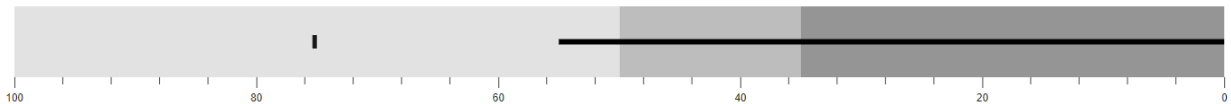
Right-to-left (RTL)

The Bullet Chart supports the right-to-left rendering that can be enabled by setting the [EnableRtl](#) property to **true**.

ASPX-CS

```
<SfBulletChart DataSource="@BulletChartData" EnableRtl="true"
ValueField="FieldValue" TargetField="TargetValue" Minimum="0" Maximum="100"
Interval="20">
<BulletChartRangeCollection>
<BulletChartRange End=35></BulletChartRange>
<BulletChartRange End=50></BulletChartRange>
<BulletChartRange End=100></BulletChartRange>
</BulletChartRangeCollection>
</SfBulletChart>
```

Refer to the [code block](#) to know about the property value of **BulletChartData**.



Animation

The actual and the target bar supports the linear animation via the [BulletChartAnimation](#) setting. The speed and the delay are controlled using the [Duration](#) and [Delay](#) properties respectively.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfBulletChart DataSource="@BulletChartData" ValueField="FieldValue"
TargetField="TargetValue" Minimum="0" Maximum="100" Interval="20">
<BulletChartAnimation Delay="0" Duration="2000"></BulletChartAnimation>
<BulletChartRangeCollection>
<BulletChartRange End=35></BulletChartRange>
<BulletChartRange End=50></BulletChartRange>
<BulletChartRange End=100></BulletChartRange>
</BulletChartRangeCollection>
</SfBulletChart>
```

Refer to the [code block](#) to know about the property value of **BulletChartData**.

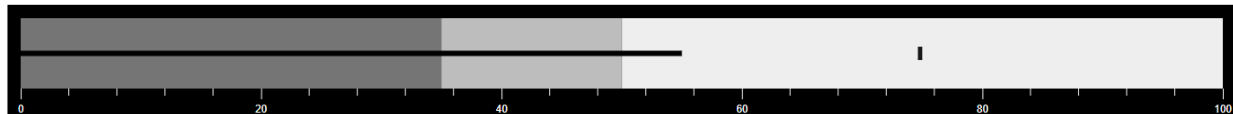
Theme

The Bullet Chart supports different type of themes via the [Theme](#) property.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
@using Syncfusion.Blazor
<SfBulletChart DataSource="@BulletChartData" Theme="Theme.HighContrast"
ValueField="FieldValue" TargetField="TargetValue" Minimum="0" Maximum="100"
Interval="20">
<BulletChartRangeCollection>
<BulletChartRange End=35></BulletChartRange>
<BulletChartRange End=50></BulletChartRange>
<BulletChartRange End=100></BulletChartRange>
</BulletChartRangeCollection>
</SfBulletChart>
```

Refer to the [code block](#) to know about the property value of **BulletChartData**.



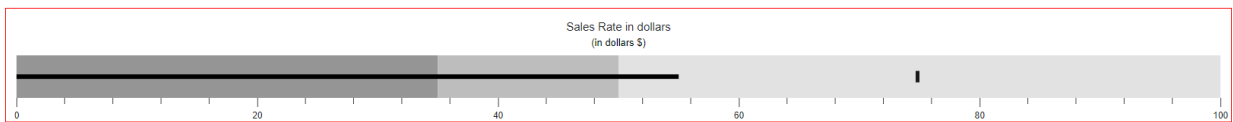
Border

The Bullet Chart border color can be enabled by setting the [Color](#) property in the [BulletChartBorder](#), and the width of the border can be customized using the [Width](#) property.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfBulletChart DataSource="@BulletChartData" Height="150px" Title="Sales
Rate in dollars" Subtitle="(in dollars $)" ValueField="FieldValue"
TargetField="TargetValue" Minimum="0" Maximum="100" Interval="20">
  <BulletChartBorder Color="red" Width="2"></BulletChartBorder>
  <BulletChartRangeCollection>
    <BulletChartRange End=35> </BulletChartRange>
    <BulletChartRange End=50></BulletChartRange>
    <BulletChartRange End=100></BulletChartRange>
  </BulletChartRangeCollection>
</SfBulletChart>
```

Refer to the [code block](#) to know about the property value of **BulletChartData**.



Data Labels in Blazor 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 [BulletChartDataLabel](#) setting.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfBulletChart DataSource="@BulletChartData" ValueField="ValueField"
TargetField="ComparativeMeasureValue" CategoryField="Category" Height="400"
Minimum="0" Maximum="20" Interval="5" LabelFormat="{value}%" Title="Profit
in Percentage">
  <BulletChartDataLabel></BulletChartDataLabel>
  <BulletChartMinorTickLines Width="0"></BulletChartMinorTickLines>
  <BulletChartRangeCollection>
    <BulletChartRange End=5> </BulletChartRange>
    <BulletChartRange End=15></BulletChartRange>
    <BulletChartRange End=20></BulletChartRange>
  </BulletChartRangeCollection>
</SfBulletChart>

@code{
public class ChartData
{
    public double ValueField { get; set; }
    public double ComparativeMeasureValue { get; set; }
    public string Category { get; set; }
}
}
```

```
public List<ChartData> BulletChartData = new List<ChartData>
{
    new ChartData { ValueField = 5, ComparativeMeasureValue = 7.5, Category = "2001" },
    new ChartData { ValueField = 7, ComparativeMeasureValue = 5, Category = "2002" },
    new ChartData { ValueField = 10, ComparativeMeasureValue = 6, Category = "2003" },
    new ChartData { ValueField = 5, ComparativeMeasureValue = 8, Category = "2004" },
    new ChartData { ValueField = 12, ComparativeMeasureValue = 5, Category = "2005" },
    new ChartData { ValueField = 8, ComparativeMeasureValue = 6, Category = "2006" }
};
```

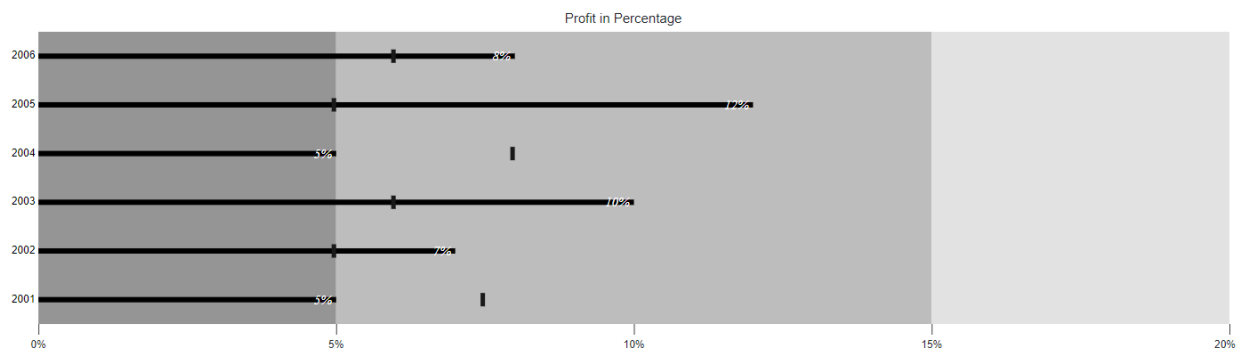
Data Label Customization

Data Labels color, opacity, font size, font family, font weight, and font style can be customized using the [BulletChartDataLabelStyle](#).

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfBulletChart DataSource="@BulletChartData" ValueField="ValueField"
TargetField="ComparativeMeasureValue" CategoryField="Category" Height="400"
Minimum="0" Maximum="20" Interval="5" LabelFormat="{value}%" Title="Profit
in Percentage">
    <BulletChartDataLabel>
        <BulletChartDataLabelStyle Color="#FFFFFF" Opacity="1" Size="15px"
FontStyle="italic"></BulletChartDataLabelStyle>
    </BulletChartDataLabel>
    <BulletChartMinorTickLines Width="0"></BulletChartMinorTickLines>
    <BulletChartRangeCollection>
        <BulletChartRange End=5> </BulletChartRange>
        <BulletChartRange End=15></BulletChartRange>
        <BulletChartRange End=20></BulletChartRange>
    </BulletChartRangeCollection>
</SfBulletChart>
```

Refer to the [code block](#) to know about the property value of the **BulletChartData**.



Tooltip in Blazor Bullet Chart Component

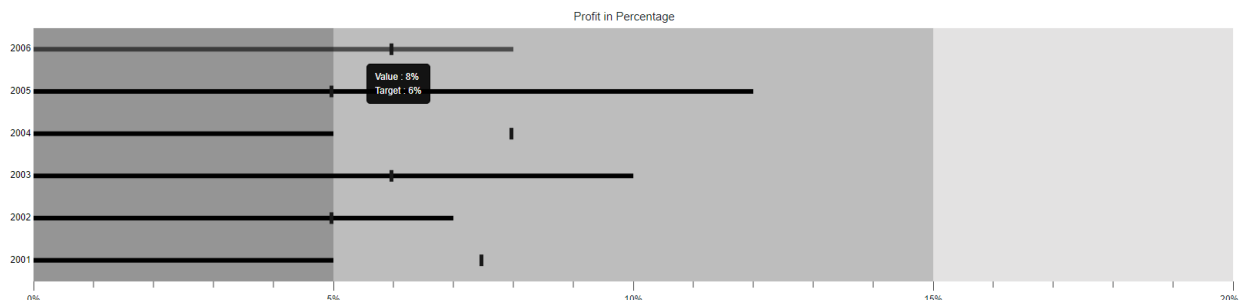
When the mouse is hovered over a bar in the Bullet Chart, the tooltip displays important summary about the actual and the target bar values.

Default Tooltip

The tooltip is not visible by default. To make it visible, set the [Enable](#) property in the [BulletChartTooltip](#) to **true**.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfBulletChart DataSource="@BulletChartData" ValueField="ValueField"
TargetField="TargetValue" CategoryField="Category" Height="400" Minimum="0"
Maximum="20" Interval="5" LabelFormat="{value}%" Title="Profit in
Percentage">
<BulletChartTooltip TValue="ChartData" Enable="true"></BulletChartTooltip>
<BulletChartRangeCollection>
<BulletChartRange End=5> </BulletChartRange>
<BulletChartRange End=15></BulletChartRange>
<BulletChartRange End=20></BulletChartRange>
</BulletChartRangeCollection>
</SfBulletChart>
@code{
public class ChartData
{
public double ValueField { get; set; }
public double TargetValue { get; set; }
public string Category { get; set; }
}
public List<ChartData> BulletChartData = new List<ChartData>
{
new ChartData { ValueField = 5, TargetValue = 7.5, Category = "2001" },
new ChartData { ValueField = 7, TargetValue = 5, Category = "2002" },
new ChartData { ValueField = 10, TargetValue = 6, Category = "2003" },
new ChartData { ValueField = 5, TargetValue = 8, Category = "2004" },
new ChartData { ValueField = 12, TargetValue = 5, Category = "2005" },
new ChartData { ValueField = 8, TargetValue = 6, Category = "2006" }
};
}
```



Tooltip Customization

The following properties can be used to customize the Bullet Chart tooltip.

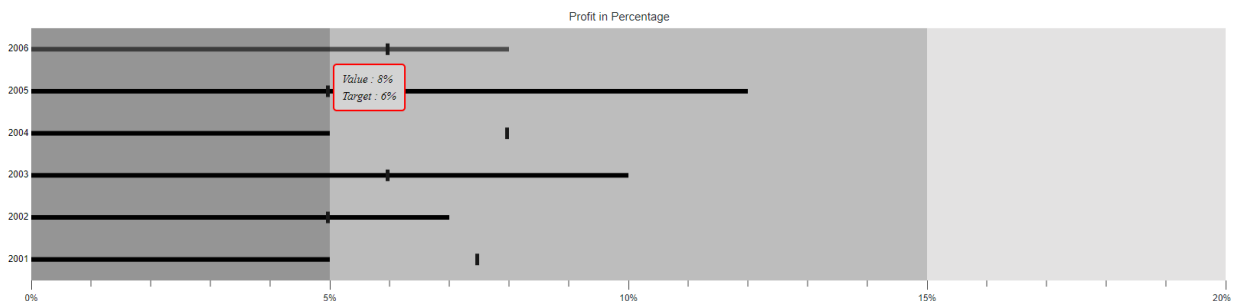
- [Fill](#) - Specifies the color of tooltip.

- [BulletChartTooltipBorder](#) - Specifies the tooltip border color and width.
- [BulletChartTooltipTextStyle](#) - Specifies the tooltip font family, font style, font weight, color and size.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfBulletChart DataSource="@BulletChartData" ValueField="ValueField"
TargetField="TargetValue" CategoryField="Category" Height="400" Minimum="0"
Maximum="20" Interval="5" LabelFormat="{value}%" Title="Profit in
Percentage">
<BulletChartTooltip TValue="ChartData" Enable="true" Fill="lightgray">
<BulletChartTooltipTextStyle Color="#000000" Opacity="1" Size="15px"
FontStyle="italic"></BulletChartTooltipTextStyle>
<BulletChartTooltipBorder Color="red" Width="2"></BulletChartTooltipBorder>
</BulletChartTooltip>
<BulletChartRangeCollection>
<BulletChartRange End=5> </BulletChartRange>
<BulletChartRange End=15></BulletChartRange>
<BulletChartRange End=20></BulletChartRange>
</BulletChartRangeCollection>
</SfBulletChart>
```

Refer to the [code block](#) to know about the property value of the **BulletChartData**.



Tooltip Template

The tooltip can be rendered as a custom component using the [Template](#) property in the [BulletChartTooltip](#) which accepts one or more UI elements as an input, that can be rendered as a part of the tooltip rendering.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfBulletChart DataSource="@BulletChartData" ValueField="ValueField"
TargetField="TargetValue" CategoryField="Category" Height="400" Minimum="0"
Maximum="20" Interval="5" LabelFormat="{value}%" Title="Profit in
Percentage">
<BulletChartTooltip TValue="ChartData" Enable="true" Fill="lightgray">
<Template>
@{
<table style="width:100%; background-color: #ffffff; border-spacing: 0px;
border-collapse: separate; border: 1px solid grey; border-radius: 10px;
padding-top: 5px; padding-bottom: 5px">
<tr>
```

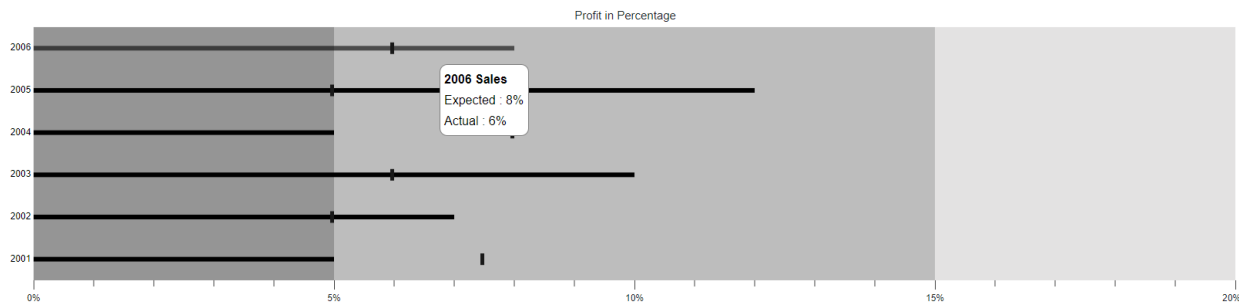


```

<td style="font-weight:bold; color:black; padding-left: 5px;padding-top: 2px;padding-bottom: 2px;">@context.Category Sales</td>
</tr>
<tr>
<td style="padding-left: 5px; color:black; padding-right: 5px; padding-bottom: 2px;">Expected : @context.ValueField% </td>
</tr>
<tr>
<td style="padding-left: 5px; color:black; padding-right: 5px">Actual : @context.TargetValue% </td>
</tr>
</table>
}
</Template>
</BulletChartTooltip>
<BulletChartRangeCollection>
<BulletChartRange End=5> </BulletChartRange>
<BulletChartRange End=15></BulletChartRange>
<BulletChartRange End=20></BulletChartRange>
</BulletChartRangeCollection>
</SfBulletChart>

```

Refer to the [code block](#) to know about the property value of the **BulletChartData**.



Legend in Blazor Bullet Chart Component

Legend is used to provide a valuable information for interpreting what the Bullet Chart displays. The legends can be represented in various colors, positions, shapes or other identifiers based on the data and it can be enabled by the [Visible](#) property to **true** in the [BulletChartLegendSettings](#).

ASPX-CS

```

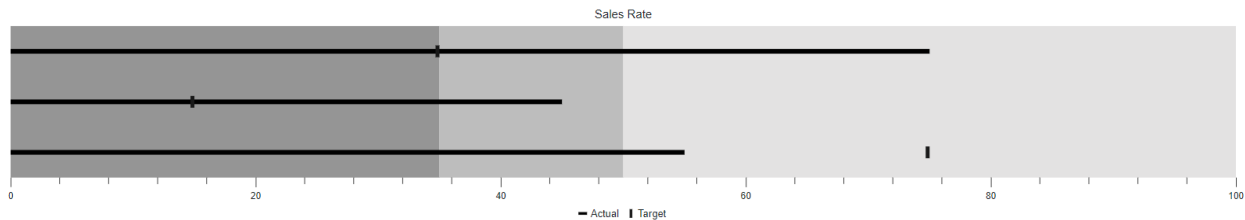
@using Syncfusion.Blazor.Charts
<SfBulletChart DataSource="@BulletChartData" Height="300px" Title="Sales Rate" ValueField="FieldValue" TargetField="TargetValue" Minimum="0" Maximum="100" Interval="20">
<BulletChartLegendSettings Visible="true"></BulletChartLegendSettings>
<BulletChartRangeCollection>
<BulletChartRange End=35> </BulletChartRange>
<BulletChartRange End=50></BulletChartRange>
<BulletChartRange End=100></BulletChartRange>
</BulletChartRangeCollection>
</SfBulletChart>
@code{
public class ChartData
{

```

```

public double FieldValue { get; set; }
public double TargetValue { get; set; }
}
public List<ChartData> BulletChartData = new List<ChartData>
{
    new ChartData { FieldValue = 55, TargetValue = 75 },
    new ChartData { FieldValue = 45, TargetValue = 15 },
    new ChartData { FieldValue = 75, TargetValue = 35 }
};
}

```



Legend items from color mapping

Legend items will be rendered based on the mapping ranges from the Bullet Chart. The legend item's name can be determined from the [Name](#) property and the shape of legend item can be customized using the [Shape](#) property in the [BulletChartRange](#). By default, the legend item is rendered on the [Rectangle](#) shape.

Legend item to be customized by the following properties.

- [Padding](#) - Specifies the padding between the legend items.
- [ShapeHeight](#) - Specifies the shape height of the legend items.
- [ShapeWidth](#) - Specifies the shape width of the legend items.
- [ShapePadding](#) - Specifies the padding between the shape and the text of the legend item.
- [BulletChartLegendTextStyle](#) - Specifies the text style of the legend item.

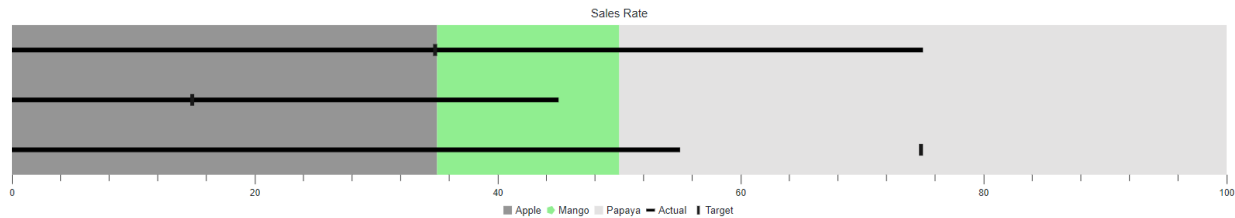
ASPX-CS

```

@using Syncfusion.Blazor.Charts
<SfBulletChart DataSource="@BulletChartData" Height="300px" Title="Sales
Rate" ValueField="FieldValue" TargetField="TargetValue" Minimum="0"
Maximum="100" Interval="20">
    <BulletChartLegendSettings Visible="true"></BulletChartLegendSettings>
    <BulletChartRangeCollection>
        <BulletChartRange End=35 Name="Apple"></BulletChartRange>
        <BulletChartRange End=50 Name="Mango" Color="lightgreen"
Shape="LegendShape.Pentagon"></BulletChartRange>
        <BulletChartRange End=100 Name="Papaya"></BulletChartRange>
    </BulletChartRangeCollection>
</SfBulletChart>

```

Refer to the [code block](#) to know the property value of the **BulletChartData**.



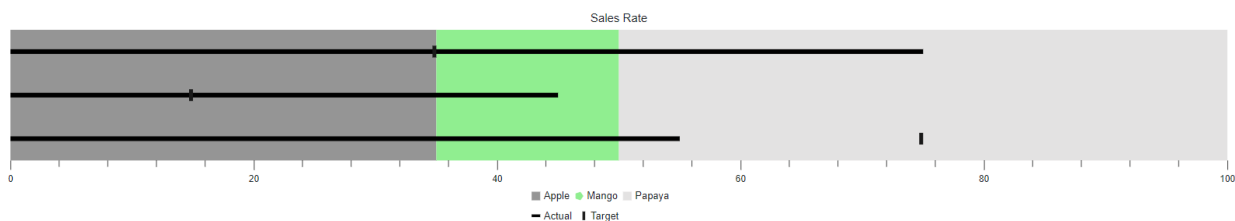
Legend size

Customize the legend size by modifying the [Height](#) and the [Width](#) properties in the [BulletChartLegendSettings](#). It accepts values in both percentage and pixel.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfBulletChart DataSource="@BulletChartData" Height="300px" Title="Sales
Rate" ValueField="FieldValue" TargetField="TargetValue" Minimum="0"
Maximum="100" Interval="20">
  <BulletChartLegendSettings Visible="true"
  Width="15%"></BulletChartLegendSettings>
  <BulletChartRangeCollection>
    <BulletChartRange End=35 Name="Apple"></BulletChartRange>
    <BulletChartRange End=50 Name="Mango" Color="lightgreen"
    Shape="LegendShape.Pentagon"></BulletChartRange>
    <BulletChartRange End=100 Name="Papaya"></BulletChartRange>
  </BulletChartRangeCollection>
</SfBulletChart>
```

Refer to the [code block](#) to know about the property value of the **BulletChartData**.



Legend with paging support

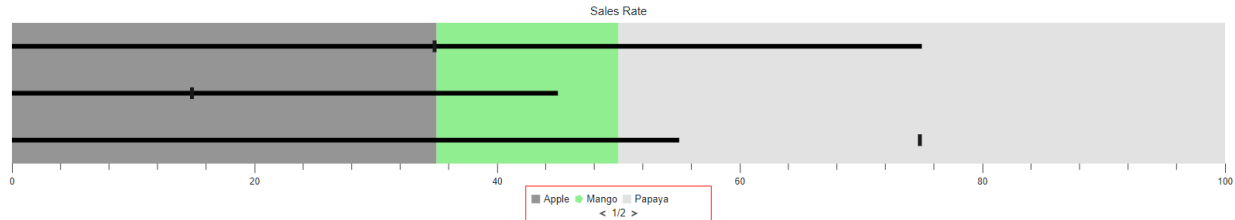
Bullet Chart supports the legend paging, if the legend items cannot be placed within the provided [Height](#) and [Width](#) of the legend.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfBulletChart DataSource="@BulletChartData" Height="300px" Title="Sales
Rate" ValueField="FieldValue" TargetField="TargetValue" Minimum="0"
Maximum="100" Interval="20">
  <BulletChartLegendSettings Visible="true" Width="15%" Height="15%">
    <BulletChartLegendBorder Color="red" Width="1"></BulletChartLegendBorder>
  </BulletChartLegendSettings>
  <BulletChartRangeCollection>
    <BulletChartRange End=35 Name="Apple"></BulletChartRange>
    <BulletChartRange End=50 Name="Mango" Color="lightgreen"
    Shape="LegendShape.Pentagon"></BulletChartRange>
    <BulletChartRange End=100 Name="Papaya"></BulletChartRange>
  </BulletChartRangeCollection>
</SfBulletChart>
```

```
</BulletChartRangeCollection>
</SfBulletChart>
```

Refer to the [code block](#) to know about the property value of the **BulletChartData**.



Position and Alignment

The legend can be placed to various positions and the following options are available to customize the legend position using the [Position](#) property:

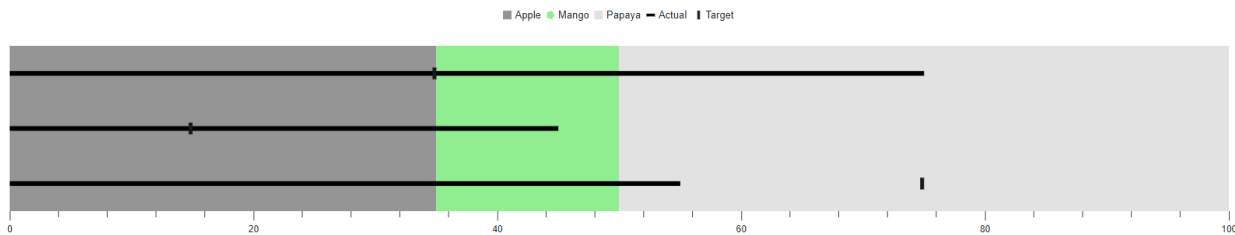
- [Auto](#)
- [Bottom](#)
- [Top](#)
- [Left](#)
- [Right](#)
- [Custom](#)

The following code example demonstrates the top legend position.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfBulletChart DataSource="@BulletChartData" Height="300px"
ValueField="FieldValue" TargetField="TargetValue" Minimum="0" Maximum="100"
Interval="20">
<BulletChartLegendSettings Height="40px" Visible="true"
Position="LegendPosition.Top"></BulletChartLegendSettings>
<BulletChartRangeCollection>
<BulletChartRange End=35 Name="Apple"></BulletChartRange>
<BulletChartRange End=50 Name="Mango" Color="lightgreen"
Shape="LegendShape.Pentagon"></BulletChartRange>
<BulletChartRange End=100 Name="Papaya"></BulletChartRange>
</BulletChartRangeCollection>
</SfBulletChart>
```

Refer to the [code block](#) to know about the property value of the **BulletChartData**.



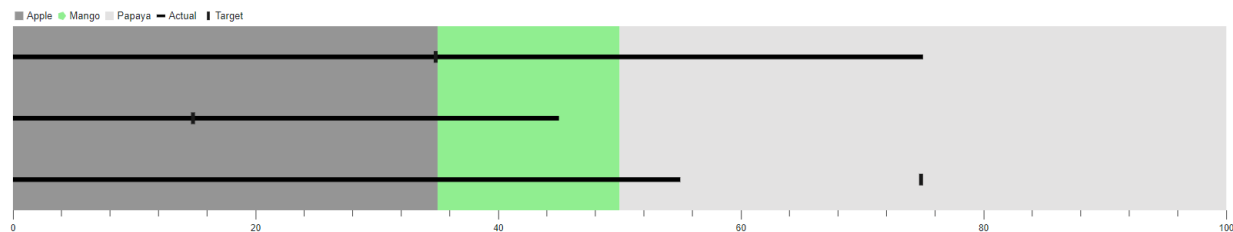
[Auto](#) position will be rendered with the responsive legend height to the [Bottom](#) of the component and the [Custom](#) position renders based on x and y coordinates by specified to [X](#) and [Y](#) properties in the [BulletChartLegendLocation](#).

The following code example demonstrates the [Custom](#) legend position.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfBulletChart DataSource="@BulletChartData" Height="300px"
ValueField="FieldValue" TargetField="TargetValue" Minimum="0" Maximum="100"
Interval="20">
<BulletChartMargin Top="30"></BulletChartMargin>
<BulletChartLegendSettings Height="40px" Visible="true"
Position="LegendPosition.Custom">
<BulletChartLegendLocation X="5" Y="0"></BulletChartLegendLocation>
</BulletChartLegendSettings>
<BulletChartRangeCollection>
<BulletChartRange End=35 Name="Apple"></BulletChartRange>
<BulletChartRange End=50 Name="Mango" Color="lightgreen"
Shape="LegendShape.Pentagon"></BulletChartRange>
<BulletChartRange End=100 Name="Papaya"></BulletChartRange>
</BulletChartRangeCollection>
</SfBulletChart>
```

Refer to the [code block](#) to know about the property value of the **BulletChartData**.



The legend alignment is used to align the legend items to the specific location. The following options are available to customize using the [Alignment](#) property:

- [Near](#)
- [Center](#)
- [Far](#)

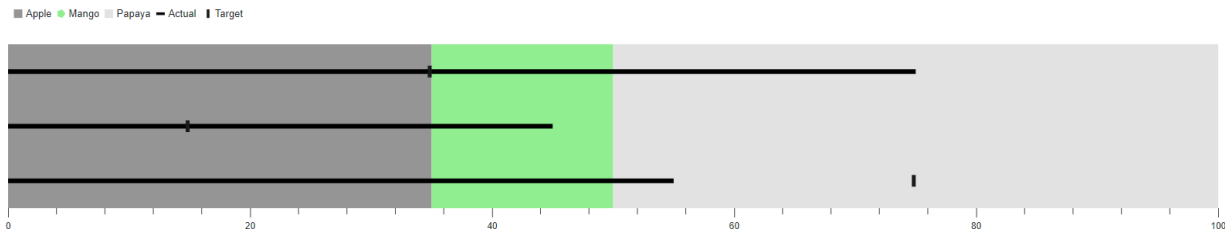
The following code example demonstrates legend item alignment.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfBulletChart DataSource="@BulletChartData" Height="300px"
ValueField="FieldValue" TargetField="TargetValue" Minimum="0" Maximum="100"
Interval="20">
<BulletChartLegendSettings Height="40px" Visible="true"
Position="LegendPosition.Top"
Alignment="Alignment.Near"></BulletChartLegendSettings>
<BulletChartRangeCollection>
<BulletChartRange End=35 Name="Apple"></BulletChartRange>
```

```
<BulletChartRange End=50 Name="Mango" Color="lightgreen"
Shape="LegendShape.Pentagon"></BulletChartRange>
<BulletChartRange End=100 Name="Papaya"></BulletChartRange>
</BulletChartRangeCollection>
</SfBulletChart>
```

Refer to the [code block](#) to know about the property value of the **BulletChartData**.



Customization

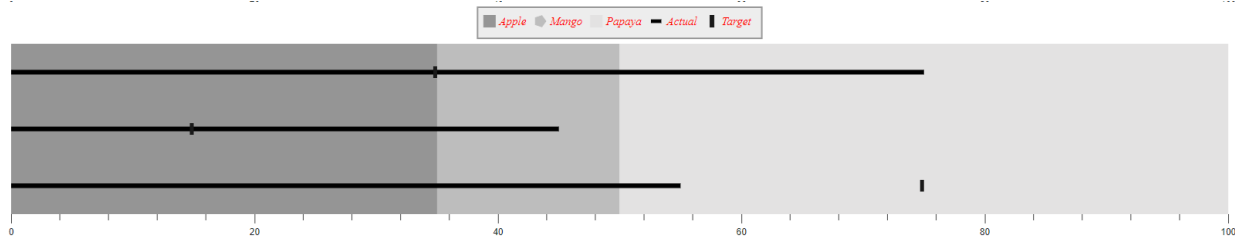
Legend can be customized by the following properties:

- [Background](#) - Specifies the fill color of the legend.
- [Opacity](#) - Specifies the opacity of the legend background.
- [BulletChartLegendMargin](#) - Specifies the bottom, left, right, and top margin of the legend.
- [BulletChartLegendBorder](#) - Specifies the color and the width of the legend border.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfBulletChart DataSource="@BulletChartData" Height="300px"
ValueField="FieldValue" TargetField="TargetValue" Minimum="0" Maximum="100"
Interval="20">
<BulletChartLegendSettings Height="40px" Visible="true"
Position="LegendPosition.Top" Background="lightgray" Opacity="0.4"
ShapeHeight="15" ShapeWidth="15">
<BulletChartMargin Top="5"></BulletChartMargin>
<BulletChartLegendTextStyle Size="15px" Color="red" Opacity="1"
FontStyle="italic"></BulletChartLegendTextStyle>
<BulletChartLegendBorder Color="#000000"
Width="2"></BulletChartLegendBorder>
</BulletChartLegendSettings>
<BulletChartRangeCollection>
<BulletChartRange End=35 Name="Apple"></BulletChartRange>
<BulletChartRange End=50 Name="Mango"
Shape="LegendShape.Pentagon"></BulletChartRange>
<BulletChartRange End=100 Name="Papaya"></BulletChartRange>
</BulletChartRangeCollection>
</SfBulletChart>
```

Refer to the [code block](#) to know about the property value of the **BulletChartData**.



Events in Blazor Bullet Chart Component

This section describes about the Bullet Chart component's events that will be triggered when appropriate actions are performed. The events should be provided to the Bullet Chart through the [BulletChartEvents](#).

Loaded

The **Loaded** event triggers after the Bullet Chart component has been loaded.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfBulletChart DataSource="@BulletChartData" ValueField="FieldValue"
TargetField="TargetValue" Minimum="0" Maximum="300" Interval="50">
<BulletChartEvents Loaded="LoadedHandler"></BulletChartEvents>
</SfBulletChart>
@code{
public class ChartData
{
public double FieldValue { get; set; }
public double TargetValue { get; set; }
}
public List<ChartData> BulletChartData = new List<ChartData>
{
new ChartData { FieldValue = 270, TargetValue = 250 }
};
public void LoadedHandler(System.EventArgs args)
{
// Here you can customize the code.
}
}
```

OnPrintComplete

The [OnPrintComplete](#) event triggers before the rendered Bullet Chart starts printing.

Argument name	Description
Cancel	Specifies the event cancel status.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<button onclick="PrintCall">OnPrint</button>
<SfBulletChart @ref="@BulletChart" DataSource="@BulletChartData"
ValueField="FieldValue" TargetField="TargetValue" Minimum="0" Maximum="300"
Interval="50">
```

```

<BulletChartEvents
OnPrintComplete="PrintCompleteHandler"></BulletChartEvents>
</SfBulletChart>
@code{
public SfBulletChart<ChartData> BulletChart { get; set; }
public class ChartData
{
public double FieldValue { get; set; }
public double TargetValue { get; set; }
}
public List<ChartData> BulletChartData = new List<ChartData>
{
new ChartData { FieldValue = 270, TargetValue = 250 }
};
public async Task PrintCall()
{
await BulletChart.PrintAsync();
}
public void PrintCompleteHandler(PrintEventArgs args)
{
// Here you can customize the code.
}
}

```

TooltipRender

The [TooltipRender](#) event triggers before the tooltip rendering.

Argument name	Description
-----	-----
Target	Specifies the Target Bar values.
Text	Specifies the content of the tooltip.
Value	Specifies the Value Bar data.
Cancel	Specifies the event cancel status.

ASPX-CS

```

@using Syncfusion.Blazor.Charts
<SfBulletChart DataSource="@BulletChartData" ValueField="FieldValue"
TargetField="TargetValue" Minimum="0" Maximum="300" Interval="50">
<BulletChartEvents TooltipRender="TooltipRenderHandler"></BulletChartEvents>
<BulletChartTooltip TValue="ChartData" Enable="true"></BulletChartTooltip>
</SfBulletChart>
@code{
public class ChartData
{
public double FieldValue { get; set; }
public double TargetValue { get; set; }
}
public List<ChartData> BulletChartData = new List<ChartData>
{
new ChartData { FieldValue = 270, TargetValue = 250 }
};
public void TooltipRenderHandler(BulletChartTooltipEventArgs args)

```



```
{
// Here you can customize the code.
}
```

LegendRender

The [LegendRender](#) event triggers before each legend item rendering.

Argument name	Description
Fill	Specifies the fill of the legend item.
Shape	Specifies the shape of the legend item.
Text	Specifies the text of the legend item.
Cancel	Specifies the event cancel status.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfBulletChart DataSource="@BulletChartData" Height="300px" Title="Sales
Rate" ValueField="FieldValue" TargetField="TargetValue" Minimum="0"
Maximum="100" Interval="20">
<BulletChartEvents LegendRender="LegendRenderHandler"></BulletChartEvents>
<BulletChartLegendSettings Visible="true"
Width="15%"></BulletChartLegendSettings>
<BulletChartRangeCollection>
<BulletChartRange End=35 Name="Apple"></BulletChartRange>
<BulletChartRange End=50 Name="Mango" Color="lightgreen"
Shape="LegendShape.Pentagon"></BulletChartRange>
<BulletChartRange End=100 Name="Papaya"></BulletChartRange>
</BulletChartRangeCollection>
</SfBulletChart>
@code{
public class ChartData
{
public double FieldValue { get; set; }
public double TargetValue { get; set; }
}
public List<ChartData> BulletChartData = new List<ChartData>
{
new ChartData { FieldValue = 55, TargetValue = 75 },
new ChartData { FieldValue = 45, TargetValue = 15 },
new ChartData { FieldValue = 75, TargetValue = 35 }
};
public void LegendRenderHandler(BulletChartLegendRenderEventArgs args)
{
// Here you can customize the code.
}
```

Button

<!-- markdownlint-disable MD024 -->

Getting Started with Blazor Button Component

This section briefly explains about how to include [Button](#) Component in your Blazor server-side application. You can refer [Getting Started with Syncfusion Blazor for Server-side in Visual Studio 2019 page](#) for the introduction and configuring the common specifications.

To get start quickly with Button Component using Blazor, you can check on this video:

{% youtube

"youtube:https://www.youtube.com/watch?v=qkHqP_Cymrl"%}

Importing Syncfusion Blazor component in the application

1. Install the [Syncfusion.Blazor](#) NuGet package to the application by using the **NuGet Package Manager**.
2. You can add the client-side style resources through [CDN](#) or from [NuGet](#) package in the `element` of the `~/Pages/_Host.cshtml` page.

HTML

```
<head>
<link href="_content/Syncfusion.Blazor/styles/bootstrap4.css"
rel="stylesheet" />
@*<link href="https://cdn.syncfusion.com/blazor/{ site.blazorversion
}/styles/bootstrap4.css" rel="stylesheet" />*@
</head>
```

For Internet Explorer 11 kindly refer the polyfills. Refer the [documentation](#) for more information.

HTML

```
<head>
<link href="_content/Syncfusion.Blazor/styles/bootstrap4.css"
rel="stylesheet" />
<script
src="https://github.com/Daddoon/Blazor.Polyfill/releases/download/3.0.1/blaz
or.polyfill.min.js"></script>
</head>
```

Adding component package to the application

Open `/_Imports.razor` file and import the **Syncfusion.Blazor.Buttons** package.

ASPX-CS

```
@using Syncfusion.Blazor.Buttons
```

Add SyncfusionBlazor service in Startup.cs

Open the **Startup.cs** file and add services required by Syncfusion components.

Add **services.AddSyncfusionBlazor()** method in the `ConfigureServices` function as follows.

CSHARP

```
namespace BlazorApplication
```

```
{  
public class Startup  
{  
    ....  
    ....  
    public void ConfigureServices(IServiceCollection services)  
    {  
        ....  
        ....  
        services.AddSyncfusionBlazor();  
    }  
}
```

To enable custom client side resource loading from CRG or CDN. You need to disable resource loading by **AddSyncfusionBlazor(true)** and load the scripts in the HEAD element of the `~/Pages/_Host.cshtml` page.

ASPX-CS

```
<head>  
<environment include="Development">  
<script src="https://cdn.syncfusion.com/blazor/{{ site.blazorversion  
}}/syncfusion-blazor.min.js">  
</script>  
</environment>  
</head>
```

Adding component package to the application

Open `/_Imports.razor` file and import the `Syncfusion.Blazor.Buttons` packages otherwise import these packages in the individual `razor` pages.

ASPX-CS

```
@using Syncfusion.Blazor  
@using Syncfusion.Blazor.Buttons
```

Adding Button component to the application

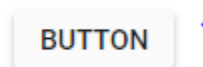
Now, add the Syncfusion Blazor Button component in `razor` page in the `Pages` folder. For example the Button component is added in the `~/Pages/Index.razor` page.

ASPX-CS

```
<SfButton>Button</SfButton>
```

Run the application

After successful compilation of your application, simply press F5 to run the application. The Blazor Button component will render in the web browser as shown below



You can refer to our [Blazor Button](#) feature tour page for its groundbreaking feature representations. You can also explore our [Blazor Button example](#) that shows how to configure the button in Blazor.

See Also

- [Getting Started with Syncfusion Blazor for Client-Side in .NET Core CLI](#)
- [Getting Started with Syncfusion Blazor for Server-Side in Visual Studio 2019 Preview](#)
- [Getting Started with Syncfusion Blazor for Server-Side in .NET Core CLI](#)

Native Events in Blazor Button Component

You can define the native event using an `event` attribute in component. The value of attribute is treated as an event handler. The event specific data will be available in event arguments.

The different event argument types for each event are,

- Focus Events - `UIFocusEventArgs`
- Mouse Events - `UIMouseEventArgs`
- Keyboard Events - `UIKeyboardEventArgs`
- Touch Events – `UITouchEventArgs`

List of Native events supported

We have provided the following native event support to the Button component:

```
| List of Native events | | | | |
|---|---|---|---|---|
| onclick | onblur | onfocus | onfocusout |
| onmousemove | onmouseover | onmouseout | onmousedown | onmouseup |
| ondblclick | onkeydown | onkeyup | onkeypress |
| ontouchend | onfocusin | onmouseup | ontouchstart |
```

How to bind click event to Button

The `onclick` attribute is used to bind the click event for button. Here, we have explained about the sample code snippets of toggle button.

CSHARP

```
@using Syncfusion.Blazor.Buttons
<SfButton @ref="ToggleBtn" @onclick="onToggleClick" CssClass="e-flat"
IsToggle="true" IsPrimary="true" Content="@Content"></SfButton>
@code {
    SfButton ToggleBtn;
    public string Content = "Play";
    private void
    onToggleClick(Microsoft.AspNetCore.Components.Web.MouseEventArgs args)
    {
        if (ToggleBtn.Content == "Play")
        {
            this.Content = "Pause";
        }
        else
```

```
{  
  this.Content = "Play";  
}  
}
```

Types and Styles in Blazor Button Component

This section explains the different styles and types of Buttons.

Button styles

The Blazor 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.

CSHARP

```
@using Syncfusion.Blazor.Buttons  
<SfButton CssClass="e-primary">Primary</SfButton>  
<SfButton CssClass="e-success">Success</SfButton>  
<SfButton CssClass="e-info">Info</SfButton>  
<SfButton CssClass="e-warning">Warning</SfButton>  
<SfButton CssClass="e-danger">Danger</SfButton>  
<SfButton CssClass="e-link">Link</SfButton>
```

Output will be as follows



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.

Button types

The types of Blazor Button are as follows:

- Flat Button
- Outline Button
- Round Button
- Primary Button

- Toggle Button

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 circular in shape. Usually, it contains an icon representing its action. To create a Round Button, set the [CssClass](#) property to `e-round`.

ASPX-CS

```
@using Syncfusion.Blazor.Buttons
<SfButton CssClass="e-flat">Flat</SfButton>
<SfButton CssClass="e-outline">Outline</SfButton>
<SfButton CssClass="e-round" IconCss="e-icons e-plus-icon"
IsPrimary="true"></SfButton>
<style>
.e-plus-icon::before {
content: '\e823';
}
</style>
```

Output will be as follows

FLAT

OUTLINE



Primary Button

The primary button is styled with background color and it is used to represent a primary action. To create a Primary Button, set the [IsPrimary](#) property to `true`.

ASPX-CS

```
@using Syncfusion.Blazor.Buttons
<SfButton IsPrimary="true">Primary</SfButton>
```

Output will be as follows

PRIMARY

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

[OnClick](#) 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 OnClick event.

CSHARP

```
@using Syncfusion.Blazor.Buttons
<SfButton CssClass="e-flat" IsPrimary="true" IconCss="@IconCss"
Content="@Content" IsToggle="true" @onclick="OnToggleClick"
@ref="ToggleBtnObj"></SfButton>
@code {
    SfButton ToggleBtnObj;
    public string IconCss = "e-icons e-play";
    public string Content = "Play";
    public void OnToggleClick()
    {
        if(ToggleBtnObj.Content == "Play")
        {
            this.Content = "Pause";
            this.IconCss = "e-icons e-pause";
        }
        else
        {
            this.Content = "Play";
            this.IconCss = "e-icons e-play";
        }
    }
}
<style>
.e-play::before {
content: '\e324';
}
.e-pause::before {
content: '\e326';
}
</style>
```

Output will be as follows



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 to `e-icons` 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.

ASPX-CS

```
@using Syncfusion.Blazor.Buttons
```

```
<SfButton IconCss="e-icons e-play-icon"
IconPosition="IconPosition.Right">PLAY</SfButton>
<SfButton IconCss="e-icons e-pause-icon">PAUSE</SfButton>
<style>
.e-play-icon::before {
content: '\e324';
}
.e-pause-icon::before {
content: '\e326';
}
</style>
```

Output will be as follows



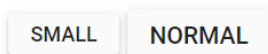
Button size

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`.

ASPX-CS

```
@using Syncfusion.Blazor.Buttons
<SfButton CssClass="e-small">SMALL</SfButton>
<SfButton>NORMAL</SfButton>
```

Output will be as follows



Styles and Appearances in Blazor Button Component

To modify the Button appearance, you need to override the default CSS of Button component. Please find the list of CSS classes and its corresponding section in Button 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-btn | To customize the button. |

| .e-btn:hover | To customize the button on hover. |

`|.e-btn:focus|` To customize the button on focus.

`|.e-btn:active|` To customize the button on active.

How To

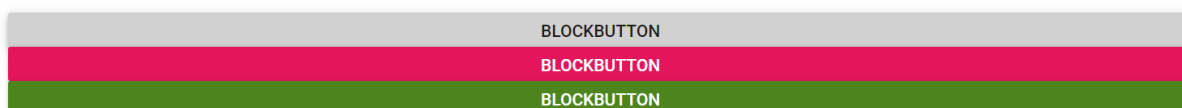
Create a Block Button in Blazor Button Component

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 to `e-block`.

ASPX-CS

```
@using Syncfusion.Blazor.Buttons
<SfButton CssClass="e-block">BLOCKBUTTON</SfButton>
<SfButton CssClass="e-block" IsPrimary="true">BLOCKBUTTON</SfButton>
<SfButton CssClass="e-block e-success">BLOCKBUTTON</SfButton>
```

Output be like



Customize Button Appearance in Blazor 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).

CSHARP

```
@using Syncfusion.Blazor.Buttons
<SfButton CssClass="e-custom">CUSTOM</SfButton>
<style>
.e-custom {
border-radius: 0;
height: 30px;
width: 80px;
}
.e-custom, .e-custom: hover, .e-custom: focus, .e-custom: active {
background-color: #ff6e40;
color: #fff;
}
</style>
```

Output be like



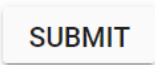
HTML Attribute Support in Blazor Button Component

HTML attribute support is given for button using [HtmlAttributes](#) property.

CSHARP

```
@using Syncfusion.Blazor.Buttons
<SfButton Content="@Content" HtmlAttributes="@submit"></SfButton>
@code {
    public string Content = "Submit";
    private Dictionary<string, object> submit = new Dictionary<string, object>()
    {
        { "type", "submit"}
    };
}
```

Output be like



Repeat Button in Blazor 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.

CSHARP

```
@using Syncfusion.Blazor.Buttons
<div id="preview">@EventName Event is triggered</div>
<SfButton Content="Button" @onclick="Click"></SfButton>
@code{
    public string EventName = "No";
    public void Click()
    {
        this.EventName = "Click";
    }
}
<style>
#preview{
float: right;
padding: 0 350px 0 0;
}
</style>
```

Output be like



Click Event is triggered

Right-To-Left in Blazor 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.

ASPX-CS

```
@using Syncfusion.Blazor.Buttons
<SfButton IconCss="e-icons e-setting-icon"
EnableRtl="true">Settings</SfButton>
<style>
.e-setting-icon::before {
content: '\e679';
}
</style>
```

Output be like



Set the disabled state in Blazor Button Component

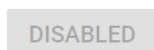
Button component can be enabled/disabled by giving [Disabled](#) property. To disable Button component, the `Disabled` property can be set as `true`.

The following example demonstrates Button in `Disabled` state.

ASPX-CS

```
@using Syncfusion.Blazor.Buttons
<SfButton Disabled="true">Disabled</SfButton>
```

Output be like



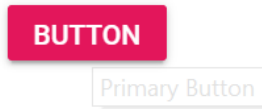
Tooltip for Button in Blazor Button Component

Tooltip can be shown on Button hover and it can be achieved by [HtmlAttributes](#) property.

CSHARP

```
@using Syncfusion.Blazor.Buttons
<SfButton Content="@Content" HtmlAttributes="@primButton"
IsPrimary="true"></SfButton>
@code {
public string Content = "Button";
private Dictionary<string, object> primButton = new Dictionary<string,
object> ()
{
{ "title", "Primary Button" }
};
}
```

Output be like



ButtonGroup

<!-- markdownlint-disable MD024 -->

Getting Started with Blazor ButtonGroup Component

This section briefly explains about how to include [ButtonGroup](#) Component in your Blazor server-side application. You can refer [Getting Started with Syncfusion Blazor for Server-side in Visual Studio page](#) for the introduction and configuring the common specifications.

Importing Syncfusion Blazor component in the application

1. Install the **Syncfusion.Blazor** NuGet package to the application by using the **NuGet Package Manager**.
2. You can add the client-side style resources through [CDN](#) or from [NuGet](#) package in the element of the `~/Pages/_Host.cshtml` page.

HTML

```
<head>
<link href="_content/Syncfusion.Blazor/styles/bootstrap4.css"
rel="stylesheet" />
@*<link
href="https://cdn.syncfusion.com/blazor/{{version}}/styles/{{theme}}.css"
rel="stylesheet" />*@
</head>
```

For Internet Explorer 11 kindly refer the polyfills. Refer the [documentation](#) for more information.

HTML

```
<head>
<link href="_content/Syncfusion.Blazor/styles/bootstrap4.css"
rel="stylesheet" />
<script
src="https://github.com/Daddoon/Blazor.Polyfill/releases/download/3.0.1/blaz
or.polyfill.min.js"></script>
</head>
```

Adding component package to the application

Open `/_Imports.razor` file and import the **Syncfusion.Blazor.SplitButtons** package.

ASPX-CS

```
@using Syncfusion.Blazor.SplitButtons
```

Add SyncfusionBlazor service in Startup.cs

Open the **Startup.cs** file and add services required by Syncfusion components. Add **services.AddSyncfusionBlazor()** method in the ConfigureServices function as follows.

CSHARP

```
namespace BlazorApplication
{
    public class Startup
    {
        ....
        ....
        public void ConfigureServices(IServiceCollection services)
        {
            ....
            ....
            services.AddSyncfusionBlazor();
        }
    }
}
```

To enable custom client side resource loading from CRG or CDN. You need to disable resource loading by **AddSyncfusionBlazor(true)** and load the scripts in the HEAD element of the **~/Pages/_Host.cshtml** page.

ASPX-CS

```
<head>
<environment include="Development">
<script src="https://cdn.syncfusion.com/blazor/{{ site.blazorversion
}}/syncfusion-blazor.min.js">
</script>
</environment>
</head>
```

Adding component package to the application

Open **/_Imports.razor** file and import the Syncfusion.Blazor.SplitButtons packages otherwise import these packages in the individual **razor** pages.

ASPX-CS

```
@using Syncfusion.Blazor
@using Syncfusion.Blazor.SplitButtons
```

Adding ButtonGroup component to the application

Now, add the Syncfusion Blazor ButtonGroup component in **razor** page in the **Pages** folder. For example, the ButtonGroup component is added in the **~/Pages/Index.razor** page.

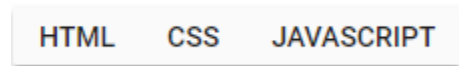
ASPX-CS

```
<SfButtonGroup>
<ButtonGroupButton>Left</ButtonGroupButton>
<ButtonGroupButton>Center</ButtonGroupButton>
<ButtonGroupButton>Right</ButtonGroupButton>
```

```
</SfButtonGroup>
```

Run the application

After successful compilation of your application, simply press F5 to run the application. The Blazor ButtonGroup component will render in the web browser as shown below



See Also

- [Getting Started with Syncfusion Blazor for Client-Side in .NET Core CLI](#)
- [Getting Started with Syncfusion Blazor for Server-Side in Visual Studio](#)
- [Getting Started with Syncfusion Blazor for Server-Side in .NET Core CLI](#)

Types and Styles in Blazor ButtonGroup Component

This section explains about different types and styles of ButtonGroup.

ButtonGroup styles

The Blazor ButtonGroup 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.

CSHARP

```
@using Syncfusion.Blazor.SplitButtons
<SfButtonGroup>
<ButtonGroupButton CssClass="e-primary">View</ButtonGroupButton>
<ButtonGroupButton CssClass="e-success">Edit</ButtonGroupButton>
<ButtonGroupButton CssClass="e-info">Delete</ButtonGroupButton>
</SfButtonGroup>
<SfButtonGroup>
<ButtonGroupButton CssClass="e-link">View</ButtonGroupButton>
<ButtonGroupButton CssClass="e-warning">Edit</ButtonGroupButton>
<ButtonGroupButton CssClass="e-danger">Delete</ButtonGroupButton>
</SfButtonGroup>
```

Output be like



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.

ButtonGroup types

The types of Blazor ButtonGroup are as follows:

- Flat ButtonGroup
- Outline ButtonGroup
- Round ButtonGroup
- Toggle ButtonGroup

Flat ButtonGroup

The Flat ButtonGroup is styled with no background color. To create a flat ButtonGroup, set the [CssClass](#) property to `e-flat`.

Outline ButtonGroup

An outline ButtonGroup has a border with transparent background. To create an outline ButtonGroup, set the [CssClass](#) property to `e-outline`.

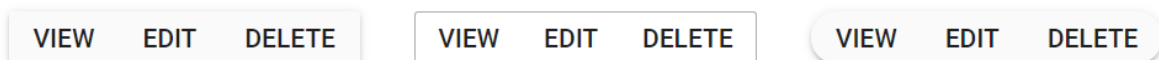
Round ButtonGroup

A round ButtonGroup is shaped like a circle. Usually, it contains an icon representing its action. To create a round ButtonGroup, set the [CssClass](#) property to `e-round-corner`.

ASPX-CS

```
@using Syncfusion.Blazor.SplitButtons
<SfButtonGroup CssClass="e-flat">
  <ButtonGroupButton>View</ButtonGroupButton>
  <ButtonGroupButton>Edit</ButtonGroupButton>
  <ButtonGroupButton>Delete</ButtonGroupButton>
</SfButtonGroup>
<SfButtonGroup CssClass="e-outline">
  <ButtonGroupButton CssClass="e-outline">View</ButtonGroupButton>
  <ButtonGroupButton CssClass="e-outline">Edit</ButtonGroupButton>
  <ButtonGroupButton CssClass="e-outline">Delete</ButtonGroupButton>
</SfButtonGroup>
<SfButtonGroup CssClass="e-round-corner">
  <ButtonGroupButton>View</ButtonGroupButton>
  <ButtonGroupButton>Edit</ButtonGroupButton>
  <ButtonGroupButton>Delete</ButtonGroupButton>
</SfButtonGroup>
```

Output be like



Icons

ButtonGroup with font icons

To create ButtonGroup with icons, [IconCss](#) property of the button component can be used. You can customize the icon's position by using the [IconPosition](#) property.

ASPX-CS

```
@using Syncfusion.Blazor.SplitButtons
<SfButtonGroup>
  <ButtonGroupButton IconCss="e-icons e-play-icon"
    IconPosition="IconPosition.Right">PLAY</ButtonGroupButton>
  <ButtonGroupButton IconCss="e-icons e-pause-icon">PAUSE</ButtonGroupButton>
  <ButtonGroupButton>Delete</ButtonGroupButton>
</SfButtonGroup>
<style>
.e-play-icon::before {
  content: '\e324';
}
.e-pause-icon::before {
  content: '\e326';
}
</style>
```

Output be like



ButtonGroup size

The two types of ButtonGroup sizes are default and small. To change the size of the default ButtonGroup to small ButtonGroup, set the [CssClass](#) property to `e-small`.

ASPX-CS

```
@using Syncfusion.Blazor.SplitButtons
<SfButtonGroup CssClass="e-small">
  <ButtonGroupButton>View</ButtonGroupButton>
  <ButtonGroupButton>Edit</ButtonGroupButton>
  <ButtonGroupButton>Delete</ButtonGroupButton>
</SfButtonGroup>
<SfButtonGroup>
  <ButtonGroupButton>View</ButtonGroupButton>
  <ButtonGroupButton>Edit</ButtonGroupButton>
  <ButtonGroupButton>Delete</ButtonGroupButton>
</SfButtonGroup>
```

Output be like



Selection and Nesting in Blazor ButtonGroup Component

Single selection

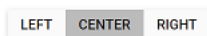
ButtonGroup supports single selection type in which only one button can be selected.

The following example illustrates the single selection behavior in ButtonGroup.

ASPX-CS

```
@using Syncfusion.Blazor.SplitButtons
<SfButtonGroup Mode="Syncfusion.Blazor.SplitButtons.SelectionMode.Single">
  <ButtonGroupButton>Left</ButtonGroupButton>
  <ButtonGroupButton @bind-Selected="@centerSelected">Center</ButtonGroupButton>
  <ButtonGroupButton>Right</ButtonGroupButton>
</SfButtonGroup>
@code {
  private bool centerSelected = true;
}
```

Output be like



Multiple selection

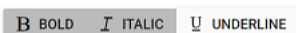
ButtonGroup supports multiple selection type in which multiple button can be selected.

The following example illustrates the multiple selection behavior in ButtonGroup.

ASPX-CS

```
@using Syncfusion.Blazor.SplitButtons
<SfButtonGroup Mode="Syncfusion.Blazor.SplitButtons.SelectionMode.Multiple">
  <ButtonGroupButton @bind-Selected="@boldSelected" IconCss="bg-icons e-btnggrp-bold">Bold</ButtonGroupButton>
  <ButtonGroupButton @bind-Selected="@italicSelected" IconCss="bg-icons e-btnggrp-italic e-icon-left">Italic</ButtonGroupButton>
  <ButtonGroupButton IconCss="bg-icons e-btnggrp-underline e-icon-left">Underline</ButtonGroupButton>
</SfButtonGroup>
@code {
  private bool boldSelected = true;
  private bool italicSelected = true;
}
```

Output be like



Nesting

Nesting with other components can be possible in ButtonGroup. The following components can be nested in ButtonGroup.

- DropDownButton
- SplitButton

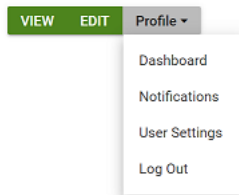
DropDownButton

In the following example, the DropDownButton component can be added in ButtonGroup tag.

ASPX-CS

```
@using Syncfusion.Blazor.SplitButtons
<SfButtonGroup>
<ButtonGroupButton CssClass="e-btn e-success">View</ButtonGroupButton>
<ButtonGroupButton CssClass="e-btn e-success">Edit</ButtonGroupButton>
<SfDropDownButton Content="Profile">
<DropDownMenuItem Text="Dashboard"></DropDownMenuItem>
<DropDownMenuItem Text="Notifications"></DropDownMenuItem>
<DropDownMenuItem Text="User Settings"></DropDownMenuItem>
<DropDownMenuItem Text="Log Out"></DropDownMenuItem>
</DropDownMenuItem>
</SfDropDownButton>
</SfButtonGroup>
```

Output be like



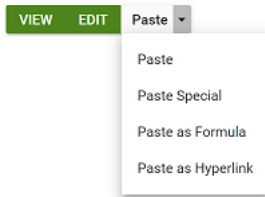
SplitButton

In the following example, SplitButton component can be added in ButtonGroup tag.

ASPX-CS

```
@using Syncfusion.Blazor.SplitButtons
<SfButtonGroup>
<ButtonGroupButton CssClass="e-btn e-success">View</ButtonGroupButton>
<ButtonGroupButton CssClass="e-btn e-success">Edit</ButtonGroupButton>
<SfSplitButton Content="Paste">
<DropDownMenuItem Text="Paste"></DropDownMenuItem>
<DropDownMenuItem Text="Paste Special"></DropDownMenuItem>
<DropDownMenuItem Text="Paste as Formula"></DropDownMenuItem>
<DropDownMenuItem Text="Paste as Hyperlink"></DropDownMenuItem>
</DropDownMenuItem>
</SfSplitButton>
</SfButtonGroup>
```

Output be like



Styles and Appearances in Blazor ButtonGroup Component

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

| CSS Class | Purpose of Class |

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

| .e-btn | To customize the ButtonGroup. |

| .e-btn:hover | To customize the ButtonGroup on hover. |

| .e-btn:focus | To customize the ButtonGroup on focus. |

| .e-btn:active | To customize the ButtonGroup on active. |

Calendar

Getting Started with Blazor Calendar Component

This section briefly explains about how to include a [Blazor Calendar](#) Component in your Blazor Server-Side and Client-Side application. You can refer to our Getting Started with [Blazor Server-Side Calendar](#) and [Blazor WebAssembly Calendar](#) documentation pages for configuration specifications.

To get start quickly with Blazor Calendar component, you can check on this video.

{% youtube

"youtube:https://www.youtube.com/watch?v=PbmIW_tWzVo"%}

Importing Syncfusion Blazor component in the application

- Install [Syncfusion.Blazor.Calendars](#) NuGet package to the application by using the **NuGet Package Manager**.
- You can add the client-side resources through [CDN](#) or from [NuGet](#) package in the **HEAD** element of the `~/wwwroot/index.html` page.

HTML

```
<head>
<link href="_content/Syncfusion.Blazor.Themes/bootstrap4.css"
rel="stylesheet" />
<!-- <link
href="https://cdn.syncfusion.com/blazor/{{version}}/styles/{{theme}}.css"
rel="stylesheet" /> -->
</head>
```

For Internet Explorer 11 kindly refer the polyfills. Refer the [documentation](#) for more information.

HTML

```
<head>
<link href="_content/Syncfusion.Blazor.Themes/bootstrap4.css"
rel="stylesheet" />
<script
src="https://github.com/Daddoon/Blazor.Polyfill/releases/download/3.0.1/blaz
or.polyfill.min.js"></script>
</head>
```

Adding component package to the application

Open `~/_Imports.razor` file and import the `Syncfusion.Blazor.Calendars` package.

ASPX-CS

```
@using Syncfusion.Blazor.Calendars
```

Add `SyncfusionBlazor` service in `Program.cs`

Open the `Program.cs` file and add services required by Syncfusion components using `builder.Services.AddSyncfusionBlazor()` method

CSHARP

```
using Syncfusion.Blazor;
namespace BlazorApplication
{
    public class Program
    {
        public static async Task Main(string[] args)
        {
            ....
            ....
            builder.Services.AddSyncfusionBlazor();
            await builder.Build().RunAsync();
        }
    }
}
```

To enable custom client side resource loading from CRG or CDN. You need to disable resource loading by `AddSyncfusionBlazor(true)` and load the scripts in the **HEAD** element of the `~/wwwroot/index.html` page.

HTML

```
<head>
<script src="https://cdn.syncfusion.com/blazor/{{ site.blazorversion
}}/syncfusion-blazor.min.js"></script>
</head>
```

Adding Calendar component to the application

To initialize the Calendar component add the below code to your `Index.razor` view page which is present under `~/Pages` folder.

The following code shows a basic Calendar component.

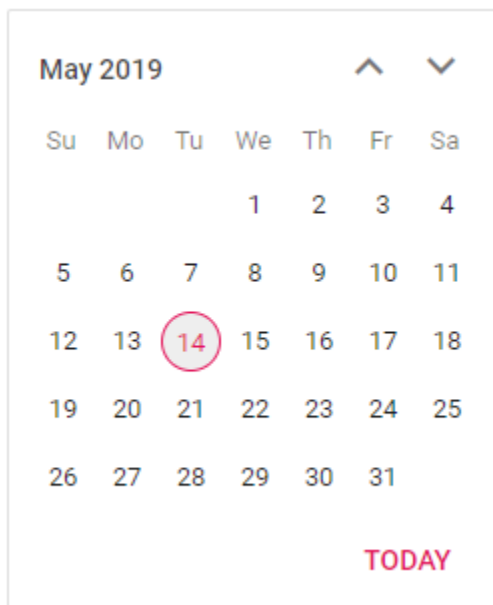
ASPX-CS

```
<SfCalendar TValue="DateTime"></SfCalendar>
```

Run the application

After successful compilation of your application, press **F5** to run the application.

The output will be as follows.



Setting the Value, Min, and Max dates

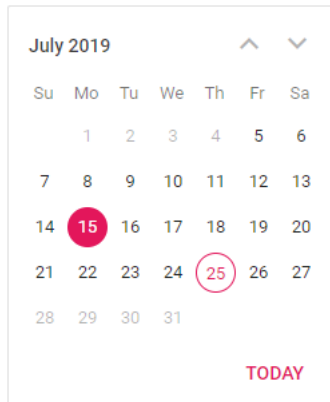
Calendar component provides an option to select a date value within a specified range by defining the [Min](#) and [Max](#) properties. Also, you can set a date value within specific range using the [Value](#) property. For more information, you can refer to the [Date Range](#) section.

Here, the Calendar allows you to select a date within the range from 5th to 27th of this month. `TValue` specifies the type of the DatePicker component.

ASPX-CS

```
<SfCalendar TValue="DateTime" Min='@MinDate' Value='@DateValue'
Max='@MaxDate'></SfCalendar>
@code{
public DateTime MinDate {get;set;} = new DateTime(DateTime.Now.Year,
DateTime.Now.Month, 05);
public DateTime MaxDate {get;set;} = new DateTime(DateTime.Now.Year,
DateTime.Now.Month, 27);
public DateTime DateValue {get;set;} = new DateTime(DateTime.Now.Year,
DateTime.Now.Month, 15);
}
```

The output will be as follows.



You can also explore our [Blazor Calendar example](#) that shows how to configure the calendar in Blazor.

See Also

- [Getting Started with Syncfusion Blazor for Client-Side in .NET Core CLI](#)
- [Getting Started with Syncfusion Blazor for Server-side in Visual Studio](#)
- [Getting Started with Syncfusion Blazor for Server-Side in .NET Core CLI](#)

Data Binding in Blazor Calendar Component

This section briefly explains how to bind the value to the Calendar component in the below different ways.

- One-Way Data Binding
- Two-Way Data Binding
- Dynamic Value Binding

One-Way Binding

We can bind the value to the Calendar component directly for **Value** property as mentioned in the following code example. In one-way binding, we need to pass property or variable name along with **@** (For Ex: "@DateValue").

ASPX-CS

```
@using Syncfusion.Blazor.Calendars
<SfCalendar TValue="DateTime?" Value="@DateValue"></SfCalendar>
<button @onclick="@UpdateValue">Update Value</button>
@code {
    public DateTime? DateValue {get;set;} = new DateTime(DateTime.Now.Year,
    DateTime.Now.Month, 28);
    public void UpdateValue()
    {
        DateValue = DateTime.Now;
    }
}
```

Two-Way Data Binding

Two-way binding can be achieved by using `bind-Value` attribute and its supports string, int, Enum, DateTime, bool types. If component value has been changed, it will affect the all places where we bind the variable for the `bind-value` attribute.

ASPX-CS

```
@using Syncfusion.Blazor.Calendars
<p>Calendar value is: @DateValue</p>
<SfCalendar TValue="DateTime?" @bind-Value="@DateValue"></SfCalendar>
@code {
    public DateTime? DateValue { get; set; } = DateTime.Now;
}
```

Dynamic Value Binding

We can change the property value dynamically by manually calling the `StateHasChanged()` method inside public event of **Blazor Calendar component** only. This method notifies the component that its state has changed and queues a re-render.

There is no need to call this method for native events since it's called after any lifecycle method has been called and can also be invoked manually to trigger a re-render. Please refer the below mentioned code example.

ASPX-CS

```
@using Syncfusion.Blazor.Calendars
<p>Calendar value is: @DateValue</p>
<SfCalendar TValue="DateTime?" Value="@DateValue">
    <CalendarEvents TValue="DateTime?" ValueChange="@onChange"></CalendarEvents>
</SfCalendar>
@code {
    public DateTime? DateValue { get; set; } = DateTime.Now;
    private void
    onChange(Syncfusion.Blazor.Calendars.ChangedEventArgs<DateTime?> args)
    {
        DateValue = args.Value;
        StateHasChanged();
    }
}
```

Data Range in Blazor Calendar Component

A calendar provides an option to select a date value within a specified range by defining the [Min](#) and [Max](#) properties. The Min date should always be lesser than the Max date.

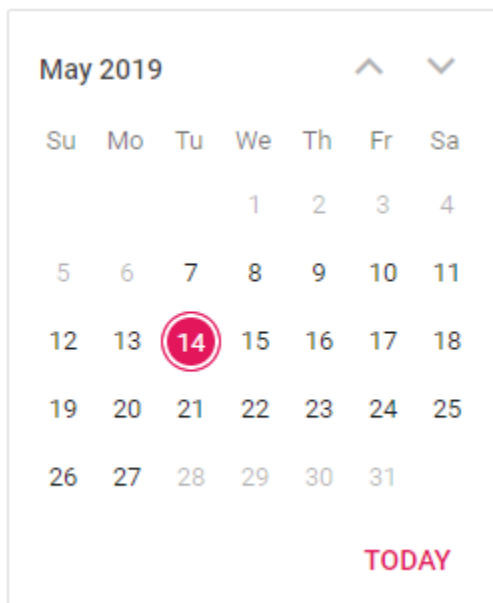
- If the value of `Min` or `Max` properties are changed through code behind, then update the `Value` property to be set within the specified range.
- If the value is out of specified date range and less than Min date, the `Value` property will be updated with Min date or the value is higher than Max date, the `Value` property will be updated with Max date.

The following code allows you to select a date within the range of 7th to 27th days in a month.

ASPX-CS

```
@using Syncfusion.Blazor.Calendars
<SfCalendar TValue="DateTime?" Min="@MinDate" Max="@MaxDate"
Value="@DateValue"></SfCalendar>
@code{
public DateTime MinDate {get;set;} = new
DateTime(DateTime.Now.Year,DateTime.Now.Month,07);
public DateTime MaxDate {get;set;} = new DateTime(DateTime.Now.Year,
DateTime.Now.Month, 27);
public DateTime? DateValue {get;set;} = new DateTime(DateTime.Now.Year,
DateTime.Now.Month, 14);
}
```

The output will be as follows.

**Multi Selection in Blazor Calendar Component**

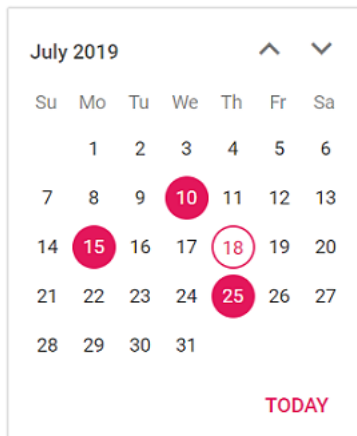
A calendar provides an option to select **single** or **multiple dates** by using the [IsMultiSelection](#) properties. By default, the IsMultiSelection property will be in disabled state.

The following code demonstrates the functionality of IsMultiSelection and Values properties in the Calendar component.

ASPX-CS

```
@using Syncfusion.Blazor.Calendars
<SfCalendar TValue="DateTime?" IsMultiSelection=true
Values="@MultipleValues"></SfCalendar>
@code {
public DateTime[] MultipleValues { get; set; } = new DateTime[] { new
DateTime(DateTime.Now.Year, DateTime.Now.Month, 10),
new DateTime(DateTime.Now.Year, DateTime.Now.Month, 15),
new DateTime(DateTime.Now.Year, DateTime.Now.Month, 25) };
}
```


The output will be as follows.



Calendar Views in Blazor Calendar Component

A Calendar has the following predefined views that provide a flexible way to navigate back and forth when selecting dates.

| **View** | **Description** |

| --- | --- |

| Month (default) | Displays the days in a month. |

| Year | Displays the months in a year. |

| Decade | Displays the years in a decade. |

Set the initial view

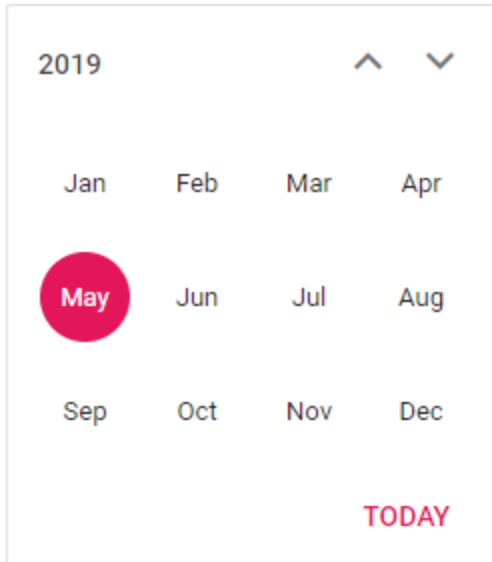
When view is defined to the [Start](#) property of the Calendar, it allows you to set the initial view on rendering.

The following example demonstrates how to set the **Year** as the start view of the Calendar.

ASPX-CS

```
@using Syncfusion.Blazor.Calendars
<SfCalendar TValue="DateTime?" Value="@DateValue"
Start="CalendarView.Year"></SfCalendar>
@code {
    public DateTime DateValue {get;set;} = DateTime.Now;
}
```

The output will be as follows.



View Restriction

By defining the Start and Depth property with the different view, drill-down and drill-up views navigation can be limited to the users. Calendar views will be drill-down upto the view which is set in [Depth](#) property and drill-up upto the view which is set in [Start](#) property.

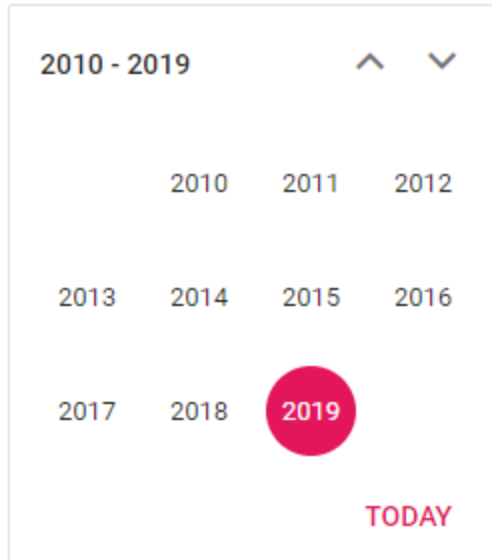
The following example displays the Calendar in **Decade** view, and allows you to select a date in **Month** view.

Depth view should always be smaller than the Start view. If the **Depth** and **Start** views are the same, then the Calendar view remains unchanged.

ASPX-CS

```
@using Syncfusion.Blazor.Calendars
<SfCalendar TValue="DateTime?" Value="@DateValue"
Start="CalendarView.Decade" Depth="CalendarView.Year"></SfCalendar>
@code
{
    public DateTime DateValue {get;set;} = DateTime.Now;
}
```

The output will be as follows.



Accessibility in Blazor Calendar Component

The web accessibility makes web content and web applications more accessible for disabled people. It especially helps in dynamic content change and development of advanced user interface components with AJAX, HTML, JavaScript, and related technologies.

Calendar provides built-in compliance with [WAI-ARIA](#) specifications. WAI-ARIA support is achieved through attributes like `aria-label`, `aria-selected`, `aria-disabled`, and `aria-activedescendant` applied for navigation buttons, and disable and active day cells.

It helps disabled persons by providing information about the widget for assistive technology in the screen readers. Calendar component contains grid role and grid cell for each day cell.

- **aria-label:** Provides text labels for an object for the previous and next month's elements. It helps the screen reader object to read.
- **aria-selected:** Indicates the currently selected date of the Calendar component.
- **aria-disabled:** Indicates the disabled state of the Calendar component.
- **aria-activedescendant:** Helps in managing the current active child of the Calendar component.
- **role:** Gives information to assistive technologies about how to handle each element in a widget.
- **grid-cell:** Defines the individual cell that can be focussed and selected.

Keyboard interaction

You can use the following keys to interact with the Calendar. This control implements keyboard navigation support by following the [WAI-ARIA practices](#).

It supports the following list of shortcut keys:

Keys	Description
------	-------------

--- ---	
-----------	--

Upper Arrow	Focuses the same day of the previous week.
-------------	--

Down Arrow	Focuses the same day of the next week.
------------	--

Left Arrow	Focuses the day before.
------------	-------------------------

- | Right Arrow | Focuses the next day. |
- | Home | Focuses the first day of the month. |
- | End | Focuses the last day of the month. |
- | Page Up | Focuses the same date of the previous month. |
- | Page Down | Focuses the same date of the next month. |
- | Enter | Selects the currently focused date. |
- | Shift + Page Up | Focuses the same date for the previous year. |
- | Shift + Page Down | Focuses the same date for the next year. |
- | Control + Upper Arrow | Moves to the inner level of view like month to year and year to decade. |
- | Control + Down Arrow | Moves out from the depth level view like decade to year and year to month. |
- | Control + Home | Focuses the first date of the current year. |
- | Control + End | Focuses the last date of the current year. |

Globalization in Blazor Calendar Component

Globalization is the combination of adapting the control to various languages by means of parsing and formatting the date or number **Internationalization** and also by adding cultural specific customizations and translating the text **localization**.

Blazor server side

Add **UseRequestLocalization** middle-ware in Configure method in **Startup.cs** file to get browser Culture Info.

Refer the following code to add configuration in Startup.cs file

C#

```
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Localization;
namespace BlazorApplication
{
    public class Startup
    {
        ....
        ....
        public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
        {
            app.UseRequestLocalization();
            ....
            ....
        }
    }
}
```

The **Localization** library allows you to localize default text content. The Calendar component has static text that can be changed to other cultures (Arabic, Deutsch, French, etc.).

In the following examples, demonstrate how to enable **Localization** for Calendar in server side Blazor samples. Here, we have used Resource file to translate the static text.

The Resource file is an XML file which contains the strings(key and value pairs) that you want to translate into different language. You can also refer Localization [link](#) to know more about how to configure and use localization in the ASP.NET Core application framework.

- Open the **Startup.cs** file and add the below configuration in the **ConfigureServices** function as follows.

CSHARP

```
using Syncfusion.Blazor;
using System.Globalization;
using Microsoft.AspNetCore.Localization;
namespace BlazorApplication
{
    public class Startup
    {
        ....
        ....
        public void ConfigureServices(IServiceCollection services)
        {
            ....
            ....
            services.AddSyncfusionBlazor();
            services.AddLocalization(options => options.ResourcesPath = "Resources");
            services.Configure<RequestLocalizationOptions>(options =>
            {
                // define the list of cultures your app will support
                var supportedCultures = new List<CultureInfo>()
                {
                    new CultureInfo("de")
                };
                // set the default culture
                options.DefaultRequestCulture = new RequestCulture("de");
                options.SupportedCultures = supportedCultures;
                options.SupportedUICultures = supportedCultures;
                options.RequestCultureProviders = new List<IRequestCultureProvider>() {
                    new QueryStringRequestCultureProvider() // Here, You can also use other
                    localization provider
                };
            });
            services.AddSingleton(typeof(ISyncfusionStringLocalizer),
                typeof(SampleLocalizer));
        }
    }
}
```

- Then, write a **class** by inheriting **ISyncfusionStringLocalizer** interface and override the **Manager** property to get the resource file details from the application end.

CSHARP

```
using Syncfusion.Blazor;
namespace blazorCalendars
{
    public class SampleLocalizer : ISyncfusionStringLocalizer
    {
        public string Get(string key)
        {
            return this.Manager.GetString(key);
        }
        public System.Resources.ResourceManager Manager
        {
            get
            {
                return blazorCalendars.Resources.SyncfusionBlazorLocale.ResourceManager;
            }
        }
    }
}
```

- Add **.resx** file to Resource folder and enter the key value (Locale Keywords) in the **Name** column and the translated string in the Value column as follows.

Name	Value (in Deutsch culture)
---	---
Calendar_Today	Heute

- Finally, Specify the culture for Calendar using **locale** property.

ASPX-CS

```
@using Syncfusion.Blazor.Calendars
<SfCalendar TValue="DateTime" Locale="de"> </SfCalendar>
```

Blazor WebAssembly

By default, the Calendar week and month names are specific to the **American English** culture. It utilizes the **Blazor Internationalization** package to parse and format the date object based on the culture by using the official [UNICODE CLDR](#) JSON data.

The following steps explain how to render the Calendar in German culture ('de-DE') in Blazor Web Assembly application.

- Open the **program.cs** file and add the below configuration in the **Builder ConfigureServices** function as follows.

CSHARP

```
using Syncfusion.Blazor;
using Microsoft.AspNetCore.Builder;
```

```

namespace WebAssemblyLocale
{
    public class Program
    {
        public static async Task Main(string[] args)
        {
            ....
            ....
            builder.Services.Configure<RequestLocalizationOptions>(options =>
            {
                // Define the list of cultures your app will support
                var supportedCultures = new List<System.Globalization.CultureInfo>()
                {
                    new System.Globalization.CultureInfo("en-US"),
                    new System.Globalization.CultureInfo("de"),
                };
                // Set the default culture
                options.DefaultRequestCulture = new
                Microsoft.AspNetCore.Localization.RequestCulture("de");
                options.SupportedCultures = supportedCultures;
                options.SupportedUICultures = supportedCultures;
                options.RequestCultureProviders = new
                List<Microsoft.AspNetCore.Localization.IRequestCultureProvider>() {
                    new Microsoft.AspNetCore.Localization.QueryStringRequestCultureProvider()
                };
            });
            ....
            ....
        }
    }
}

```

- Download the required locale packages to render the Blazor Calendar component with specified locale.
- To download the locale definition of Blazor components, use this [link](#).
- After downloading the `blazor-locale` package, copy the `blazor-locale` folder with required local definition file into `wwwroot` folder.
- By default, the `blazor-locale` package contains the localized text for static text present in components like button text, placeholder, tooltip, and more.
- Set the culture by using the `SetCulture` method.

ASPX-CS

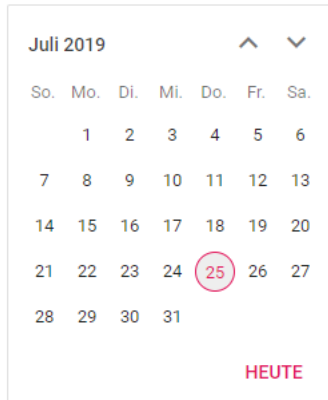
```

@using Syncfusion.Blazor.Calendars
@inject HttpClient Http;
<SfCalendar TValue="DateTime?" Locale="de"></SfCalendar>
@code {
    [Inject]
    protected IJSRuntime JsRuntime { get; set; }
    protected override async Task OnInitializedAsync()
    {
        this.JsRuntime.Sf().LoadLocaleData(await Http.GetJsonAsync<object>("blazor-locale/src/de.json")).SetCulture("de");
    }
}

```

```
}
}
```

The output will be as follows.



Customize the localized text

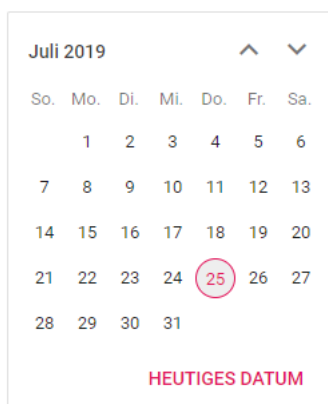
- You can change the localized text of particular component by editing the `wwwroot/blazor-locale/src/{{locale name}}.json` file.
- In the following code, modified the localized text of `today button` in `de` culture.

[`wwwroot/blazor-locale/src/de.json`]

XML

```
{
  "de": {
    "calendar": {
      "today": "Heutiges Datum"
    }
  }
}
```

The output will be as follows.



Right-To-Left

The Calendar supports RTL (right-to-left) functionality for languages like Arabic and Hebrew to display the text in the right-to-left direction. Use the [EnableRtl](#) property to set the RTL direction.

The following code example initializes the Calendar component in Arabic culture.

ASPX-CS

```
@using Syncfusion.Blazor.Calendars
@inject HttpClient Http;
<SfCalendar TValue="DateTime?" Locale="ar" EnableRtl=true></SfCalendar>
@code {
    [Inject]
    protected IJSRuntime JsRuntime { get; set; }
    protected override async Task OnInitializedAsync()
    {
        this.JsRuntime.Sf().LoadLocaleData(await Http.GetJsonAsync<object>("blazor-locale/src/ar.json")).SetCulture("ar");
    }
}
```

The output will be as follows.



Events in Blazor Calendar Component

This section explains the list of events of the Calendar component which will be triggered for appropriate Calendar actions.

From v17.2, added only the limited number of events for the Calendar component. The event names are different from the previous releases and also removed several events. The following are the event name changes from v17.1. to v17.2.*

Event Name(v17.1.) /Event Name(v17.2.)

change | [ValueChange](#)

renderDayCell | [OnRenderDayCell](#)

[OnRenderDayCell](#)

onRenderDayCellHandler event triggers when each day cell of the Calendar is rendered.

ASPX-CS

```
@using Syncfusion.Blazor.Calendars
<SfCalendar TValue="DateTime">
  <CalendarEvents TValue="DateTime?"
    OnRenderDayCell="onRenderDayCellHandler"></CalendarEvents>
</SfCalendar>
@code{
public void onRenderDayCellHandler(RenderDayCellEventArgs args)
{
  // Here you can customize your code
}
```

[ValueChange](#)

ValueChange event triggers when the Calendar value is changed.

ASPX-CS

```
@using Syncfusion.Blazor.Calendars
<SfCalendar TValue="DateTime">
  <CalendarEvents TValue="DateTime?"
    ValueChange="ValuechangeHandler"></CalendarEvents>
</SfCalendar>
@code{
public void ValuechangeHandler(ChangedEventArgs<DateTime?> args)
{
  // Here you can customize your code
}
```

[Created](#)

Created event triggers when Calendar is created.

ASPX-CS

```
@using Syncfusion.Blazor.Calendars
<SfCalendar TValue="DateTime">
  <CalendarEvents TValue="DateTime?"
    Created="createdHandler"></CalendarEvents>
</SfCalendar>
@code{
public void createdHandler(object args)
{
  // Here you can customize your code
}
```

Destroyed

OnOpen event triggers when Calendar is destroyed.

ASPX-CS

```

@using Syncfusion.Blazor.Calendars
<SfCalendar TValue="DateTime">
  <CalendarEvents TValue="DateTime?"
    Destroyed="DestroyHandler"></CalendarEvents>
</SfCalendar>
@code{
public void DestroyHandler(object args)
{
    // Here you can customize your code
}
}

```

Navigated

Navigated event triggers when the Calendar is navigated to another level or within the same level of view.

ASPX-CS

```

@using Syncfusion.Blazor.Calendars
<SfCalendar TValue="DateTime">
  <CalendarEvents TValue="DateTime?"
    Navigated="NavigatedHandler"></CalendarEvents>
</SfCalendar>
@code{
public void NavigatedHandler(NavigatedEventArgs args)
{
    // Here you can customize your code
}
}

```

Calendar will be limited with these events and new events will be added in future based on the user requests. If the event you are looking for is not in the list, then please request [here](#).

Special Dates in Blazor Calendar Component

You can customize specific dates in a calendar by using the [OnRenderDayCell](#) event. This event gets triggered on each day cell element creation that allows you to customize or disable the specific dates in the calendar. Here, list of dates in the current month are customized with custom styles by adding the personal-appointment and official-appointment class.

ASPX-CS

```

@using Syncfusion.Blazor.Calendars
<div class="control-wrapper">
  <SfCalendar TValue="DateTime?" @bind-Value="@SelectedDate">
    <CalendarEvents TValue="DateTime?" OnRenderDayCell="CustomDates"
      ValueChange="OnChange"></CalendarEvents>
  </SfCalendar>
</div>
<div id="display-date">
  <span>Selected Day : @SelectedValue</span>

```

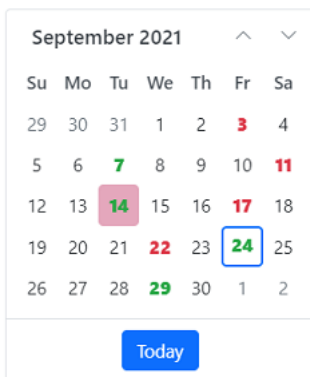
```
</div>
@code {
public DateTime? SelectedDate { get; set; }
public string SelectedValue { get; set; } =
DateTime.Now.ToString("M/d/yyyy");
public DateTime? CurrentDate { get; set; } = DateTime.Now;
public void CustomDates(RenderDayCellEventArgs args)
{
var CurrentMonth = CurrentDate.Value.Month;
if (args.Date.Month == CurrentMonth && (args.Date.Day == 7 || args.Date.Day
== 14 || args.Date.Day == 24 || args.Date.Day == 29)) {
args.CellData.ClassList += " personal-appointment";
}
if (args.Date.Month == CurrentMonth && (args.Date.Day == 3 || args.Date.Day
== 11 || args.Date.Day == 17 || args.Date.Day == 22))
{
args.CellData.ClassList += " official-appointment";
}
}
public void OnChange(ChangedEventArgs<DateTime?> args)
{
var Count = 0;
var CurrentMonth = CurrentDate.Value.Month;
if (args.Value.Value.Month == CurrentMonth && (args.Value.Value.Day == 7 ||
args.Value.Value.Day == 14 || args.Value.Value.Day == 24 ||
args.Value.Value.Day == 29))
{
this.SelectedValue = this.SelectedDate?.ToString("M/d/yyyy") + " (Personal
appointment)";
Count++;
}
if (args.Value.Value.Month == CurrentMonth && (args.Value.Value.Day == 3 ||
args.Value.Value.Day == 11 || args.Value.Value.Day == 17 ||
args.Value.Value.Day == 22))
{
this.SelectedValue = this.SelectedDate?.ToString("M/d/yyyy") + " (Official
appointment)";
Count++;
}
if (Count == 0)
{
this.SelectedValue = this.SelectedDate?.ToString("M/d/yyyy");
}
}
}
<style>
#display-date {
max-width: 300px;
margin: 0 auto;
padding: 15px 0;
font-size: 13px;
}
.control-wrapper {
width: 300px;
margin: 0 auto;
padding-top: 20px;
}
}
```

```

.e-calendar .e-content .e-cell.personal-appointment span.e-day,
.e-calendar .e-content td:hover.e-cell.personal-appointment span.e-day,
.e-calendar .e-content td.e-selected.e-focused-date.e-cell.personal-
appointment span.e-day {
color: #28a745;
font-weight: 800;
}
.e-calendar .e-content .e-cell.official-appointment span.e-day,
.e-calendar .e-content td:hover.e-cell.official-appointment span.e-day,
.e-calendar .e-content td.e-selected.e-focused-date.e-cell.official-
appointment span.e-day {
color: #dc3545;
font-weight: 800;
}
.e-calendar .e-content td.e-selected.e-focused-date.e-cell.personal-
appointment span.e-day,
.e-calendar .e-content td.e-selected.e-focused-date.e-cell.official-
appointment span.e-day {
background-color: #b511485e;
}
</style>

```

The output will be as follows.



Selected Day : 9/14/2021 (Personal appointment)

How To

Show Dates of Other Months in Blazor Calendar Component

The following code demonstrates how to show dates of other months. Using the styles below, you can bring the dates of other months to visibility from its hidden state.

ASPX-CS

```

@using Syncfusion.Blazor.Calendars
<SfCalendar TValue="DateTime?"></SfCalendar>
<style>
.e-control.e-calendar {
max-width: 260px;
}
.e-calendar .e-content tr.e-month-hide,
.e-calendar .e-content td.e-other-month>span.e-day {
display: block;
}

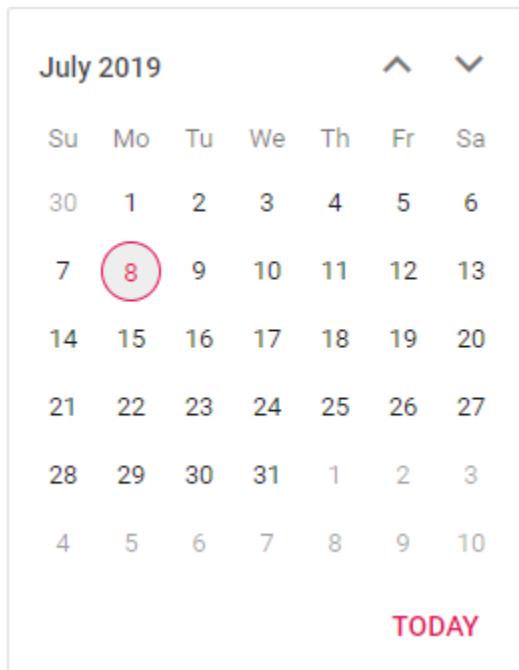
```

```

}
.e-calendar .e-content td.e-month-hide,
.e-calendar .e-content td.e-other-month {
pointer-events: auto;
touch-action: auto;
}
</style>

```

The output will be as follows.



Week Number in Blazor Calendar Component

You can enable **WeekNumber** in the Calendar by using the [WeekNumber](#) property.

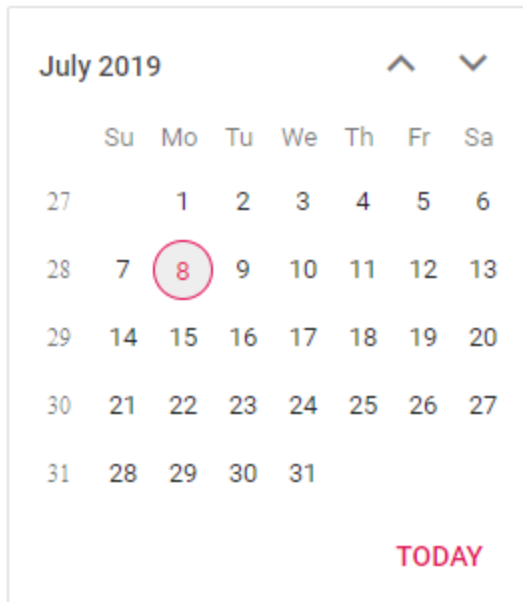
ASPX-CS

```

@using Syncfusion.Blazor.Calendars
<SfCalendar TValue="DateTime?" WeekNumber=true></SfCalendar>

```

The output will be as follows.



Week Rule

You can enable **WeekRule** in the Calendar by using the [WeekRule](#) property. This property provide an option to specify the rule for defining the first week of the year. Please find the possible values of **WeekRule** property.

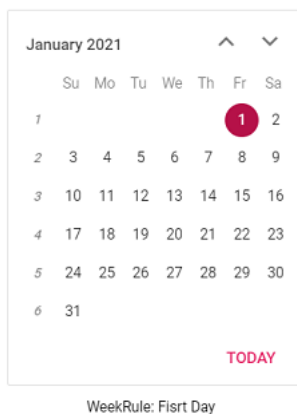
Types | Description

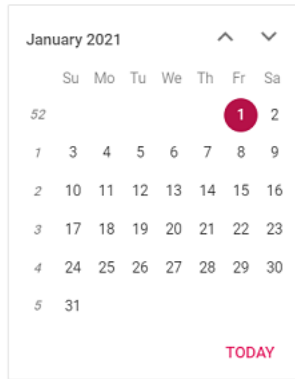
FirstDay | Set the first week of the year's week number to be started from 1. Then it followed as 1, 2, 3 ...

FirstFullWeek | Set the first week of the year's week number to be started from 52 or 53 (i.e December last week's week Number). Then it followed as 53, 1, 2 ...

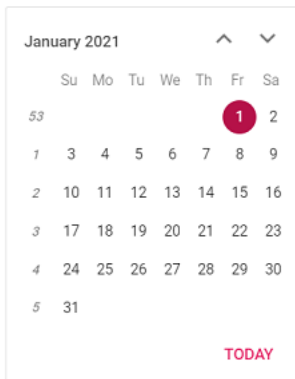
FirstFourDayWeek | Set the week number based on the majority of dates present in the week for the respected months. If January dates are presented in the week more than December, the first week of the year's week number will be started from 1. If December dates are presented in the week more than January, the first week of the year's week number will be started from 52 or 53.

The output will be as follows.





WeekRule: First Full Week



WeekRule: First Four Day Week

Change the First Day of Week in Blazor Calendar Component

The Calendar provides an option to change the first day of the week by using the [FirstDayOfWeek](#) property. Generally, the day of the week starts from 0 (Sunday) and ends with 6 (Saturday).

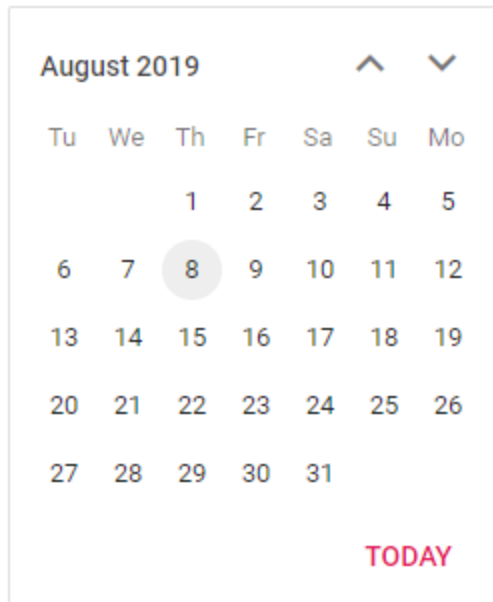
By default, the first day of the week is culture specific.

The following code shows the Calendar with **Tuesday** as the first day of the week.

ASPX-CS

```
@using Syncfusion.Blazor.Calendars
<SfCalendar TValue="DateTime?" FirstDayOfWeek=2></SfCalendar>
```

The output will be as follows.



Card

<!-- markdownlint-disable MD040 -->

Getting Started with Blazor Card Component

This section briefly explains about how to include a [Card](#) in your Blazor server-side application. You can refer to our Getting Started with [Syncfusion Blazor for Server-Side in Visual Studio 2019](#) page for the introduction and configuring the common specifications.

To get start quickly with Blazor Card component, you can check on this video.

{% youtube

"youtube:https://www.youtube.com/watch?v=k8KSZIf5VPs"%}

Importing Syncfusion Blazor component in the application

1. Install **Syncfusion.Blazor.Card** NuGet package to the application by using the **NuGet Package Manager**.
2. You can add the client-side style resources through [CDN](#) or from [NuGet](#) package in the **HEAD** element of the `~/Pages/_Host.cshtml` page.

ASPX-CS

```
<head>
<environment include="Development">
  ....
  ....
  <link href="_content/Syncfusion.Blazor/styles/fabric.css" rel="stylesheet"
  />
  <!--CDN-->
  @*<link href="https://cdn.syncfusion.com/blazor/18.4.42/styles/fabric.css"
  rel="stylesheet" />*@
</environment>
</head>
```

For Internet Explorer 11 kindly refer the polyfills. Refer the [documentation](#) for more information.

ASPX-CS

```
<head>
<environment include="Development">
<link href="_content/Syncfusion.Blazor/styles/fabric.css" rel="stylesheet"
/>
<script
src="https://github.com/Daddoon/Blazor.Polyfill/releases/download/3.0.1/blaz
or.polyfill.min.js"></script>
</environment>
</head>
```

Adding component package to the application

Open `~/_Imports.razor` file and import the **Syncfusion.Blazor**.

ASPX-CS

```
@using Syncfusion.Blazor.Cards
```

Add SyncfusionBlazor service in Startup.cs

Open the **Startup.cs** file and add services required by Syncfusion components using **service.AddSyncfusionBlazor()** method. Add this method in the **ConfigureServices** function as follows.

CSHARP

```
using Syncfusion.Blazor;
namespace BlazorApplication
{
    public class Startup
    {
        ....
        ....
        public void ConfigureServices(IServiceCollection services)
        {
            ....
            ....
            services.AddSyncfusionBlazor();
        }
    }
}
```

Adding Card component

To initialize the Card component, add the below code to your **Index.razor** view page which is present under `~/Pages` folder.

ASPX-CS

```
<SfCard> Sample Card </SfCard>
```

Adding a header and content

1. You can create Card with a header in a specific structure. For adding header you can use `CardHeader` tag and in that `Title` and `SubTitle` can be given.
2. Also content will be added by using `CardContent` tag.

ASPX-CS

```
@using Syncfusion.Blazor.Cards
<div class="control-section">
<div class="row">
<div class="col-xs-6 col-sm-6 col-lg-6 col-md-6">
<SfCard>
<CardHeader Title="Debunking Five Data Science Myths" SubTitle="By John Doe
| Jan 20, 2018" />
<CardContent Content="Tech evangelists are currently pounding their pulpits
about all things AI, machine learning, analytics—anything that sounds like
the future and probably involves lots of numbers. Many of these topics can
be grouped under the intimidating term data science." />
</SfCard>
</div>
</div>
</div>
```

Run the application

After successful compilation of your application, the Syncfusion Blazor Card component will render in the web browser.

Debunking Five Data Science Myths

By John Doe | Jan 20, 2018

Tech evangelists are currently pounding their pulpits about all things AI, machine learning, analytics—anything that sounds like the future and probably involves lots of numbers. Many of these topics can be grouped under the intimidating term data science.

<!-- markdownlint-disable MD036 -->

Header and Content in Blazor Card Component

Header

The Card can be created with header title, sub title and images. For adding header you need to add `CardHeader` Component. Card provides below elements and corresponding class definitions to include header.

Elements | Description

`Title` | Main title text with in the header.

SubTitle | A sub-title within the header.

CardImage | To include heading image within the header.

Title and Subtitle

For adding header to the Card, Title Property.

- Add **Title** Property inside the header caption for adding main title.
- Add **SubTitle** Property inside the header caption element for adding Title.

Image

Card header has an option for adding images in the header. It is aligned with either before or after the header based on the HTML element positioned in the header structure. The header image can be added by **ImageUrl** component which can be placed before or after the header caption wrapper element.

ASPX-CS

```
@using Syncfusion.Blazor.Cards
<SfCard ID="HugeImage">
  <CardHeader Title="Laura Callahan" SubTitle="Sales Coordinator and
  Representative" ImageUrl="images/cards/football.png" />
</SfCard>
<SfCard ID="SecondCard">
  <CardHeader Title="Laura Callahan" SubTitle="Sales Coordinator and
  Representative" ImageUrl="images/cards/football.png" />
</SfCard>
```

Content

Content in Card holds texts, images, links and all possible HTML elements. Its adaptable within the Card root element.

- Create a **Content** component.
- Place content **div** element in the Card root element or within any Card inner elements.

ASPX-CS

```
@using Syncfusion.Blazor.Cards
<SfCard ID="HugeImage">
  <CardHeader Title="Laura Callahan" SubTitle="Sales Coordinator and
  Representative" ImageUrl="images/cards/football.png" />
</SfCard>
<SfCard ID="SecondCard">
  <CardContent Content="Laura received a BA in psychology from the University
  of Washington. She has also completed a course in business French. She reads
  and writes French." />
</SfCard>
```

<!-- markdownlint-disable MD036 -->

Image and Divider in Blazor Card Component

Images

The Card supports to include images within the elements, you can add image as direct element anywhere inside card root by adding the `CardImage` component. Using the class defined, you can write CSS styles to load images to that element.

By default, card images occupies full width of its parent element.

ASPX-CS

```
@using Syncfusion.Blazor.Cards
<SfCard>
  <CardImage/>
</SfCard>
```

Title

Card image is supported to include a `Title` property for the image. By default, Title is placed over the image on left-bottom position with overlay.

ASPX-CS

```
@using Syncfusion.Blazor.Cards
<SfCard>
  <CardHeader Title="JavaScript"></CardHeader>
  <CardContent>
    JavaScript Succinctly was written to give readers an accurate, concise
    examination
    of JavaScript objects and their supporting nuances, such as complex values,
    primitive
    values scope, inheritance, the head object, and more.
  </CardContent>
</SfCard>
```

Divider

Divider used to separate the elements inside the card. You can add divider inside the card elements to separate it. Set `EnableSeparator` property to `true` in card content for adding a divider.

ASPX-CS

```
@using Syncfusion.Blazor.Cards
<SfCard>
  <CardHeader Title="Explore Cities"></CardHeader>
  <CardContent EnableSeparator="true">
    Sydney is a city on the east coast of Australia. Sydney is the capital city
    of New South
    Wales. About four million people live in Sydney which makes it the biggest
    city in Oceania.
  </CardContent>
  <CardContent EnableSeparator="true">
    New York City has been described as the cultural, financial, and media
    capital of the
    world, and exerts a significant impact upon commerce and etc.
  </CardContent>
  <CardContent EnableSeparator="true">
```

```
Malaysia is one of the Southeast Asian countries, on a peninsula of the
Asian continent,
to a certain extent; it can be recognized as part of the Asian continent.
</CardContent>
</SfCard>
```

Action Buttons in Blazor Card Component

You can include Action buttons within the Card and customize them. Action button is a `div` element with `CardFooter` component followed by button tag or anchor tag within the card root element.

For adding action buttons you can create a `CardFooterContent` component within the card action element.

ASPX-CS

```
@using Syncfusion.Blazor.Cards
<SfCard ID="HugeImage">
<CardFooter>
<CardFooterContent>
</CardFooterContent>
</CardFooter>
</SfCard>
```

Vertical

By default, action buttons positioned in horizontal alignment, and also it can be aligned to show in vertical alignment by adding `Orientation` property.

ASPX-CS

```
@using Syncfusion.Blazor.Cards
@using Syncfusion.Blazor.Buttons
<SfCard ID="HugeImage" Orientation="CardOrientation.Vertical">
<CardImage Image="images/cards/steven.png"/>
<CardHeader Title="Harrisburg Keith" SubTitle="@CardSubTitle"/>
<CardContent Content="Hi, I'm creative graphic design for print, new media
based in Edenbridge"/>
<CardFooter>
<CardFooterContent>
<SfButton CssClass="e-btn e-outline e-primary">FOLLOW US</SfButton>
</CardFooterContent>
</CardFooter>
</SfCard>
```

Horizontal Card in Blazor Card Component

By default, all the card elements are aligned vertically one after the other as in the DOM. You can achieve the element to align horizontally as well by using `Orientation` property.

Stacked cards

A horizontally aligned card can push a specific column to align vertically using the `CardStacked` component. This will align the stacked section vertically to differentiate from horizontal layout.

ASPX-CS

```
@using Syncfusion.Blazor.Cards
```

```
<SfCard Orientation="CardOrientation.Horizontal" ID="Trimmer">
  <CardStacked>
    <CardHeader Title="Philips Trimmer" />
    <CardContent Content="Philips trimmers are designed to last longer than 4
ordinary trimmers and DuraPower Technology which optimizes power." />
  </CardStacked>
  
</SfCard>
<style>
.e-card-image {
background: url('../sample.jpg');
height: 160px;;
}
.e-card {
width: 300px;
margin: auto;
}
</style>
```



Philips Trimmer

Powered by the innovative DuraPower Technology which optimizes power consumption, Philips trimmers are designed to last longer than 4 ordinary trimmers.

Style and Appearance in Blazor Card Component

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

Customizing the card

Use the following CSS to customize the card properties.

CSS

```
.e-card {
background-color: aqua;
padding-left: 20px;
margin-bottom: 20px;
}
```

Customizing the Header element

Use the following CSS to customize the Header element properties.

CSS

```
.e-card .e-card-header {  
  font-family: cursive;  
  font-style: italic;  
}
```

Customizing the card content

Use the following CSS to customize the card content properties.

CSS

```
.e-card .e-card-content {  
  font-size: 20px;  
  color: gray;  
  line-height: initial;  
  font-weight: normal;  
}
```

Divider used to separate the elements inside the card

Use the following CSS to customize the Divider used to separate the elements inside the card properties.

CSS

```
.e-card .e-card-separator {  
  padding-bottom: 30px;  
}
```

Including image within card element

Use the following CSS to Include image within card element.

CSS

```
.e-card .e-card-image {  
  background-image: url(images.png);  
  background-color: yellow;  
  height: 160px;  
}
```

Including a title or caption for the image

Use the following CSS to Include a title or caption for the image.

CSS

```
.e-card .e-card-image .e-card-title {  
  font-family: cursive;  
  font-style: italic;  
}
```

To include heading image within the header

Use the following CSS to Include heading image within the header.

CSS

```
.e-card .e-card-header .e-card-header-image {  
height: 48px;  
width: 48px;  
}
```

[Customizing the Header main title](#)

Use the following CSS to Customize the Header main title.

CSS

```
.e-card .e-card-header .e-card-header-caption .e-card-header-title {  
font-size: large;  
color: aquamarine;  
}
```

[Customizing the Header subtitle](#)

Use the following CSS to Customize the Header subtitle.

CSS

```
.e-card .e-card-header .e-card-header-caption .e-card-sub-title {  
font-size: 20px;  
font-variant: all-petite-caps;  
}
```

[Including action buttons or anchor tags](#)

Use the following CSS to Include action buttons or anchor tags.

CSS

```
.e-card .e-card-actions .e-card-btn {  
padding-left: 20px;  
background-color: wheat;  
}
```

[To align card elements horizontally](#)

Use the following CSS to align card elements horizontally.

CSS

```
.e-card .e-card-horizontal {  
margin: auto;  
width: inherit;  
}
```

[To align elements vertically within the horizontal layout](#)

Use the following CSS to align elements vertically within the horizontal layout.

CSS

```
.e-card .e-card-horizontal .e-card-stacked {  
justify-content: flex-start;
```

```
margin: initial;
}
```

Charts

<!-- markdownlint-disable MD040 -->

Blazor Charts Component in Server Side App using Visual Studio

This section briefly explains about how to include a **Chart** in your Blazor Server-Side application. You can refer [Getting Started with Syncfusion Blazor for Server-Side in Visual Studio](#) page for the introduction and configuring the common specifications.

Importing Syncfusion Blazor component in the application

1. Install the **Syncfusion.Blazor** NuGet package to the application by using the **NuGet Package Manager**.
2. You can add the client-side resources through [CDN](#) or from [NuGet](#) package in the **HEAD** element of the `~/Pages/_Host.cshtml` page. For Internet Explorer 11, kindly refer the polyfills. Refer the [documentation](#) for more information.

ASPX-CS

```
<head>
<environment include="Development">
  ....
  ....
<script
src="https://github.com/Daddoon/Blazor.Polyfill/releases/download/3.0.1/blaz
or.polyfill.min.js"></script>
</environment>
</head>
```

3. Now add the lodash script to the **HEAD** element of the `/Pages/Host.cshtml` page, since we have used it in our [chart interactive](#) features. The absence of the script will result in console errors.

ASPX-CS

```
<head>
<environment include="Development">
  ....
  ....
<script
src="https://github.com/Daddoon/Blazor.Polyfill/releases/download/3.0.1/blaz
or.polyfill.min.js"></script>
<script
src="https://cdnjs.cloudflare.com/ajax/libs/lodash.js/4.17.20/lodash.min.js"
></script>
</environment>
</head>
```

Adding component package to the application

Open `~/_Imports.razor` file and include the **Syncfusion.Blazor** namespaces.

ASPX-CS

```
@using Syncfusion.Blazor
@using Syncfusion.Blazor.Charts
```

Add SyncfusionBlazor service in Startup.cs

Open the **Startup.cs** file and add services required by Syncfusion components using **service.AddSyncfusionBlazor()** method. Add this method in the **ConfigureServices** function as follows.

CSHARP

```
using Syncfusion.Blazor;
namespace BlazorApplication
{
    public class Startup
    {
        ....
        ....
        public void ConfigureServices(IServiceCollection services)
        {
            ....
            ....
            services.AddSyncfusionBlazor();
        }
    }
}
```

During initial loading, we collect and send individual character size information in-order to render the chart. To avoid any disconnection, increase the buffer size to 64 KB or more over the SignalR connection.

CSHARP

```
using Syncfusion.Blazor;
namespace BlazorApplication
{
    public class Startup
    {
        ....
        ....
        public void ConfigureServices(IServiceCollection services)
        {
            ....
            ....
            services.AddSyncfusionBlazor();
            services.AddSignalR(e => {
                e.MaximumReceiveMessageSize = 65536;
            });
        }
    }
}
```

Use the following configuration to host your Blazor server application on **Azure SignalR**.

CSHARP

```
using Syncfusion.Blazor;  
namespace BlazorApplication  
{  
    public class Startup  
    {  
        ....  
        ....  
        public void ConfigureServices(IServiceCollection services)  
        {  
            ....  
            ....  
            services.AddSyncfusionBlazor();  
            services.AddSignalR(e => {e.MaximumReceiveMessageSize =  
                65536;}).AddAzureSignalR();  
        }  
    }  
}
```

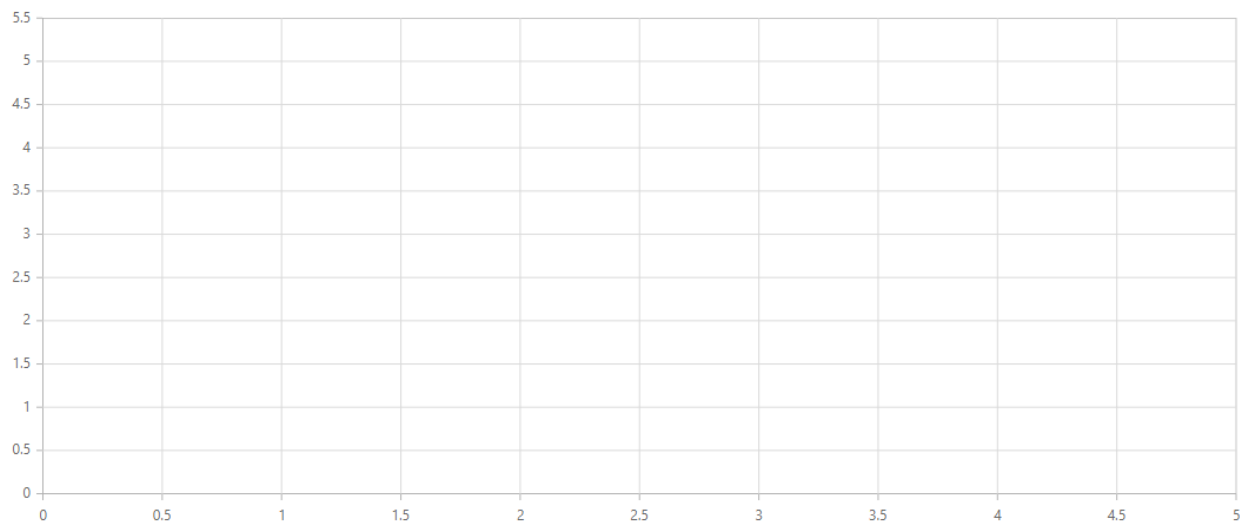
Add Chart Component

To initialize the chart component add the below code to your **Index.razor** view page under **~/Pages** folder. In a new application, if **Index.razor** page has any default content template, then those content can be completely removed and following code can be added.

ASPX-CS

```
@page "/"  
<SfChart>  
</SfChart>
```

On successful compilation of your application, the Syncfusion Blazor Chart component will render in the web browser.



Populate Chart with Data

To bind data for the chart component, you can assign an [IEnumerable](#) object to the [DataSource](#) property. It can also be provided as an instance of the [DataManager](#).

CSHARP

```
public class SalesInfo
{
    public string Month { get; set; }
    public double SalesValue { get; set; }
}

public List<SalesInfo> Sales = new List<SalesInfo>
{
    new SalesInfo { Month = "Jan", SalesValue = 35 },
    new SalesInfo { Month = "Feb", SalesValue = 28 },
    new SalesInfo { Month = "Mar", SalesValue = 34 },
    new SalesInfo { Month = "Apr", SalesValue = 32 },
    new SalesInfo { Month = "May", SalesValue = 40 },
    new SalesInfo { Month = "Jun", SalesValue = 32 },
    new SalesInfo { Month = "Jul", SalesValue = 35 }
};
```

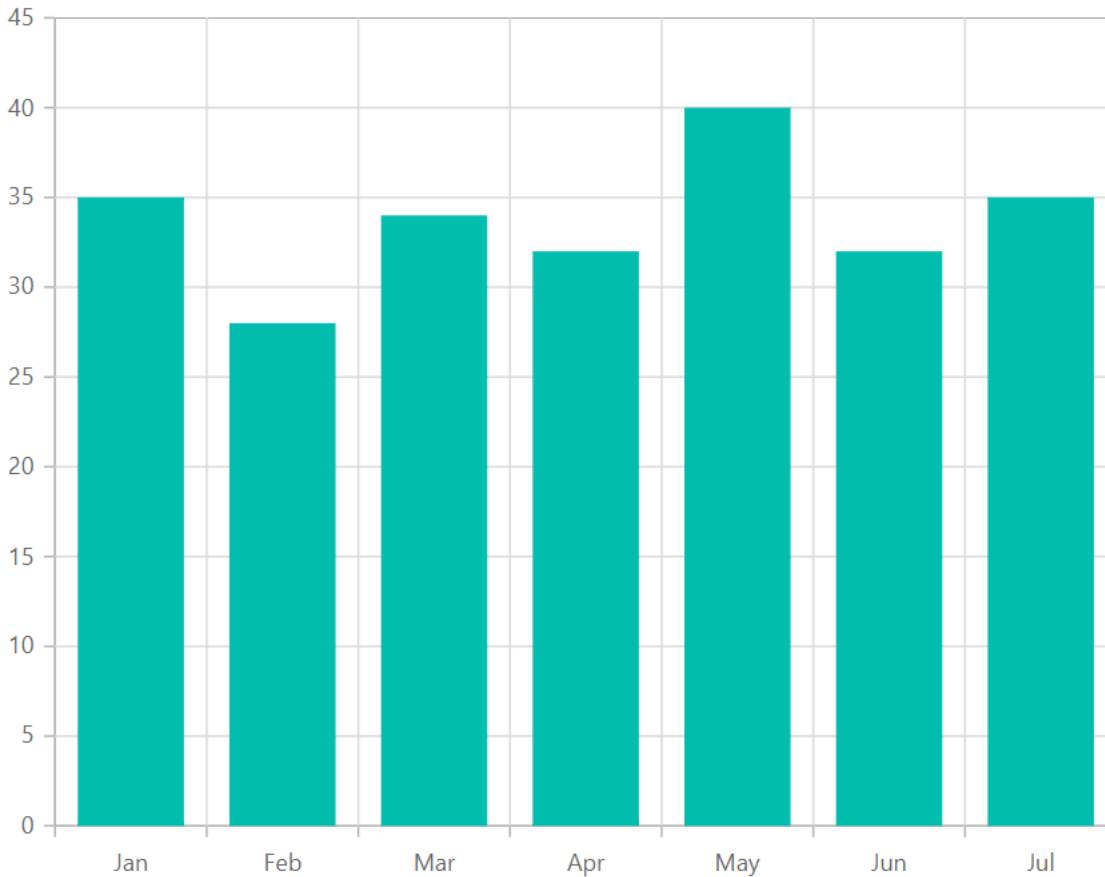
Now, map the data fields `Month` and `Sales` to the series [XName](#) and [YName](#) properties, then set the data to the [DataSource](#) property, and the [chart type](#) to `Column` because we will be viewing the data in a column chart.

ASPX-CS

```
@page "/"
@using Syncfusion.Blazor.Charts
<SfChart>
    <ChartPrimaryXAxis
        ValueType="Syncfusion.Blazor.Charts.ValueType.Category"></ChartPrimaryXAxis>
    <ChartSeriesCollection>
        <ChartSeries DataSource="@Sales" XName="Month" YName="SalesValue"
            Type="ChartSeriesType.Column">
        </ChartSeries>
    </ChartSeriesCollection>
</SfChart>

@code {
    public class SalesInfo
    {
        public string Month { get; set; }
        public double SalesValue { get; set; }
    }

    public List<SalesInfo> Sales = new List<SalesInfo>
    {
        new SalesInfo { Month = "Jan", SalesValue = 35 },
        new SalesInfo { Month = "Feb", SalesValue = 28 },
        new SalesInfo { Month = "Mar", SalesValue = 34 },
        new SalesInfo { Month = "Apr", SalesValue = 32 },
        new SalesInfo { Month = "May", SalesValue = 40 },
        new SalesInfo { Month = "Jun", SalesValue = 32 },
        new SalesInfo { Month = "Jul", SalesValue = 35 }
    };
}
```



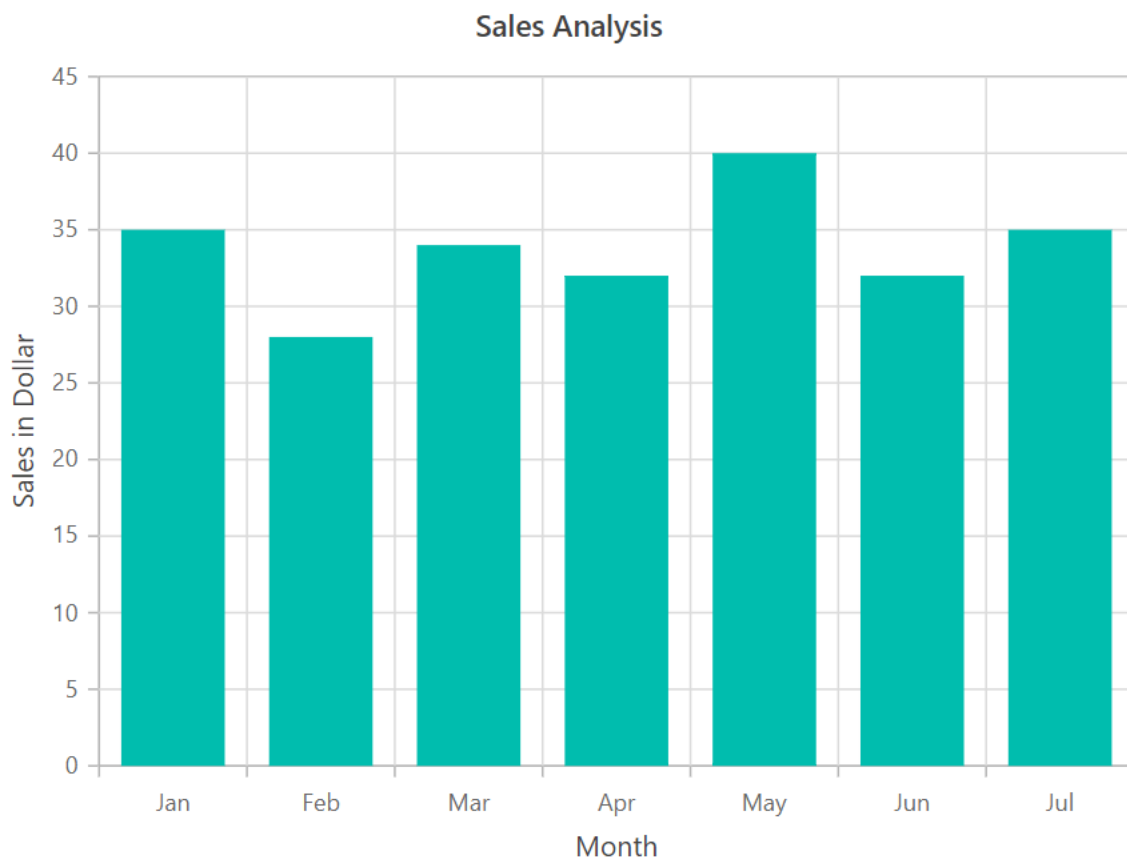
Add Titles

Using the [Title](#) property, you can add a title to the chart and the axes to provide the user with quick information about the data plotted in the chart.

ASPX-CS

```
@page "/"
@using Syncfusion.Blazor.Charts
<SfChart Title="Sales Analysis">
  <ChartPrimaryXAxis Title="Month"
  ValueType="Syncfusion.Blazor.Charts.ValueType.Category"></ChartPrimaryXAxis>
  <ChartPrimaryYAxis Title="Sales in Dollar"></ChartPrimaryYAxis>
  <ChartSeriesCollection>
    <ChartSeries DataSource="@Sales" XName="Month" YName="SalesValue"
    Type="ChartSeriesType.Column">
  </ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code {
  public class SalesInfo
  {
    public string Month { get; set; }
    public double SalesValue { get; set; }
  }
}
```

```
public List<SalesInfo> Sales = new List<SalesInfo>
{
    new SalesInfo { Month = "Jan", SalesValue = 35 },
    new SalesInfo { Month = "Feb", SalesValue = 28 },
    new SalesInfo { Month = "Mar", SalesValue = 34 },
    new SalesInfo { Month = "Apr", SalesValue = 32 },
    new SalesInfo { Month = "May", SalesValue = 40 },
    new SalesInfo { Month = "Jun", SalesValue = 32 },
    new SalesInfo { Month = "Jul", SalesValue = 35 }
};
```



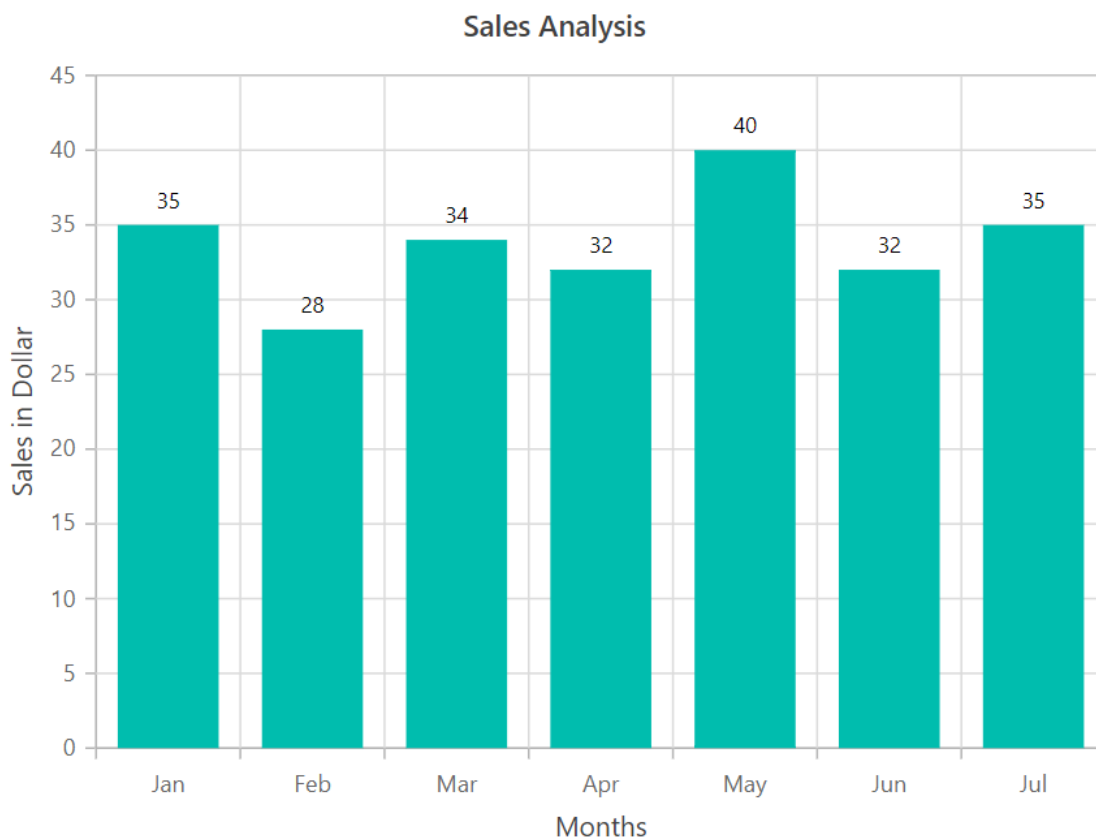
Add Data Label

You can add data labels to improve the readability of the chart. This can be achieved by setting the [Visible](#) property to **true** in the [ChartDataLabel](#).

ASPX-CS

```
@page "/"
@using Syncfusion.Blazor.Charts
<SfChart Title="Sales Analysis">
    <ChartPrimaryXAxis Title="Month"
        ValueType="Syncfusion.Blazor.Charts.ValueType.Category"></ChartPrimaryXAxis>
    <ChartPrimaryYAxis Title="Sales in Dollar"></ChartPrimaryYAxis>
    <ChartSeriesCollection>
```

```
<ChartSeries DataSource="@Sales" XName="Month" YName="SalesValue"
Type="ChartSeriesType.Column">
<ChartMarker>
<ChartDataLabel Visible="true"></ChartDataLabel>
</ChartMarker>
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code {
public class SalesInfo
{
public string Month { get; set; }
public double SalesValue { get; set; }
}
public List<SalesInfo> Sales = new List<SalesInfo>
{
new SalesInfo { Month = "Jan", SalesValue = 35 },
new SalesInfo { Month = "Feb", SalesValue = 28 },
new SalesInfo { Month = "Mar", SalesValue = 34 },
new SalesInfo { Month = "Apr", SalesValue = 32 },
new SalesInfo { Month = "May", SalesValue = 40 },
new SalesInfo { Month = "Jun", SalesValue = 32 },
new SalesInfo { Month = "Jul", SalesValue = 35 }
};
}
```

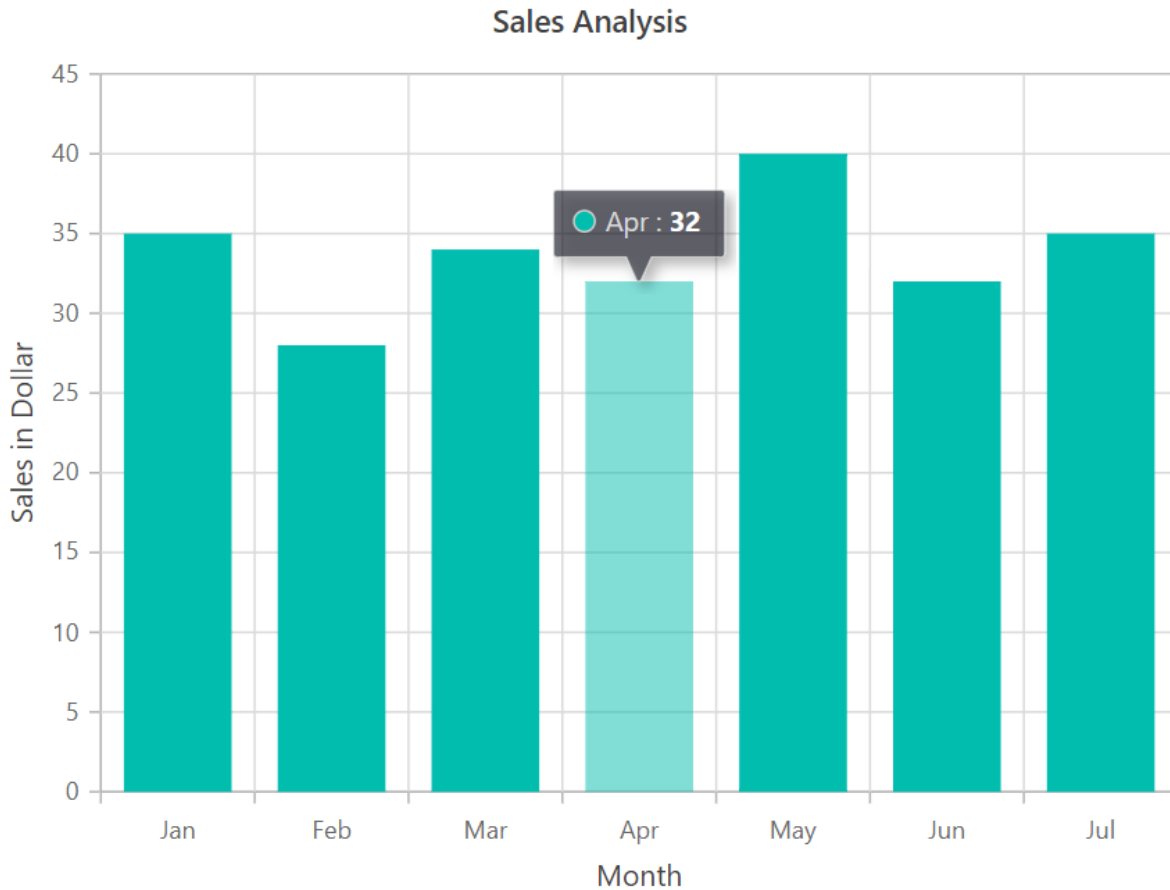


Enable Tooltip

When space constraints prevent you from displaying information using data labels, the tooltip comes in handy. The tooltip can be enabled by setting the [Enable](#) property in [ChartTooltipSettings](#) to **true**.

ASPX-CS

```
@page "/"
@using Syncfusion.Blazor.Charts
<SfChart Title="Sales Analysis">
  <ChartPrimaryXAxis Title="Month"
  ValueType="Syncfusion.Blazor.Charts.ValueType.Category"></ChartPrimaryXAxis>
  <ChartPrimaryYAxis Title="Sales in Dollar"></ChartPrimaryYAxis>
  <ChartTooltipSettings Enable="true"></ChartTooltipSettings>
  <ChartSeriesCollection>
    <ChartSeries DataSource="@Sales" XName="Month" YName="SalesValue"
    Type="ChartSeriesType.Column">
  </ChartSeries>
  </ChartSeriesCollection>
</SfChart>
@code {
public class SalesInfo
{
public string Month { get; set; }
public double SalesValue { get; set; }
}
public List<SalesInfo> Sales = new List<SalesInfo>
{
new SalesInfo { Month = "Jan", SalesValue = 35 },
new SalesInfo { Month = "Feb", SalesValue = 28 },
new SalesInfo { Month = "Mar", SalesValue = 34 },
new SalesInfo { Month = "Apr", SalesValue = 32 },
new SalesInfo { Month = "May", SalesValue = 40 },
new SalesInfo { Month = "Jun", SalesValue = 32 },
new SalesInfo { Month = "Jul", SalesValue = 35 }
};
}
```



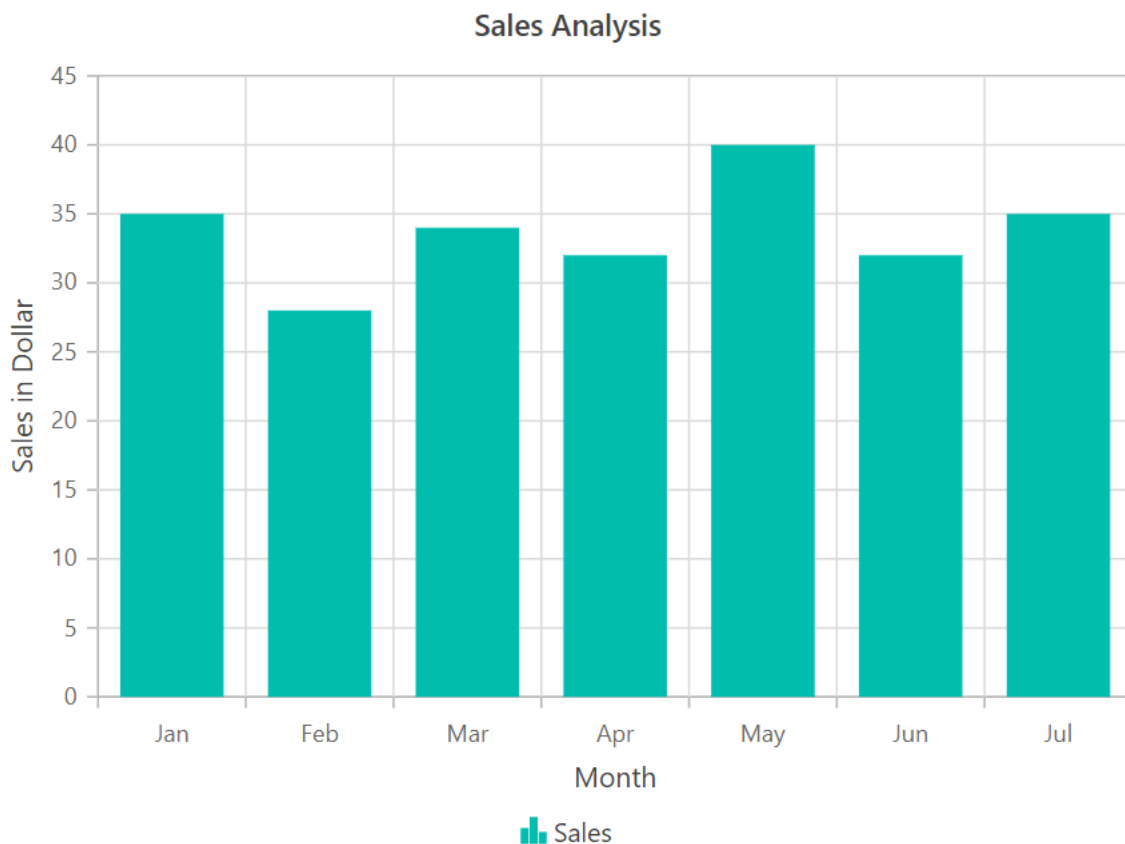
Enable Legend

You can use legend for the chart by setting the [Visible](#) property to **true** in [ChartLegendSettings](#). The legend name can be changed by using the [Name](#) property in the series.

ASPX-CS

```
@page "/"
@using Syncfusion.Blazor.Charts
<SfChart Title="Sales Analysis">
  <ChartPrimaryXAxis Title="Month"
    ValueType="Syncfusion.Blazor.Charts.ValueType.Category"></ChartPrimaryXAxis>
  <ChartPrimaryYAxis Title="Sales in Dollar"></ChartPrimaryYAxis>
  <ChartLegendSettings Visible="true"></ChartLegendSettings>
  <ChartSeriesCollection>
    <ChartSeries DataSource="@Sales" Name="Sales" XName="Month"
      YName="SalesValue" Type="ChartSeriesType.Column">
    </ChartSeries>
  </ChartSeriesCollection>
</SfChart>
@code {
  public class SalesInfo
  {
    public string Month { get; set; }
    public double SalesValue { get; set; }
  }
  public List<SalesInfo> Sales = new List<SalesInfo>
```

```
{  
  new SalesInfo { Month = "Jan", SalesValue = 35 },  
  new SalesInfo { Month = "Feb", SalesValue = 28 },  
  new SalesInfo { Month = "Mar", SalesValue = 34 },  
  new SalesInfo { Month = "Apr", SalesValue = 32 },  
  new SalesInfo { Month = "May", SalesValue = 40 },  
  new SalesInfo { Month = "Jun", SalesValue = 32 },  
  new SalesInfo { Month = "Jul", SalesValue = 35 }  
};  
}
```



You can find the fully working sample [here](#). And also you can refer to our [Blazor Chart](#) feature tour page for its groundbreaking feature representations. You can also explore our [Blazor Chart Example](#) to know various chart types and how to represent time-dependent data, showing trends in data at equal intervals.

See also

- [Getting Started with Syncfusion Blazor for Client-Side in .NET Core CLI](#)
- [Getting Started with Syncfusion Blazor for Server-Side in Visual Studio](#)
- [Getting Started with Syncfusion Blazor for Server-Side in .NET Core CLI](#)

<!-- markdownlint-disable MD040 -->

Blazor Charts Component in WebAssembly App using Visual Studio

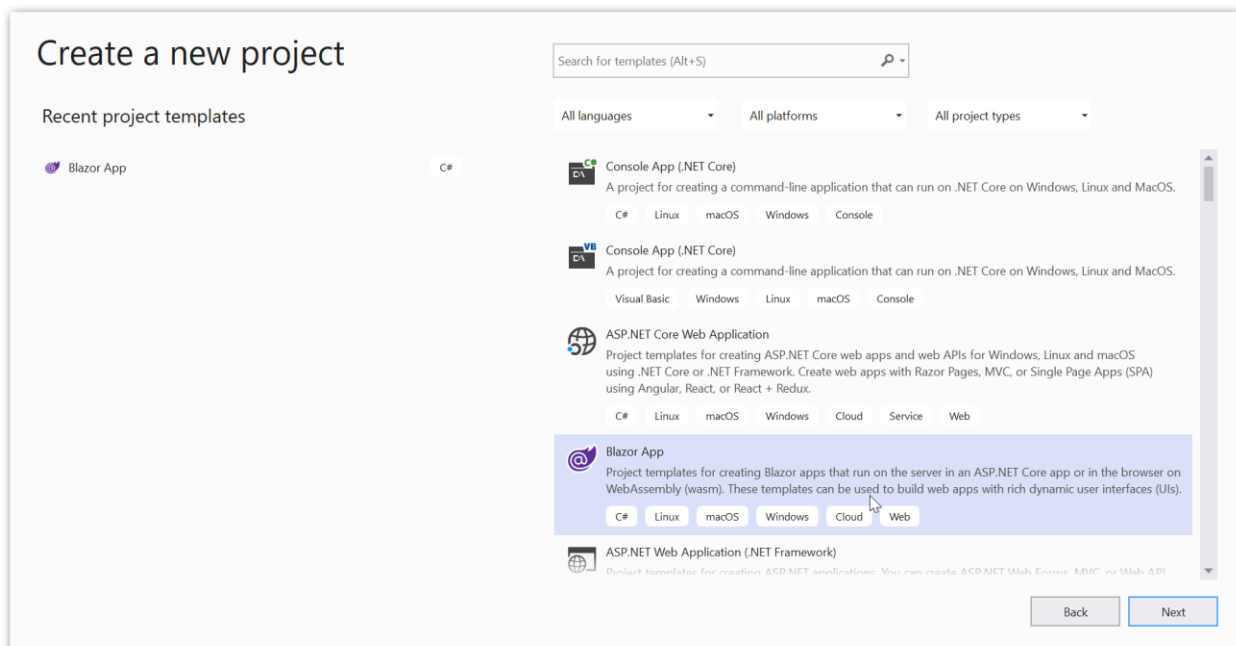
This article provides a step-by-step instructions to configure Syncfusion Blazor Chart in a simple Blazor WebAssembly application using Visual Studio 2019.

Create a Blazor WebAssembly project in Visual Studio 2019

1. Choose **Create a new project** from the Visual Studio dashboard.



2. Select **Blazor App** from the template and click **Next** button.



- Now, the project configuration window will popup. Click **Create** button to create a new project with the default project configuration.

Configure your new project

Blazor App C# Linux macOS Windows Cloud Web

Project name

BlazorApp1

Location

C:\Users\ThangamaniBalasubram\source\repos

Solution name ⓘ

BlazorApp1

☒ Place solution and project in the same directory

- Depending on the requirement, select **.NET Core 3.1 or higher** as the target framework.

Create a new Blazor app

.NET Core 3.1

.NET Core 3.1

.NET 5.0

A project template for creating a Blazor server app that runs server-side inside an ASP.NET Core app and handles user interactions over a SignalR connection. This template can be used for web apps with rich dynamic user interfaces (UIs).

Blazor WebAssembly App

A project template for creating a Blazor app that runs on WebAssembly. This template can be used for web apps with rich dynamic user interfaces (UIs).

Authentication

No Authentication

[Change](#)

Advanced

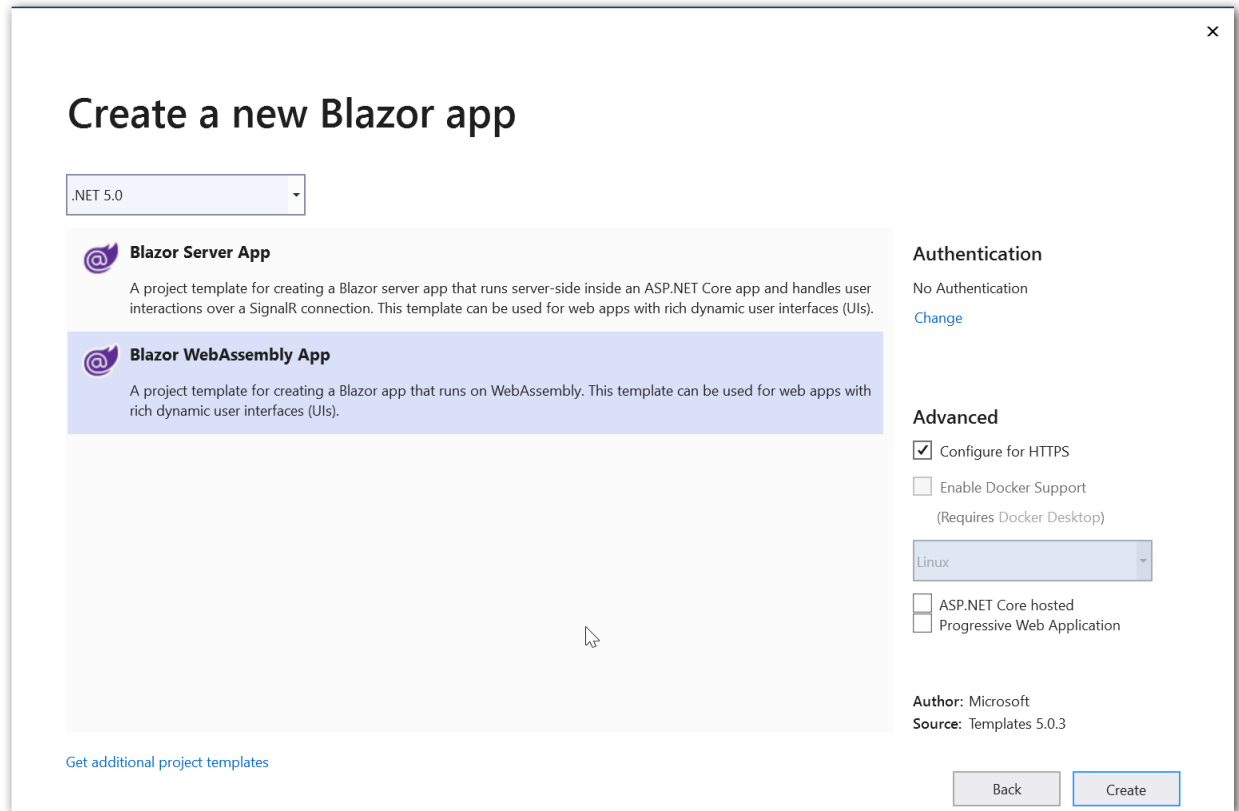
☒ Configure for HTTPS

☐ Enable Docker Support

(Requires [Docker Desktop](#))

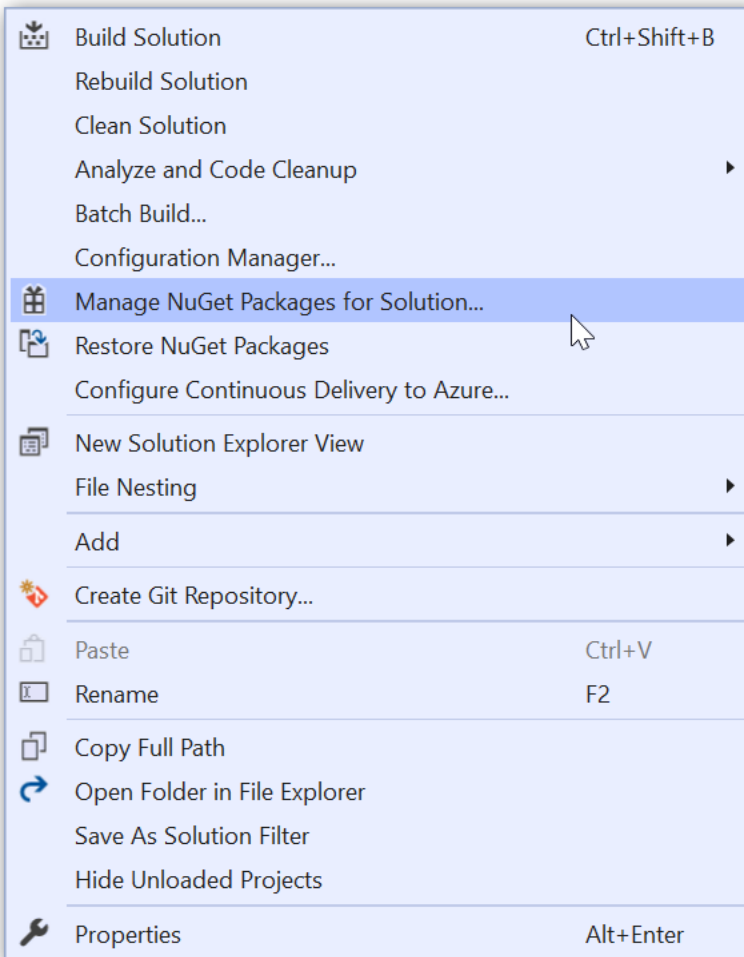
Linux

- Choose **Blazor WebAssembly App** from the dashboard and click **Create** button to create a new Blazor WebAssembly application.

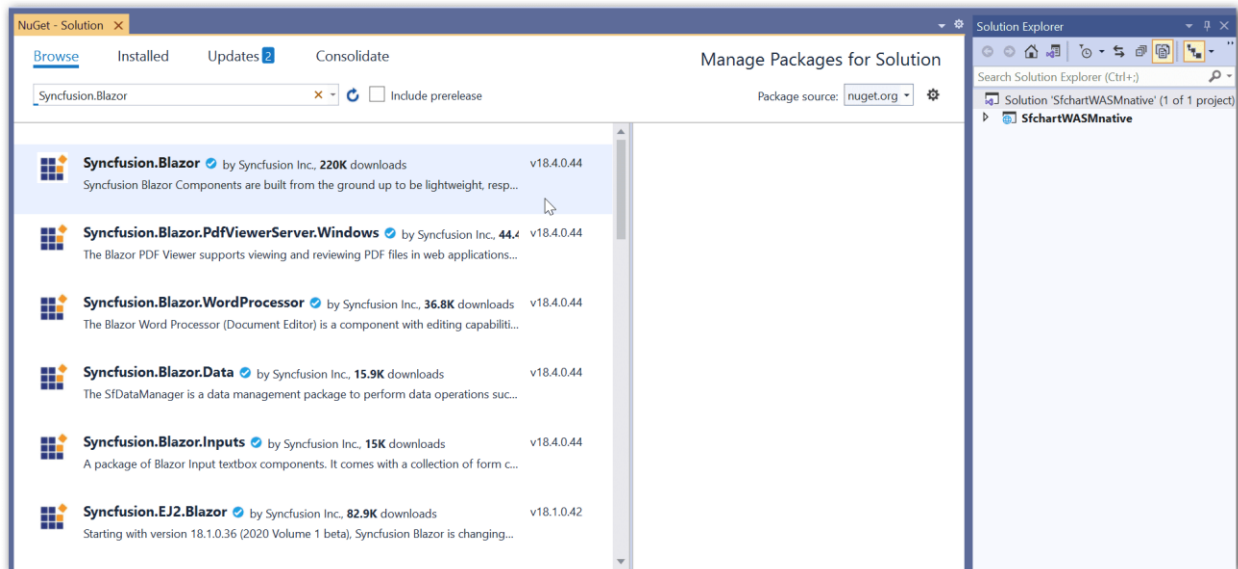


Importing Syncfusion Blazor component in the application

1. Now, install **Syncfusion.Blazor** NuGet package to the newly created application by using the **NuGet Package Manager**. Right-click the project and select **Manage NuGet Packages**.



2. Search **Syncfusion.Blazor** keyword in the **Browser** tab and install Syncfusion.Blazor NuGet package in the application.



3. You can add the client-side resources through [CDN](#) or from [NuGet](#) package in the **HEAD** element of the `~/wwwroot/index.html` page. For Internet Explorer 11, kindly refer the polyfills.

HTML

```
<head>
...
...
<script
src="https://github.com/Daddoon/Blazor.Polyfill/releases/download/3.0.1/blaz
or.polyfill.min.js"></script>
</head>
```

4. Now add the lodash script to the **HEAD** element of the `~/wwwroot/index.html` page, since we have used it in our [chart interactive](#) features. The absence of the script will result in console errors.

HTML

```
<head>
...
...
<script
src="https://github.com/Daddoon/Blazor.Polyfill/releases/download/3.0.1/blaz
or.polyfill.min.js"></script>
<script
src="https://cdnjs.cloudflare.com/ajax/libs/lodash.js/4.17.20/lodash.min.js"
></script>
</head>
```

Adding component package to the application

Open `~/_Imports.razor` file and include the **Syncfusion.Blazor.** namespaces.

ASPX-CS

```
@using Syncfusion.Blazor  
@using Syncfusion.Blazor.Charts
```

Add SyncfusionBlazor service in Startup.cs

Open the ~/Program.cs file and register the Syncfusion Blazor Service.

CSHARP

```
using Syncfusion.Blazor;  
namespace WebApplication1  
{  
    public class Program  
    {  
        public static async Task Main(string[] args)  
        {  
            ....  
            ....  
            builder.Services.AddSyncfusionBlazor();  
            await builder.Build().RunAsync();  
        }  
    }  
}
```

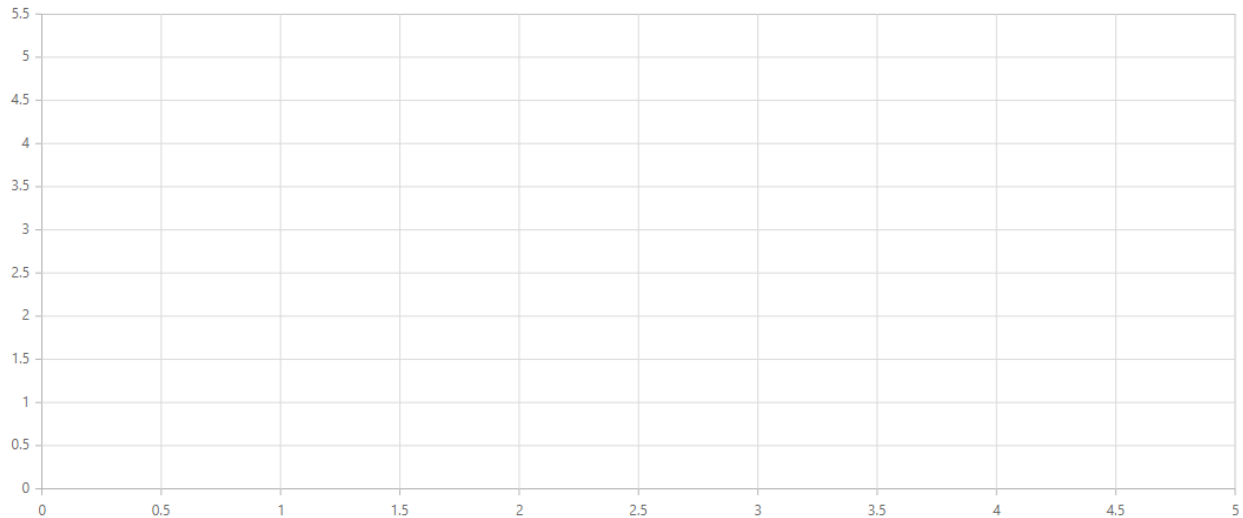
Add Chart Component

To initialize the chart component add the below code to your **Index.razor** view page under ~/Pages folder. In a new application, if **Index.razor** page has any default content template, then those content can be completely removed and following code can be added.

ASPX-CS

```
@page "/"  
<SfChart>  
</SfChart>
```

On successful compilation of your application, the Syncfusion Blazor Chart component will render in the web browser.



Populate Chart with Data

To bind data for the chart component, you can assign a `IEnumerable` object to the [DataSource](#) property. It can also be provided as an instance of the [DataManager](#).

CSHARP

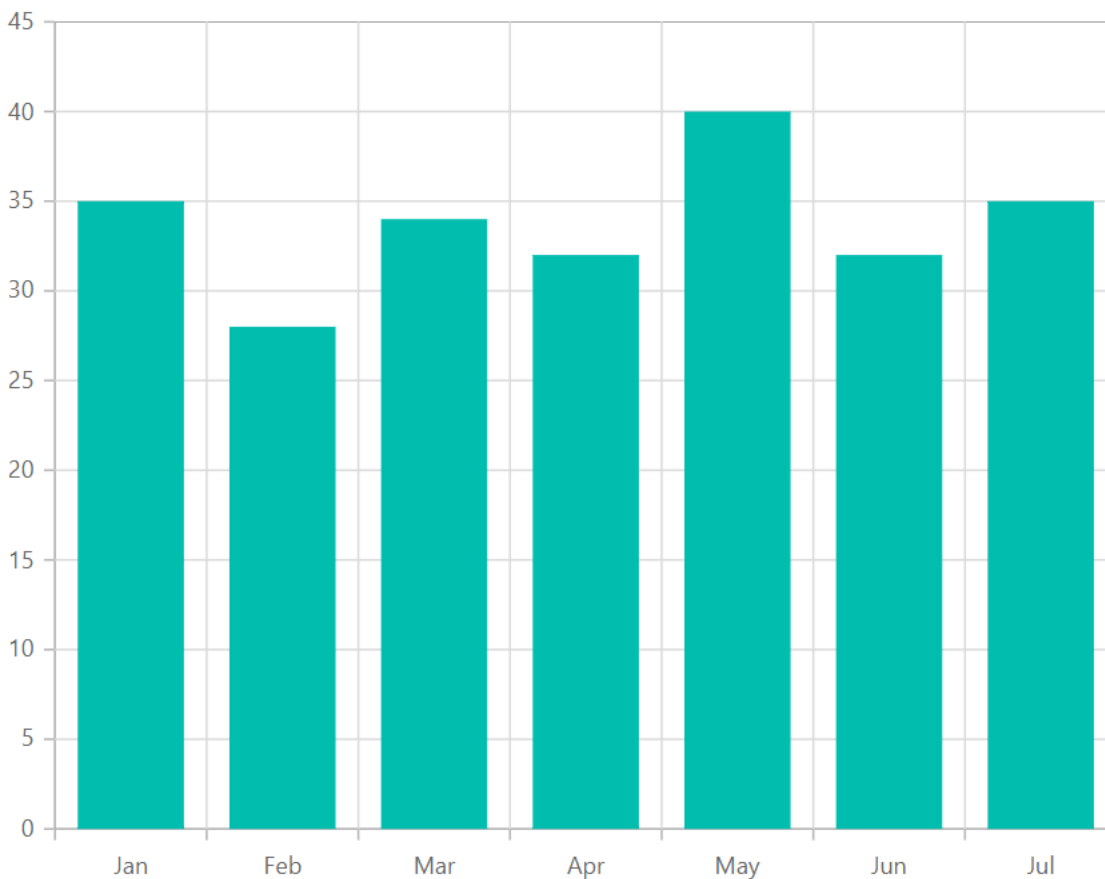
```
public class SalesInfo
{
    public string Month { get; set; }
    public double SalesValue { get; set; }
}
public List<SalesInfo> Sales = new List<SalesInfo>
{
    new SalesInfo { Month = "Jan", SalesValue = 35 },
    new SalesInfo { Month = "Feb", SalesValue = 28 },
    new SalesInfo { Month = "Mar", SalesValue = 34 },
    new SalesInfo { Month = "Apr", SalesValue = 32 },
    new SalesInfo { Month = "May", SalesValue = 40 },
    new SalesInfo { Month = "Jun", SalesValue = 32 },
    new SalesInfo { Month = "Jul", SalesValue = 35 }
};
```

Now, map the data fields `Month` and `Sales` to the series [XName](#) and [YName](#) properties, then set the data to the [DataSource](#) property, and the [chart type](#) to `Column` because we will be viewing the data in a column chart.

ASPX-CS

```
@page "/"
@using Syncfusion.Blazor.Charts
<SfChart>
    <ChartPrimaryXAxis
        ValueType="Syncfusion.Blazor.Charts.ValueType.Category"></ChartPrimaryXAxis>
    <ChartSeriesCollection>
        <ChartSeries DataSource="@Sales" XName="Month" YName="SalesValue"
            Type="ChartSeriesType.Column">
        </ChartSeries>
    </ChartSeriesCollection>
```

```
</SfChart>
@code {
public class SalesInfo
{
public string Month { get; set;}
public double SalesValue { get; set;}
}
public List<SalesInfo> Sales = new List<SalesInfo>
{
new SalesInfo { Month = "Jan", SalesValue = 35 },
new SalesInfo { Month = "Feb", SalesValue = 28 },
new SalesInfo { Month = "Mar", SalesValue = 34 },
new SalesInfo { Month = "Apr", SalesValue = 32 },
new SalesInfo { Month = "May", SalesValue = 40 },
new SalesInfo { Month = "Jun", SalesValue = 32 },
new SalesInfo { Month = "Jul", SalesValue = 35 }
};
}
```

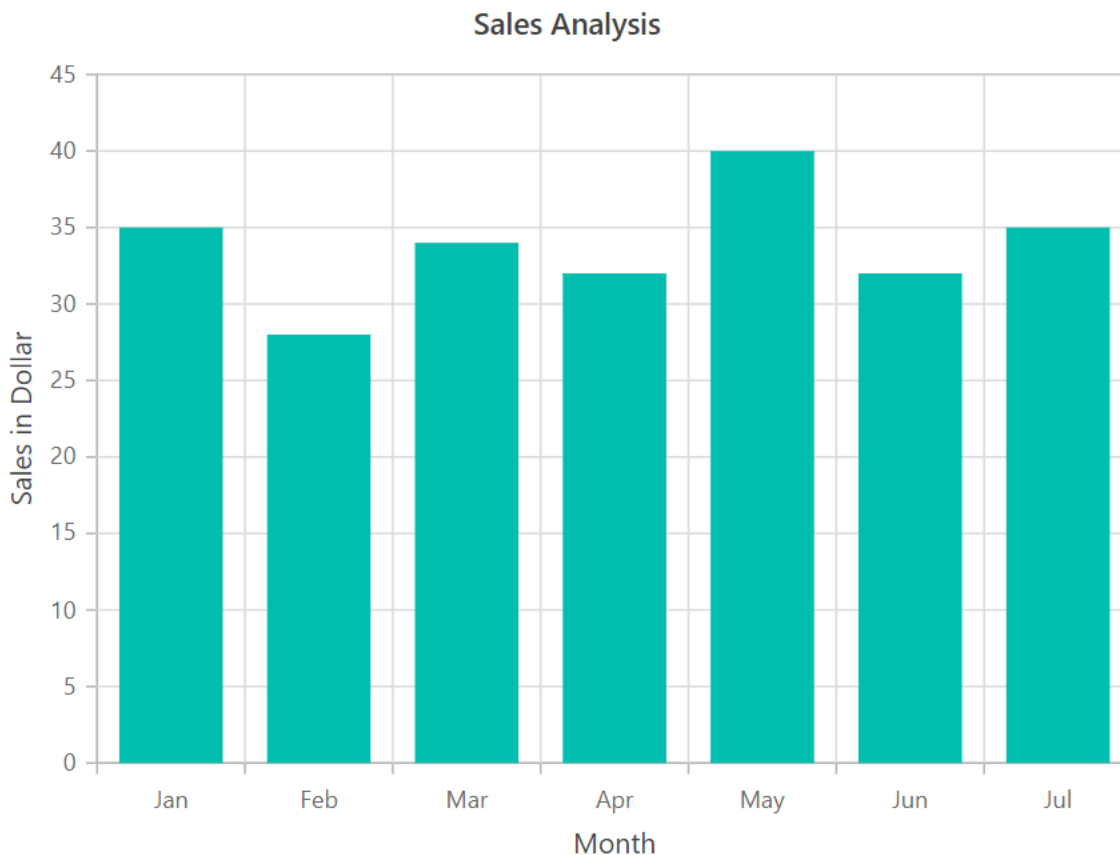


Add Titles

Using the [Title](#) property, you can add a title to the chart and the axes to provide the user with quick information about the data plotted in the chart.

ASPX-CS

```
@page "/"
@using Syncfusion.Blazor.Charts
<SfChart Title="Sales Analysis">
  <ChartPrimaryXAxis Title="Month"
  ValueType="Syncfusion.Blazor.Charts.ValueType.Category"></ChartPrimaryXAxis>
  <ChartPrimaryYAxis Title="Sales in Dollar"></ChartPrimaryYAxis>
  <ChartSeriesCollection>
    <ChartSeries DataSource="@Sales" XName="Month" YName="SalesValue"
    Type="ChartSeriesType.Column">
  </ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code {
public class SalesInfo
{
public string Month { get; set;}
public double SalesValue { get; set;}
}
public List<SalesInfo> Sales = new List<SalesInfo>
{
new SalesInfo { Month = "Jan", SalesValue = 35 },
new SalesInfo { Month = "Feb", SalesValue = 28 },
new SalesInfo { Month = "Mar", SalesValue = 34 },
new SalesInfo { Month = "Apr", SalesValue = 32 },
new SalesInfo { Month = "May", SalesValue = 40 },
new SalesInfo { Month = "Jun", SalesValue = 32 },
new SalesInfo { Month = "Jul", SalesValue = 35 }
};
}
```



Add Data Label

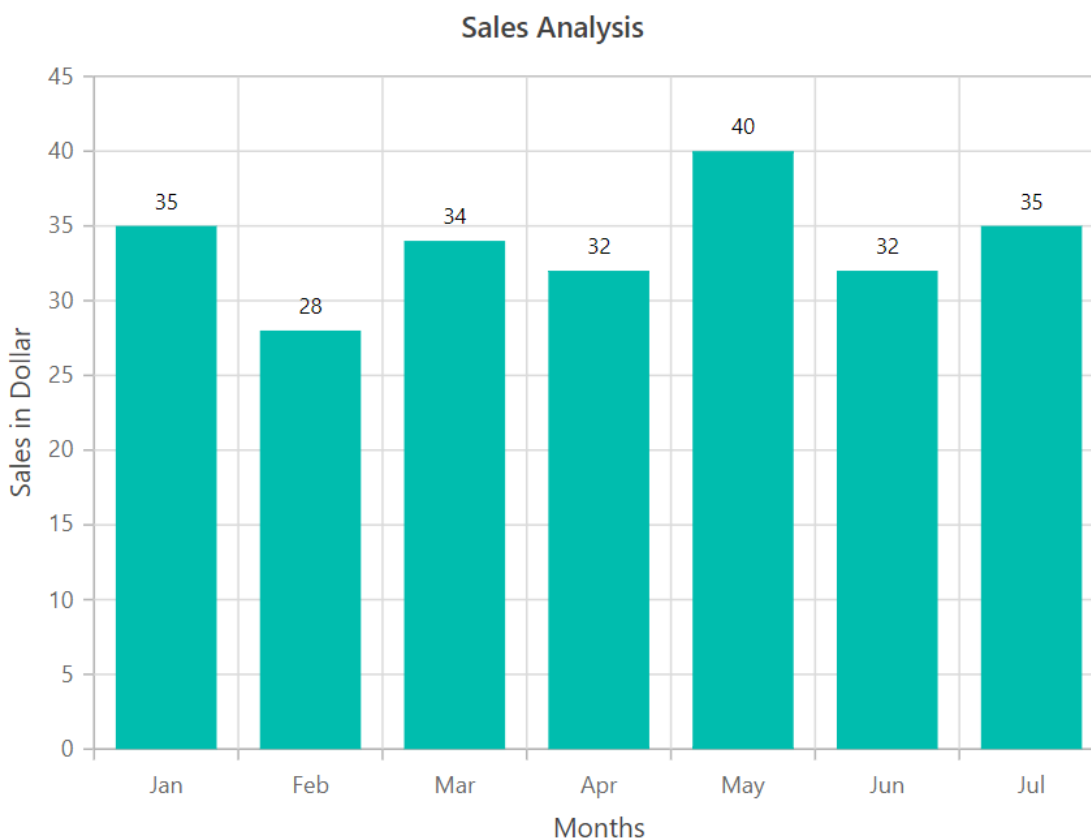
You can add data labels to improve the readability of the chart. This can be achieved by setting the [Visible](#) property to **true** in the [ChartDataLabel](#).

ASPX-CS

```
@page "/"
@using Syncfusion.Blazor.Charts
<SfChart Title="Sales Analysis">
  <ChartPrimaryXAxis Title="Month"
  ValueType="Syncfusion.Blazor.Charts.ValueType.Category"></ChartPrimaryXAxis>
  <ChartPrimaryYAxis Title="Sales in Dollar"></ChartPrimaryYAxis>
  <ChartSeriesCollection>
    <ChartSeries DataSource="@Sales" XName="Month" YName="SalesValue"
    Type="ChartSeriesType.Column">
      <ChartMarker>
        <ChartDataLabel Visible="true"></ChartDataLabel>
      </ChartMarker>
    </ChartSeries>
  </ChartSeriesCollection>
</SfChart>

@code {
public class SalesInfo
{
public string Month { get; set; }
public double SalesValue { get; set; }
}
```

```
public List<SalesInfo> Sales = new List<SalesInfo>
{
    new SalesInfo { Month = "Jan", SalesValue = 35 },
    new SalesInfo { Month = "Feb", SalesValue = 28 },
    new SalesInfo { Month = "Mar", SalesValue = 34 },
    new SalesInfo { Month = "Apr", SalesValue = 32 },
    new SalesInfo { Month = "May", SalesValue = 40 },
    new SalesInfo { Month = "Jun", SalesValue = 32 },
    new SalesInfo { Month = "Jul", SalesValue = 35 }
};
```



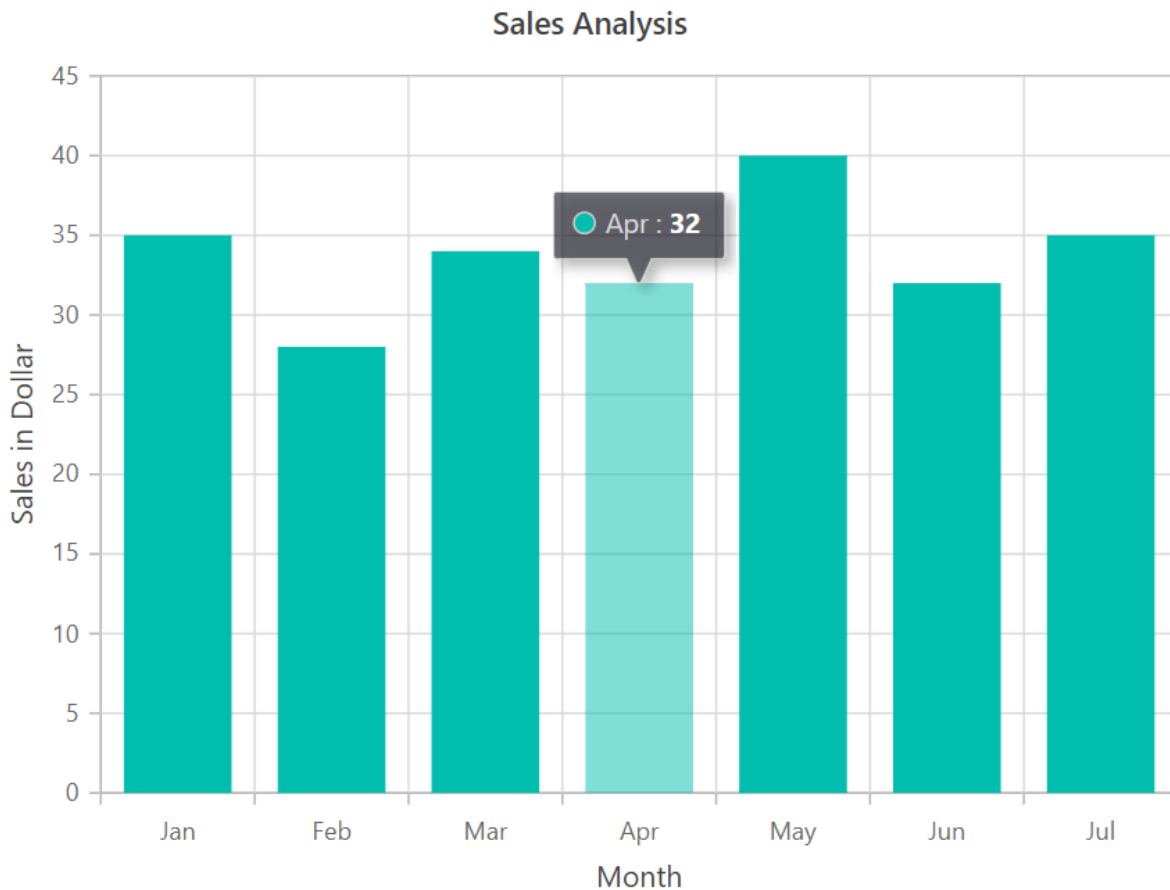
Enable Tooltip

When space constraints prevent you from displaying information using data labels, the tooltip comes in handy. The tooltip can be enabled by setting the [Enable](#) property in [ChartTooltipSettings](#) to **true**.

ASPX-CS

```
@page "/"
@using Syncfusion.Blazor.Charts
<SfChart Title="Sales Analysis">
    <ChartPrimaryXAxis Title="Month"
        ValueType="Syncfusion.Blazor.Charts.ValueType.Category"></ChartPrimaryXAxis>
    <ChartPrimaryYAxis Title="Sales in Dollar"></ChartPrimaryYAxis>
    <ChartTooltipSettings Enable="true"></ChartTooltipSettings>
    <ChartSeriesCollection>
```

```
<ChartSeries DataSource="@Sales" XName="Month" YName="SalesValue"  
Type="ChartSeriesType.Column">  
</ChartSeries>  
</ChartSeriesCollection>  
</SfChart>  
@code {  
public class SalesInfo  
{  
public string Month { get; set; }  
public double SalesValue { get; set; }  
}  
public List<SalesInfo> Sales = new List<SalesInfo>  
{  
new SalesInfo { Month = "Jan", SalesValue = 35 },  
new SalesInfo { Month = "Feb", SalesValue = 28 },  
new SalesInfo { Month = "Mar", SalesValue = 34 },  
new SalesInfo { Month = "Apr", SalesValue = 32 },  
new SalesInfo { Month = "May", SalesValue = 40 },  
new SalesInfo { Month = "Jun", SalesValue = 32 },  
new SalesInfo { Month = "Jul", SalesValue = 35 }  
};  
}
```

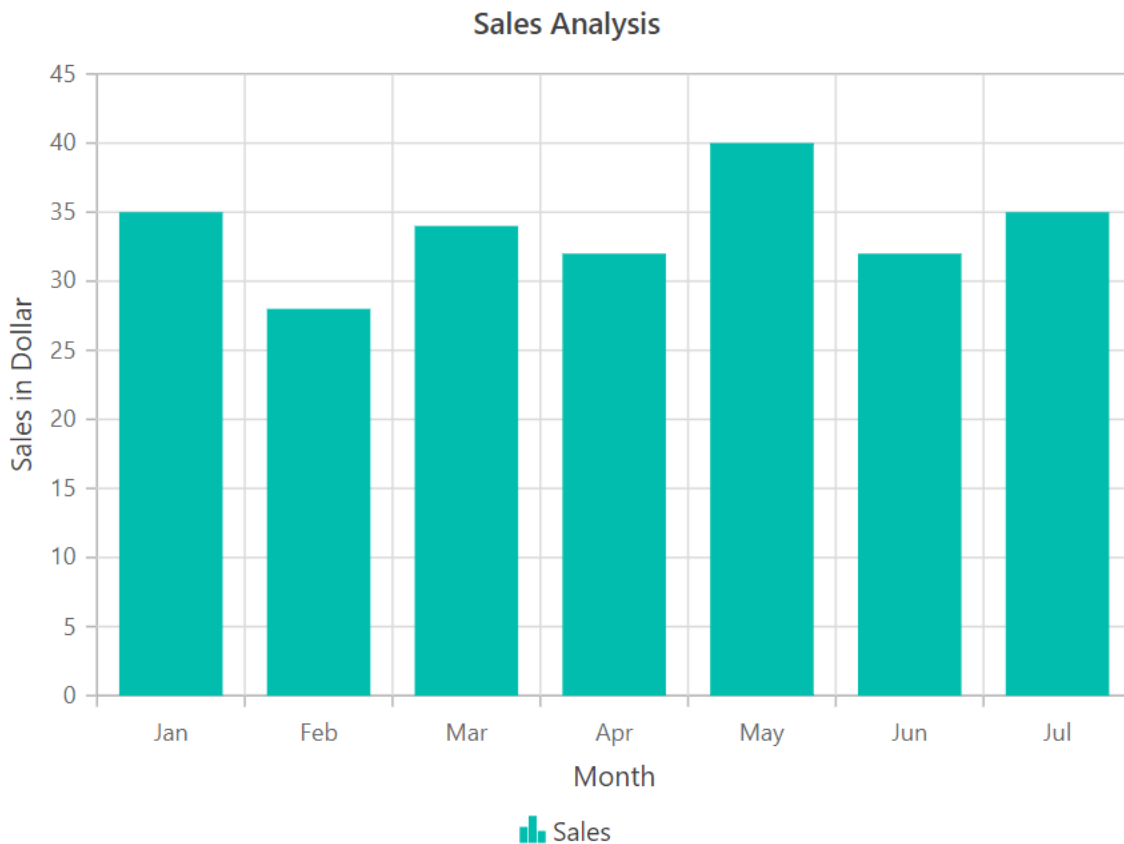


Enable Legend

You can use legend for the chart by setting the [Visible](#) property to **true** in [ChartLegendSettings](#). The legend name can be changed by using the [Name](#) property in the series.

ASPX-CS

```
@page "/"
@using Syncfusion.Blazor.Charts
<SfChart Title="Sales Analysis">
  <ChartPrimaryXAxis Title="Month"
  ValueType="Syncfusion.Blazor.Charts.ValueType.Category"></ChartPrimaryXAxis>
  <ChartPrimaryYAxis Title="Sales in Dollar"></ChartPrimaryYAxis>
  <ChartLegendSettings Visible="true"></ChartLegendSettings>
  <ChartSeriesCollection>
    <ChartSeries DataSource="@Sales" Name="Sales" XName="Month"
    YName="SalesValue" Type="ChartSeriesType.Column">
  </ChartSeries>
  </ChartSeriesCollection>
</SfChart>
@code {
public class SalesInfo
{
public string Month { get; set; }
public double SalesValue { get; set; }
}
public List<SalesInfo> Sales = new List<SalesInfo>
{
new SalesInfo { Month = "Jan", SalesValue = 35 },
new SalesInfo { Month = "Feb", SalesValue = 28 },
new SalesInfo { Month = "Mar", SalesValue = 34 },
new SalesInfo { Month = "Apr", SalesValue = 32 },
new SalesInfo { Month = "May", SalesValue = 40 },
new SalesInfo { Month = "Jun", SalesValue = 32 },
new SalesInfo { Month = "Jul", SalesValue = 35 }
};
}
```

You can find the fully working sample for chart [here](#). And also you can refer to our [Blazor Charts](#) feature tour page for its groundbreaking feature representations. You can also explore our [Blazor Chart Example](#) to know various chart types and how to represent time-dependent data, showing trends in data at equal intervals.

See also

- [Getting Started with Syncfusion Blazor for Client-Side in .NET Core CLI](#)
- [Getting Started with Syncfusion Blazor for Server-Side in Visual Studio](#)
- [Getting Started with Syncfusion Blazor for Server-Side in .NET Core CLI](#)

<!-- markdownlint-disable MD036 -->

Working with Data in Blazor Charts Component

The Chart uses [SfDataManager](#), which supports both RESTful JSON data services binding and IEnumerable binding. The [DataSource](#) value can be set using either [SfDataManager](#) property values or a list of business objects.

It supports the following data binding methods:

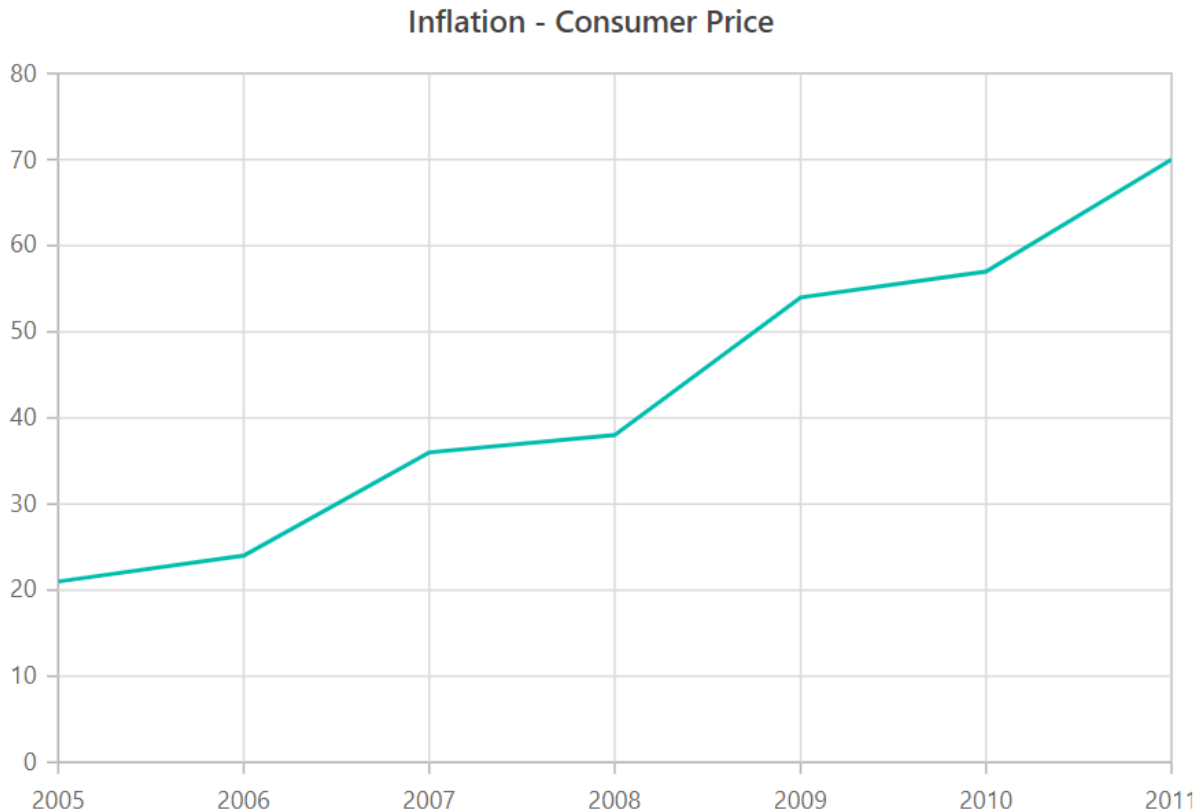
- List binding
- Remote data

List binding

An `IEnumerable` object can be assigned to the [DataSource](#) property. The list data source can alternatively be given as an instance of [SfDataManager](#) or by using [SfDataManager](#) or as a component of the [SfDataManager](#) or by using [SfDataManager](#). The data fields should now be mapped to the [XName](#) and [YName](#) properties.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart Title="Inflation - Consumer Price" Width="60%">
  <ChartPrimaryXAxis IntervalType="IntervalType.Years" LabelFormat="yyyy"
  ValueType="Syncfusion.Blazor.Charts.ValueType.DateTime">
  </ChartPrimaryXAxis>
  <ChartSeriesCollection>
  <ChartSeries DataSource="@ConsumerReports" XName="XValue" YName="YValue"
  Type="ChartSeriesType.Line">
  </ChartSeries>
  </ChartSeriesCollection>
</SfChart>
@code{
public class ChartData
{
public DateTime XValue { get; set; }
public double YValue { get; set; }
}
public List<ChartData> ConsumerReports = new List<ChartData>
{
new ChartData { XValue = new DateTime(2005, 01, 01), YValue = 21 },
new ChartData { XValue = new DateTime(2006, 01, 01), YValue = 24 },
new ChartData { XValue = new DateTime(2007, 01, 01), YValue = 36 },
new ChartData { XValue = new DateTime(2008, 01, 01), YValue = 38 },
new ChartData { XValue = new DateTime(2009, 01, 01), YValue = 54 },
new ChartData { XValue = new DateTime(2010, 01, 01), YValue = 57 },
new ChartData { XValue = new DateTime(2011, 01, 01), YValue = 70 },
};
}
```



By default, [SfDataManager](#) uses **BlazorAdaptor** for list data-binding.

ExpandoObject binding

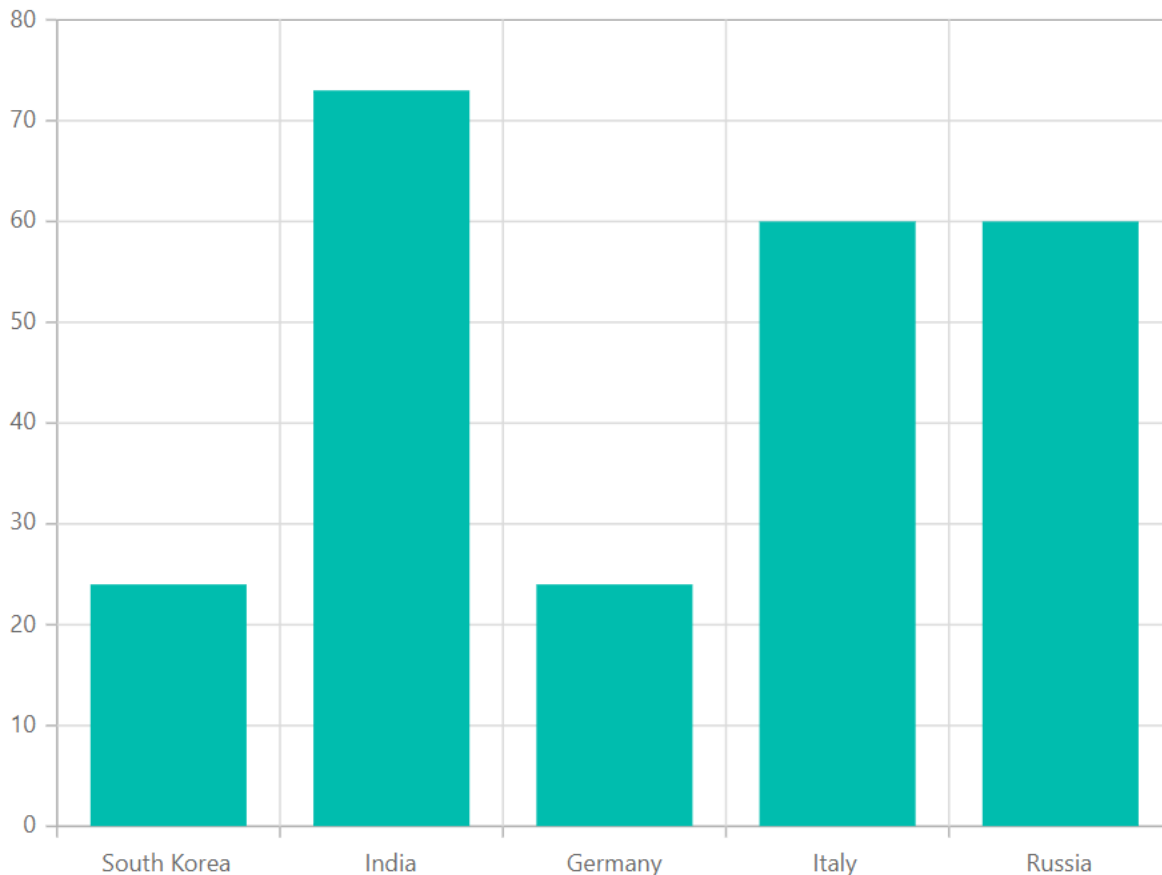
Chart is a generic component which is strongly bound to a model type. There are cases when the model type is unknown during compile time. In such circumstances data can be bound to the chart as a list of **ExpandoObjects**. The **ExpandoObject** can be bound to chart by assigning to the [DataSource](#) property.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
@using System.Dynamic

<SfChart>
<ChartPrimaryXAxis ValueType="Syncfusion.Blazor.Charts.ValueType.Category"
/>
<ChartSeriesCollection>
<ChartSeries DataSource="@MedalDetails" XName="X" YName="Y"
Type="ChartSeriesType.Column">
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code{
private List<string> countries = new List<string> { "South Korea", "India",
"Germany", "Italy", "Russia" };
private Random randomNum = new Random();
public List<ExpandoObject> MedalDetails { get; set; } = new
List<ExpandoObject>();
protected override void OnInitialized()
{
```

```
MedalDetails = Enumerable.Range(0, 5).Select((x) =>
{
    dynamic d = new ExpandoObject();
    d.X = countries[x];
    d.Y = randomNum.Next(20, 80);
    return d;
}).Cast<ExpandoObject>().ToList<ExpandoObject>();
}
```



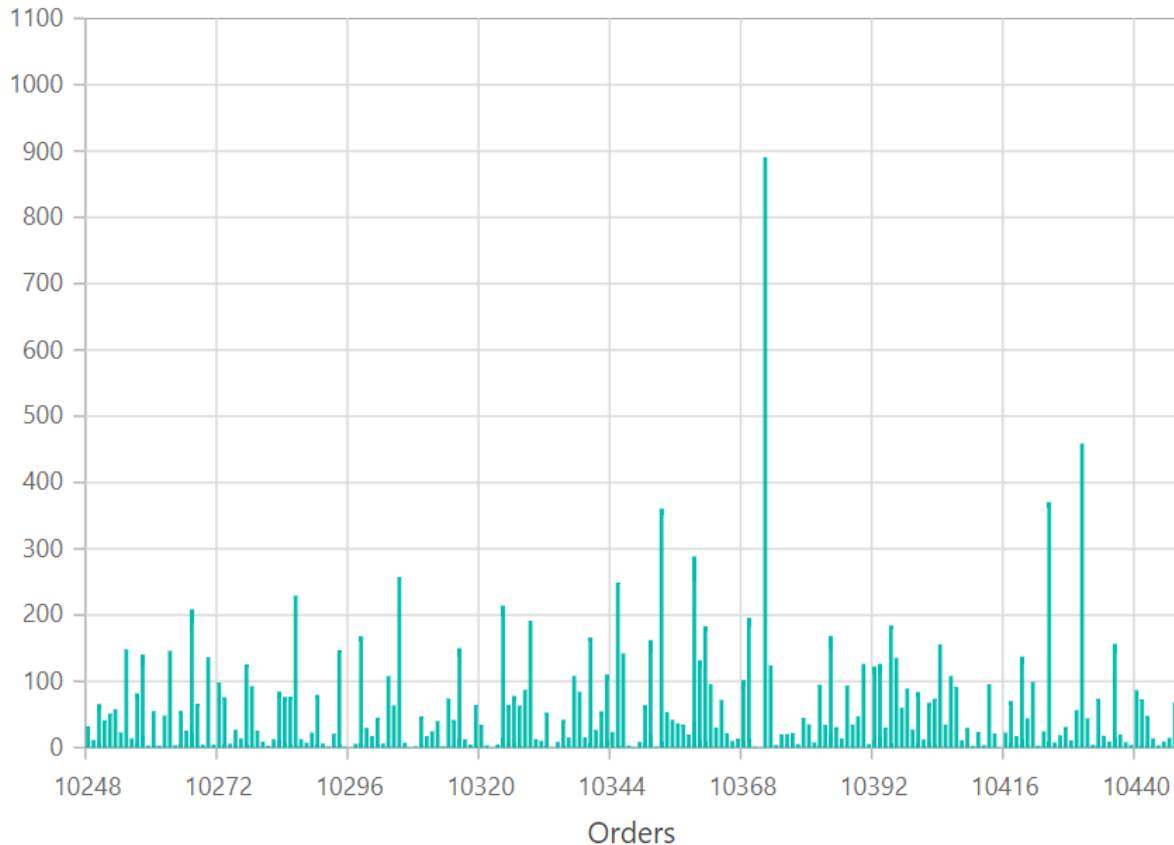
Remote Data

Assign service data as an instance of [SfDataManager](#) to the [DataSource](#) property to bind remote data to the chart component. Provide the endpoint Url to communicate with a remote data source.

ASPX-CS

```
@using Syncfusion.Blazor.Data
@using Syncfusion.Blazor.Charts
<SfChart>
<SfDataManager
    Url="https://services.odata.org/V4/Northwind/Northwind.svc/Orders"
    Adaptor="Adaptors.ODataV4Adaptor"></SfDataManager>
<ChartPrimaryXAxis Title="Orders"
    ValueType="Syncfusion.Blazor.Charts.ValueType.Category"
    RangePadding="ChartRangePadding.Additional"></ChartPrimaryXAxis>
<ChartSeriesCollection>
```

```
<ChartSeries XName="OrderID" YName="Freight"
Type="ChartSeriesType.Column"></ChartSeries>
</ChartSeriesCollection>
</SfChart>
```



Binding with OData services

[OData](#) is a standardized data creation and consumption protocol. The [SfDataManager](#) can be used to retrieve data from an [OData](#) service. For remote data binding using the [OData](#) service, see the code below.

Binding with OData v4 services

The [SfDataManager](#) can retrieve and consume OData v4 services, which is an upgraded version of OData protocols. Please refer to the [OData documentation](#) for additional information on OData v4 services. To bind an OData v4 service, use the **ODataV4Adaptor**.

Web API

The [WebApiAdaptor](#) can be used to bind a chart to a Web API created using an [OData](#) endpoint.

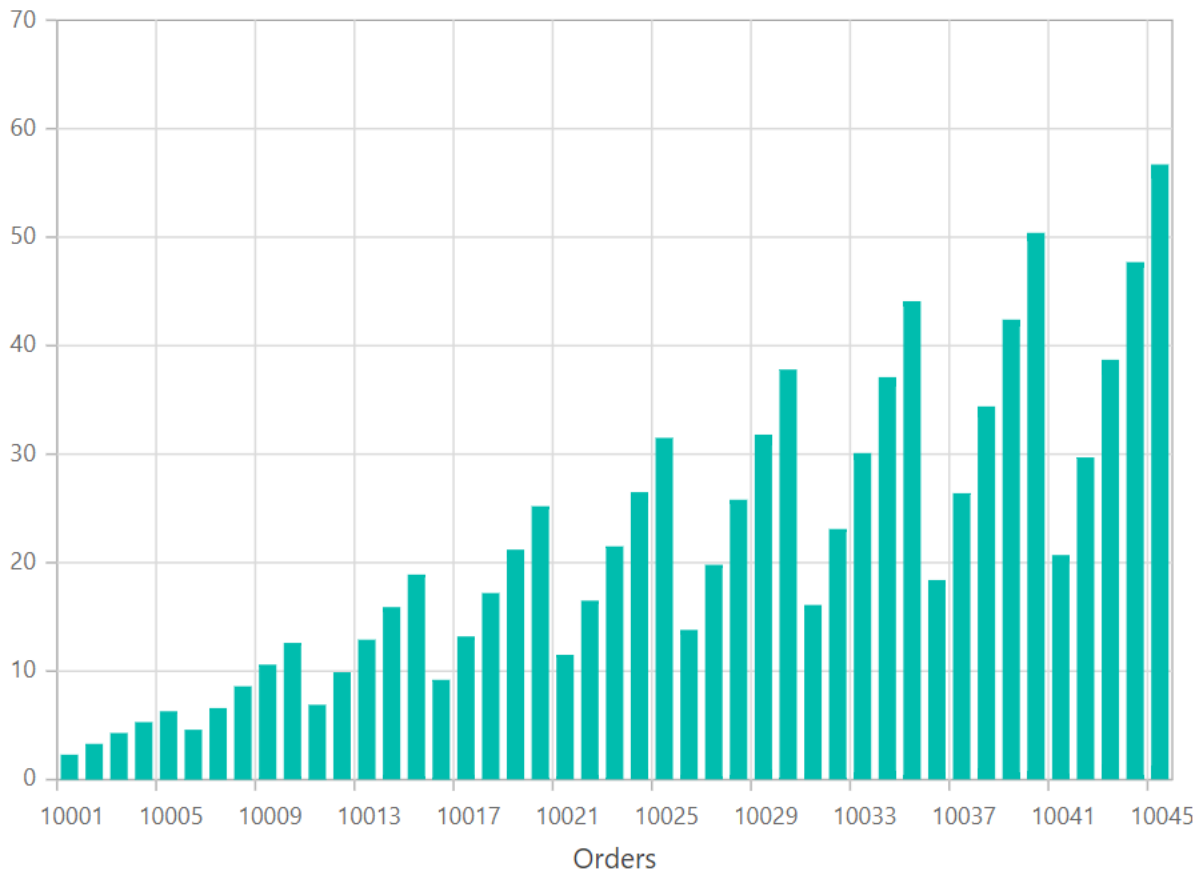
ASPX-CS

```
@using Syncfusion.Blazor.Data
@using Syncfusion.Blazor.Charts
<SfChart>
<SfDataManager Url="https://ej2services.syncfusion.com/production/web-
services/api/Orders" Adaptor="Adaptors.WebApiAdaptor"></SfDataManager>
```

```

<ChartPrimaryXAxis Title="Orders"
ValueType="Syncfusion.Blazor.Charts.ValueType.Category"
></ChartPrimaryXAxis>
<ChartSeriesCollection>
<ChartSeries XName="OrderID" YName="Freight"
Type="ChartSeriesType.Column"></ChartSeries>
</ChartSeriesCollection>
</SfChart>

```



Sending additional parameters to the server

To create a data request with a custom parameter, add additional parameters to the [Query](#) object and assign it to the chart's Query property.

The following sample code shows how to send parameters using the Query property in the series.

ASPX-CS

```

@using Syncfusion.Blazor.Data
@using Syncfusion.Blazor.Charts
<SfChart>
<SfDataManager
Url="https://services.odata.org/V4/Northwind/Northwind.svc/Orders"
Adaptor="Adaptors.ODataV4Adaptor"></SfDataManager>
<ChartPrimaryXAxis Title="Orders"
ValueType="Syncfusion.Blazor.Charts.ValueType.Category"
RangePadding="ChartRangePadding.Additional"></ChartPrimaryXAxis>

```

```

<ChartSeriesCollection>
<ChartSeries Query="ChartQuery" XName="OrderID" YName="Freight"
Type="ChartSeriesType.Column"></ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code{
public Query ChartQuery { get; set; }
protected override void OnInitialized()
{
ChartQuery = new Query().Take(10).Where("Freight", "GreaterThan", 300,
false);
}
}

```

Entity Framework

Entity Framework acts as a modern object-database mapper for .NET. This section explains how to bind data to the chart component from a **Microsoft SQL Server** database.

Create DbContext class

To connect to a Microsoft SQL Server database, the first step is to construct a DbContext class named **OrderContext**.

CSHARP

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.EntityFrameworkCore;
using System.ComponentModel.DataAnnotations;
using EFChart.Data;
namespace EFChart.Data
{
public class OrderContext : DbContext
{
public virtual DbSet<Order> Orders { get; set; }
protected override void OnConfiguring(DbContextOptionsBuilder
optionsBuilder)
{
if (!optionsBuilder.IsConfigured)
{
// Configures the context to connect to a Microsoft SQL Serve database
optionsBuilder.UseSqlServer(@"Data
Source=(LocalDB)\MSSQLLocalDB;AttachDbFilename='D:\blazor\EFTreeMap\App_Data
\NORTHWND.MDF';Integrated Security=True;Connect Timeout=30");
}
}
}
public class Order
{
[Key]
public int? OrderID { get; set; }
[Required]
public string CustomerID { get; set; }
[Required]
public int EmployeeID { get; set; }
}
}

```

```
}
}
```

Create data access layer to perform data operation

Then construct a class named **OrderDataAccessLayer** that will serve as a data access layer for retrieving records from the database table.

CSHARP

```
using Microsoft.EntityFrameworkCore;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using static BlazorApp1.Data.OrderContext;
using EFChart.Data;
namespace EFChart.Data
{
    public class OrderDataAccessLayer
    {
        OrderContext db = new OrderContext();
        //To Get all Orders details
        public DbSet<Order> GetAllOrders()
        {
            try
            {
                return db.Orders;
            }
            catch
            {
                throw;
            }
        }
    }
}
```

Creating Web API Controller

Need to create a Web API Controller that allows the chart to receive data directly from the Entity Framework.

CSHARP

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Primitives;
using static BlazorApp1.Data.OrderContext;
using EFChart.Data;
namespace EFChart.Controller
{
    [Route("api/[controller]")]
```



```

[ApiController]
public class DefaultController : ControllerBase
{
    OrderDataAccessLayer db = new OrderDataAccessLayer();
    [HttpGet]
    public object Get()
    {
        IQueryable<Order> data = db.GetAllOrders().AsQueryable();
        var count = data.Count();
        var queryString = Request.Query;
        if (queryString.Keys.Contains("$inlinecount"))
        {
            StringValues Skip;
            StringValues Take;
            int skip = (queryString.TryGetValue("$skip", out Skip)) ?
            Convert.ToInt32(Skip[0]) : 0;
            int top = (queryString.TryGetValue("$top", out Take)) ?
            Convert.ToInt32(Take[0]) : data.Count();
            return new { Items = data.Skip(skip).Take(top), Count = count };
        }
        else
        {
            return data;
        }
    }
}

```

Add Web API Controller services in Startup.cs

As shown below, open the **Startup.cs** file and add the services and endpoints necessary for Web API Controller.

C# SHARP

```

using EFChart.Data;
using Newtonsoft.Json.Serialization;
namespace BlazorApplication
{
    public class Startup
    {
        ....
        ....
        public void ConfigureServices(IServiceCollection services)
        {
            ....
            ....
            services.AddSingleton<OrderDataAccessLayer>();
            // Adds services for controllers to the specified
            Microsoft.Extensions.DependencyInjection.IServiceCollection.
            services.AddControllers().AddNewtonsoftJson(options =>
            {
                options.SerializerSettings.ContractResolver = new DefaultContractResolver();
            });
        }
        public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
        {

```

```

.....
.....
app.UseEndpoints(endpoints =>
{
// Adds endpoints for controller actions to the
Microsoft.AspNetCore.Routing.IEndpointRouteBuilder
endpoints.MapDefaultControllerRoute();
.....
.....
});
}
}
}
}

```

Configure chart component

Use the [DataSource](#) property or [SfDataManager](#) to bind data to the chart.

For instance, to bind data directly from the data access layer class **OrderDataAccessLayer**, assign the [DataSource](#) property to be **OrderData.GetAllOrders()**.

ASPX-CS

```

@using EFChart.Data;
@inject OrderDataAccessLayer OrderData;
@using Syncfusion.Blazor.Charts
<SfChart DataSource="@OrderData.GetAllOrders()">
<ChartPrimaryXAxis ValueType="Syncfusion.Blazor.Charts.ValueType.Category">
</ChartPrimaryXAxis>
<ChartSeriesCollection>
<ChartSeries XName="CustomerID" YName="OrderID"
Type="ChartSeriesType.Column">
</ChartSeries>
</ChartSeriesCollection>
</SfChart>

```

On the other hand, to configure the chart using Web API, provide the appropriate endpoint Url within [SfDataManager](#) along with [Adaptor](#). In order to interact with the Web API and consume data from the Entity Framework effectively, [WebApiAdaptor](#) must be used.

ASPX-CS

```

@using Syncfusion.Blazor.Charts
@using Syncfusion.Blazor.Data
<SfChart>
<SfDataManager Url="api/Default"
Adaptor="Syncfusion.Blazor.Adaptors.WebApiAdaptor">
</SfDataManager>
<ChartPrimaryXAxis ValueType="Syncfusion.Blazor.Charts.ValueType.Category">
</ChartPrimaryXAxis>
<ChartSeriesCollection>
<ChartSeries XName="CustomerID" YName="OrderID"
Type="ChartSeriesType.Column">
</ChartSeries>
</ChartSeriesCollection>
</SfChart>

```

Empty points

Empty points are defined as data points having NaN values. Empty points can be customized using [EmptyPointSettings](#) property in series. Default Empty Point [Mode](#) is [Gap](#).

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart>
<ChartPrimaryXAxis ValueType="Syncfusion.Blazor.Charts.ValueType.Category">
</ChartPrimaryXAxis>
<ChartSeriesCollection>
<ChartSeries DataSource="@SalesReports" XName="Month" YName="Sales"
Type="ChartSeriesType.Column">
<ChartEmptyPointSettings Fill="blue" Mode="@Mode">
</ChartEmptyPointSettings>
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code{
public class ChartData
{
public string Month { get; set; }
public Nullable<double> Sales { get; set; }
}
public EmptyPointMode Mode = EmptyPointMode.Average;
public List<ChartData> SalesReports = new List<ChartData>
{
new ChartData{ Month= "Jan", Sales= 35 },
new ChartData{ Month= "Feb", Sales= 28 },
new ChartData{ Month= "Mar", Sales= double.NaN },
new ChartData{ Month= "Apr", Sales= 32 },
new ChartData{ Month= "May", Sales= 40 },
new ChartData{ Month= "Jun", Sales= 32 },
new ChartData{ Month= "Jul", Sales= 35 },
new ChartData{ Month= "Aug", Sales= double.NaN },
new ChartData{ Month= "Sep", Sales= 38 },
new ChartData{ Month= "Oct", Sales= 30 },
new ChartData{ Month= "Nov", Sales= 25 },
new ChartData{ Month= "Dec", Sales= 32 }
};
}
```

Customizing empty point

The [Fill](#) property specifies the color of [EmptyPointSettings](#) and the [ChartEmptyPointBorder](#) specifies [Color](#) and [Width](#) of border for [EmptyPointSettings](#).

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart>
<ChartPrimaryXAxis ValueType="Syncfusion.Blazor.Charts.ValueType.Category">
</ChartPrimaryXAxis>
<ChartSeriesCollection>
<ChartSeries DataSource="@SalesReports" XName="Month" YName="Sales"
Type="ChartSeriesType.Column">
<ChartEmptyPointSettings Fill="red" Mode="@Mode">
</ChartEmptyPointSettings>
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
```

```
<ChartEmptyPointBorder Color="black" Width="2"></ChartEmptyPointBorder>
</ChartEmptyPointSettings>
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code{
public class ChartData
{
public string Month { get; set; }
public Nullable<double> Sales { get; set; }
}
public EmptyPointMode Mode = EmptyPointMode.Average;
public List<ChartData> SalesReports = new List<ChartData>
{
new ChartData{ Month= "Jan", Sales= 35 },
new ChartData{ Month= "Feb", Sales= 28 },
new ChartData{ Month= "Mar", Sales= double.NaN },
new ChartData{ Month= "Apr", Sales= 32 },
new ChartData{ Month= "May", Sales= 40 },
new ChartData{ Month= "Jun", Sales= 32 },
new ChartData{ Month= "Jul", Sales= 35 },
new ChartData{ Month= "Aug", Sales= double.NaN },
new ChartData{ Month= "Sep", Sales= 38 },
new ChartData{ Month= "Oct", Sales= 30 },
new ChartData{ Month= "Nov", Sales= 25 },
new ChartData{ Month= "Dec", Sales= 32 }
};
}
```



Refer to our [Blazor Charts](#) feature tour page for its groundbreaking feature representations and also explore our [Blazor Chart Example](#) to know various chart types and how to represent time-dependent data, showing trends at equal intervals.

See Also

- [Data label](#)
- [Tooltip](#)
- [Marker](#)

Chart Dimensions in Blazor Charts Component

* When no size is specified, the default height and width are 450px and 600px, respectively.

* To avoid delayed rendering, architectural changes were made to the Chart when the width/height were specified [in percentages](#) or [through style settings](#) applied in the component's parent. As a result, the Chart is initially rendered with the default width and height and then redrawn by adjusting only the size of the Chart in a responsive manner. By including the following script in the header tag, the redrawn scenario can now be avoided.

HTML

```
<head>  
...
```

```

...
<script
src="https://cdnjs.cloudflare.com/ajax/libs/lodash.js/4.17.20/lodash.min.js"
></script>
<!-- To avoid the redraw scenario, add the CDN link below. --->
<script src="https://cdn.syncfusion.com/blazor/syncfusion-blazor-
base.min.js"></script>
</head>

```

Size for Container

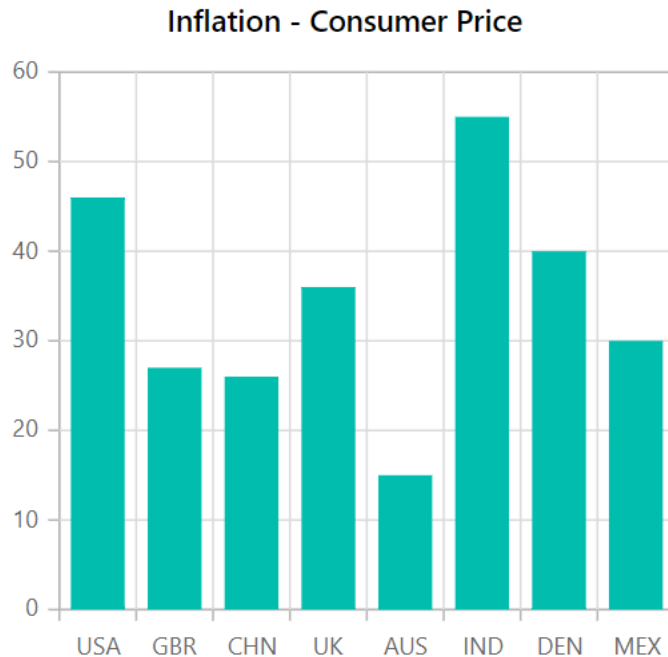
The chart can be scaled to fit the container. As shown below, the size can be set using CSS.

ASPX-CS

```

@using Syncfusion.Blazor.Charts
<div style="width:350px; height:350px; background-color:red">
<SfChart Title="Inflation - Consumer Price">
<ChartPrimaryXAxis
ValueType="Syncfusion.Blazor.Charts.ValueType.Category"></ChartPrimaryXAxis>
<ChartSeriesCollection>
<ChartSeries DataSource="@ConsumerDetails" XName="X" YName="YValue"
Type="ChartSeriesType.Column">
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
</div>
@code{
public class ChartData
{
public string X { get; set; }
public double YValue { get; set; }
}
public List<ChartData> ConsumerDetails = new List<ChartData>
{
new ChartData { X= "USA", YValue= 46 },
new ChartData { X= "GBR", YValue= 27 },
new ChartData { X= "CHN", YValue= 26 },
new ChartData { X= "UK", YValue= 36 },
new ChartData { X= "AUS", YValue= 15 },
new ChartData { X= "IND", YValue= 55 },
new ChartData { X= "DEN", YValue= 40 },
new ChartData { X= "MEX", YValue= 30 }
};
}

```



Size for Chart

The [Width](#) and [Height](#) properties specify the size of the chart in pixels or percentages directly.

In Pixel

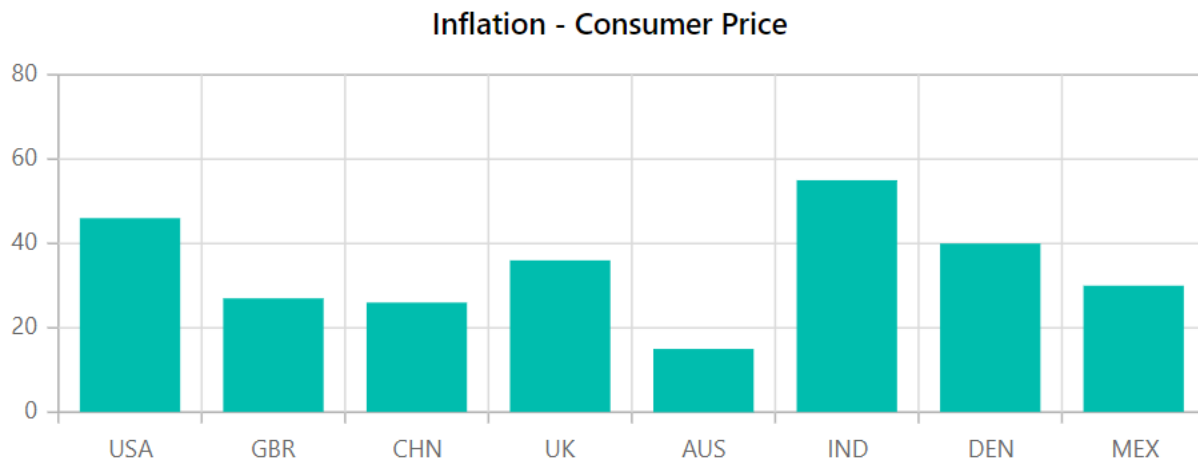
The [Width](#) and [Height](#) properties can be set in pixel as shown below.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart Title="Inflation - Consumer Price" Width="650px" Height="250px">
  <ChartPrimaryXAxis
    ValueType="Syncfusion.Blazor.Charts.ValueType.Category"></ChartPrimaryXAxis>
  <ChartSeriesCollection>
    <ChartSeries DataSource="@ConsumerDetails" XName="X" YName="YValue"
      Type="ChartSeriesType.Column">
    </ChartSeries>
  </ChartSeriesCollection>
</SfChart>

@code{
public class ChartData
{
public string X { get; set; }
public double YValue { get; set; }
}
public List<ChartData> ConsumerDetails = new List<ChartData>
{
new ChartData { X= "USA", YValue= 46 },
new ChartData { X= "GBR", YValue= 27 },
new ChartData { X= "CHN", YValue= 26 },
new ChartData { X= "UK", YValue= 36 },
new ChartData { X= "AUS", YValue= 15 },
new ChartData { X= "IND", YValue= 55 },
new ChartData { X= "DEN", YValue= 40 },
new ChartData { X= "MEX", YValue= 30 }
}
```

```
};
}
```



In Percentage

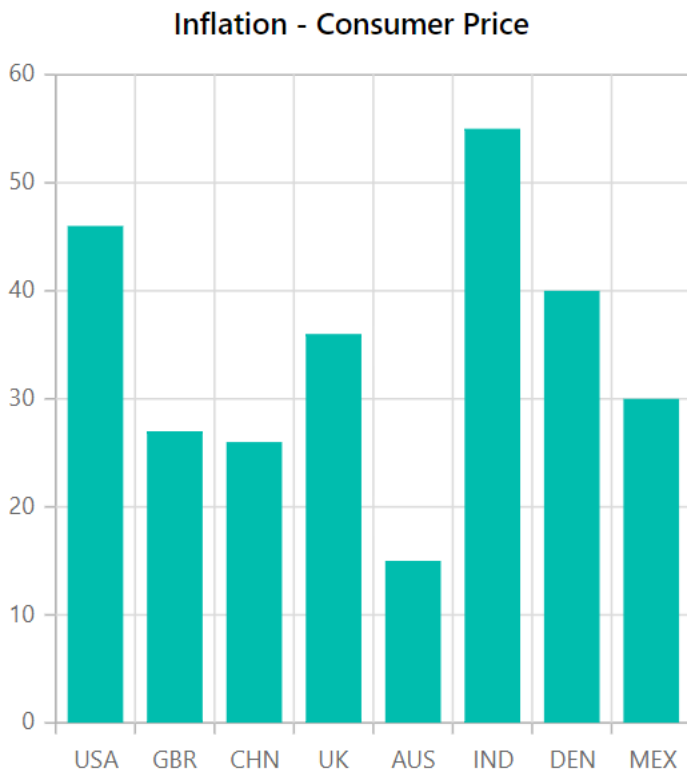
By setting the values of [Width](#) and [Height](#) properties in percentage, the chart gets its dimension with respect to its container. For example, when the [Height](#) is set to **50%**, the chart is half the height of its container.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart Title="Inflation - Consumer Price" Width="60%" height="90%">
  <ChartPrimaryXAxis
    ValueType="Syncfusion.Blazor.Charts.ValueType.Category"></ChartPrimaryXAxis>
  <ChartSeriesCollection>
    <ChartSeries DataSource="@ConsumerDetails" XName="X" YName="YValue"
      Type="ChartSeriesType.Column">
    </ChartSeries>
  </ChartSeriesCollection>
</SfChart>

@code{
public class ChartData
{
    public string X { get; set; }
    public double YValue { get; set; }
}

public List<ChartData> ConsumerDetails = new List<ChartData>
{
    new ChartData { X= "USA", YValue= 46 },
    new ChartData { X= "GBR", YValue= 27 },
    new ChartData { X= "CHN", YValue= 26 },
    new ChartData { X= "UK", YValue= 36 },
    new ChartData { X= "AUS", YValue= 15 },
    new ChartData { X= "IND", YValue= 55 },
    new ChartData { X= "DEN", YValue= 40 },
    new ChartData { X= "MEX", YValue= 30 }
};
}
```

Refer to our [Blazor Charts](#) feature tour page for its groundbreaking feature representations and also explore our [Blazor Chart Example](#) to know various chart types and how to represent time-dependent data, showing trends at equal intervals.

See Also

- [Data Label](#)
- [Tooltip](#)
- [Marker](#)

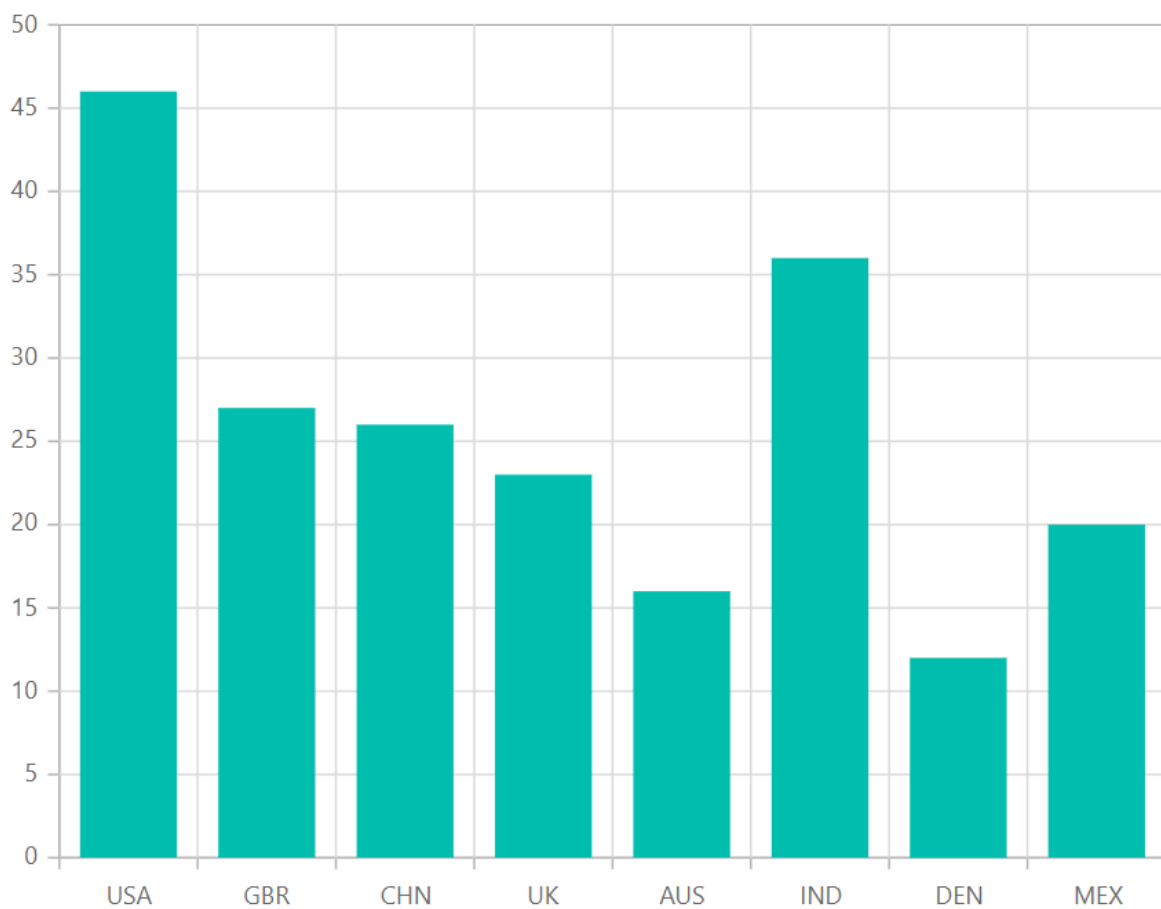
Category Axis in Blazor Charts Component

The category axis is used to represent string values instead of integers.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart>
  <ChartPrimaryXAxis ValueType="Syncfusion.Blazor.Charts.ValueType.Category">
  </ChartPrimaryXAxis>
  <ChartSeriesCollection>
    <ChartSeries DataSource="@MedalDetails" XName="X" YName="YValue"
    Type="ChartSeriesType.Column">
    </ChartSeries>
  </ChartSeriesCollection>
</SfChart>
@code{
public class ChartData
{
  public string X { get; set; }
}
```

```
public double YValue { get; set; }  
}  
public List<ChartData> MedalDetails = new List<ChartData>  
{  
    new ChartData { X= "USA", YValue= 46 },  
    new ChartData { X= "GBR", YValue= 27 },  
    new ChartData { X= "CHN", YValue= 26 },  
    new ChartData { X= "UK", YValue= 23 },  
    new ChartData { X= "AUS", YValue= 16 },  
    new ChartData { X= "IND", YValue= 36 },  
    new ChartData { X= "DEN", YValue= 12 },  
    new ChartData { X= "MEX", YValue= 20 },  
};  
}
```



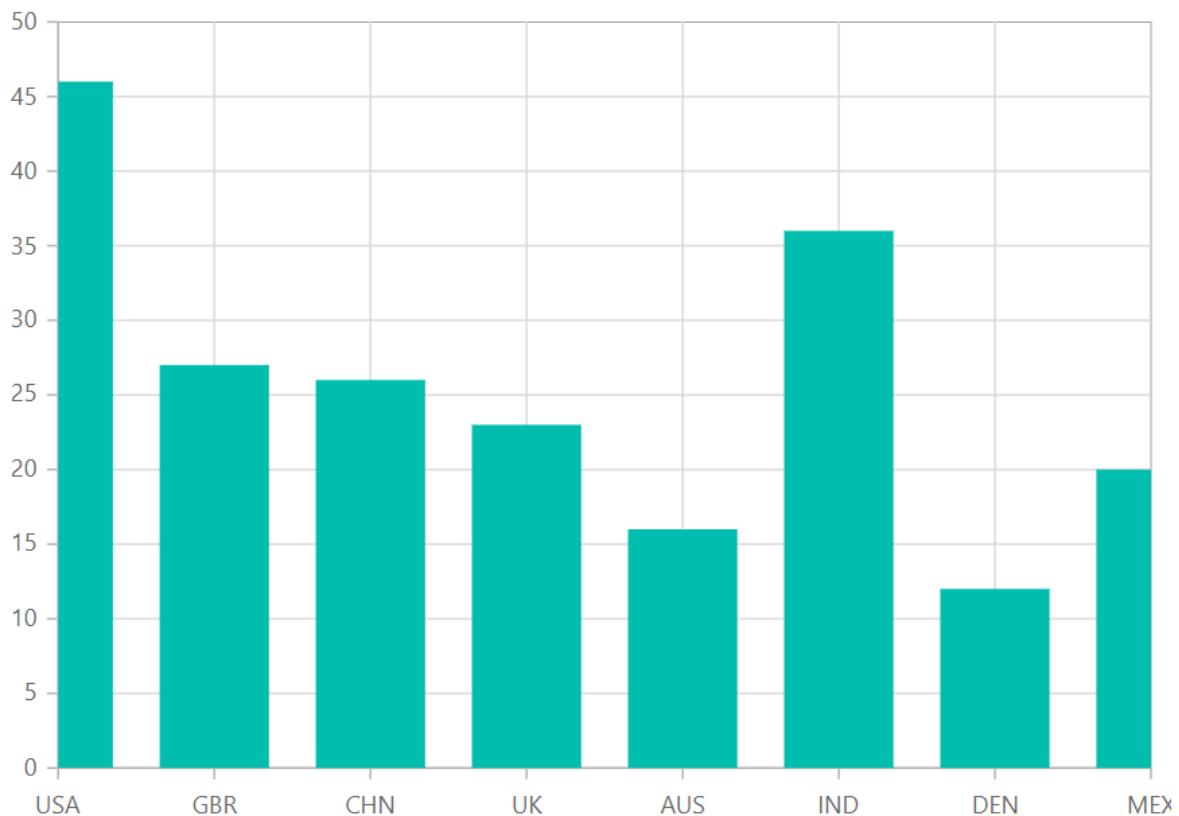
Labels Placement

The category labels are positioned between ticks by default, but the [LabelPlacement](#) property allows them to be placed on the ticks as well. The available options are [BetweenTicks](#) (default) and [OnTicks](#).

ASPX-CS

```
@using Syncfusion.Blazor.Charts  
<SfChart>
```

```
<ChartPrimaryXAxis LabelPlacement="LabelPlacement.OnTicks"
ValueType="Syncfusion.Blazor.Charts.ValueType.Category">
</ChartPrimaryXAxis>
<ChartSeriesCollection>
<ChartSeries DataSource="@MedalDetails" XName="X" YName="YValue"
Type="ChartSeriesType.Column">
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code{
public class ChartData
{
public string X { get; set; }
public double YValue { get; set; }
}
public List<ChartData> MedalDetails = new List<ChartData>
{
new ChartData { X= "USA", YValue= 46 },
new ChartData { X= "GBR", YValue= 27 },
new ChartData { X= "CHN", YValue= 26 },
new ChartData { X= "UK", YValue= 23 },
new ChartData { X= "AUS", YValue= 16 },
new ChartData { X= "IND", YValue= 36 },
new ChartData { X= "DEN", YValue= 12 },
new ChartData { X= "MEX", YValue= 20 },
};
}
```

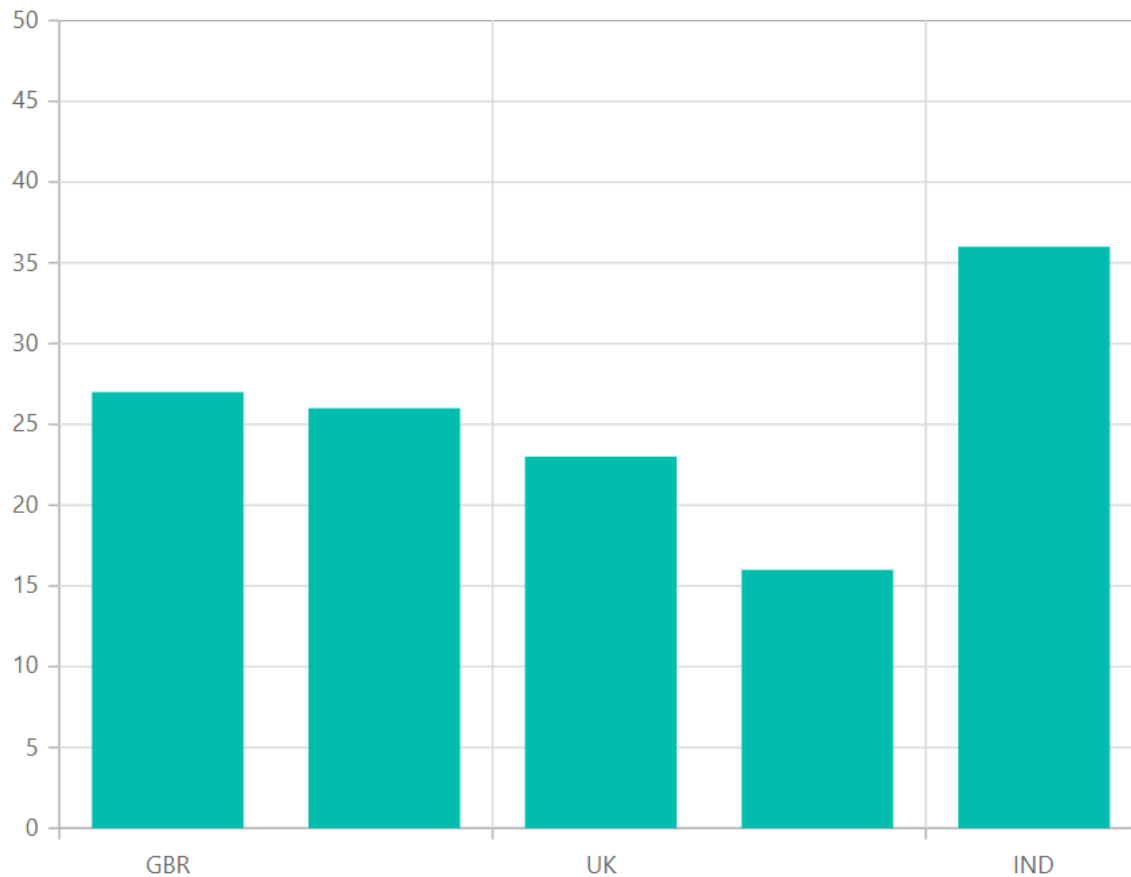


Range and Interval

The [Minimum](#), [Maximum](#), and [Interval](#) properties can be used to customize the range of the [Category](#) axis.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart>
<ChartPrimaryXAxis Maximum="5" Minimum="1" Interval="2"
ValueType="Syncfusion.Blazor.Charts.ValueType.Category">
</ChartPrimaryXAxis>
<ChartSeriesCollection>
<ChartSeries DataSource="@MedalDetails" XName="X" YName="YValue"
Type="ChartSeriesType.Column">
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code{
public class ChartData
{
public string X { get; set; }
public double YValue { get; set; }
}
public List<ChartData> MedalDetails = new List<ChartData>
{
new ChartData { X= "USA", YValue= 46 },
new ChartData { X= "GBR", YValue= 27 },
new ChartData { X= "CHN", YValue= 26 },
new ChartData { X= "UK", YValue= 23 },
new ChartData { X= "AUS", YValue= 16 },
new ChartData { X= "IND", YValue= 36 },
new ChartData { X= "DEN", YValue= 12 },
new ChartData { X= "MEX", YValue= 20 },
};
}
```



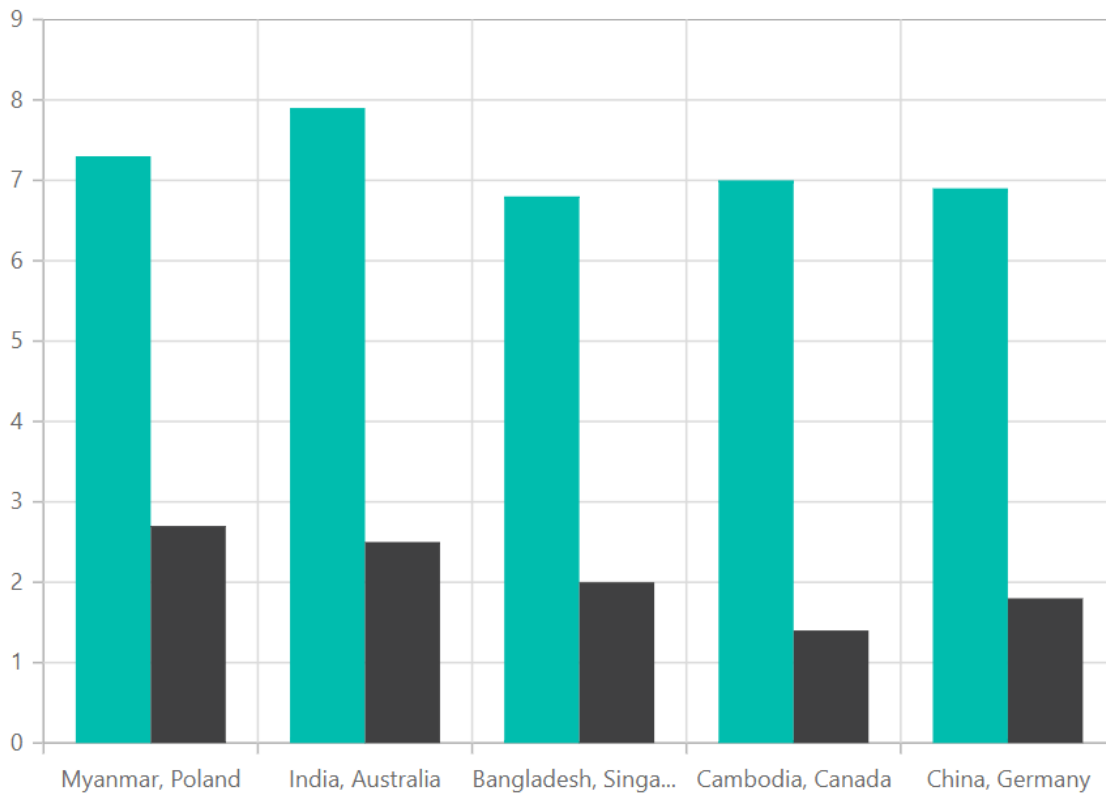
Indexed Category Axis

The category axis can also be rendered using the data source index values. This can be achieved by setting the [IsIndexed](#) property in the axis to **true**.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart>
  <ChartPrimaryXAxis IsIndexed="true"
  ValueType="Syncfusion.Blazor.Charts.ValueType.Category">
  </ChartPrimaryXAxis>
  <ChartSeriesCollection>
    <ChartSeries DataSource="@WeatherReports1" XName="X" YName="Y"
    Type="ChartSeriesType.Column">
    </ChartSeries>
    <ChartSeries DataSource="@WeatherReports2" XName="X" YName="Y"
    Type="ChartSeriesType.Column">
    </ChartSeries>
  </ChartSeriesCollection>
</SfChart>
@code{
public class WeatherData
{
public string X { get; set; }
public double Y { get; set; }
}
```

```
}  
public List<WeatherData> WeatherReports1 = new List<WeatherData>  
{  
    new WeatherData{ X= "Myanmar", Y= 7.3 },  
    new WeatherData{ X= "India", Y= 7.9 },  
    new WeatherData{ X= "Bangladesh", Y= 6.8 },  
    new WeatherData{ X= "Cambodia", Y=7.0 },  
    new WeatherData{ X= "China", Y= 6.9 }  
};  
public List<WeatherData> WeatherReports2 = new List<WeatherData>  
{  
    new WeatherData{ X= "Poland", Y=2.7 },  
    new WeatherData{ X= "Australia", Y=2.5 },  
    new WeatherData{ X= "Singapore", Y=2.0 },  
    new WeatherData{ X= "Canada", Y=1.4 },  
    new WeatherData{ X= "Germany", Y=1.8 }  
};  
}
```



Refer to our [Blazor Charts](#) feature tour page for its groundbreaking feature representations and also explore our [Blazor Chart Example](#) to know various chart types and how to represent time-dependent data, showing trends in data at equal intervals.

See Also

- [Data Label](#)
- [Tooltip](#)

- [Marker](#)

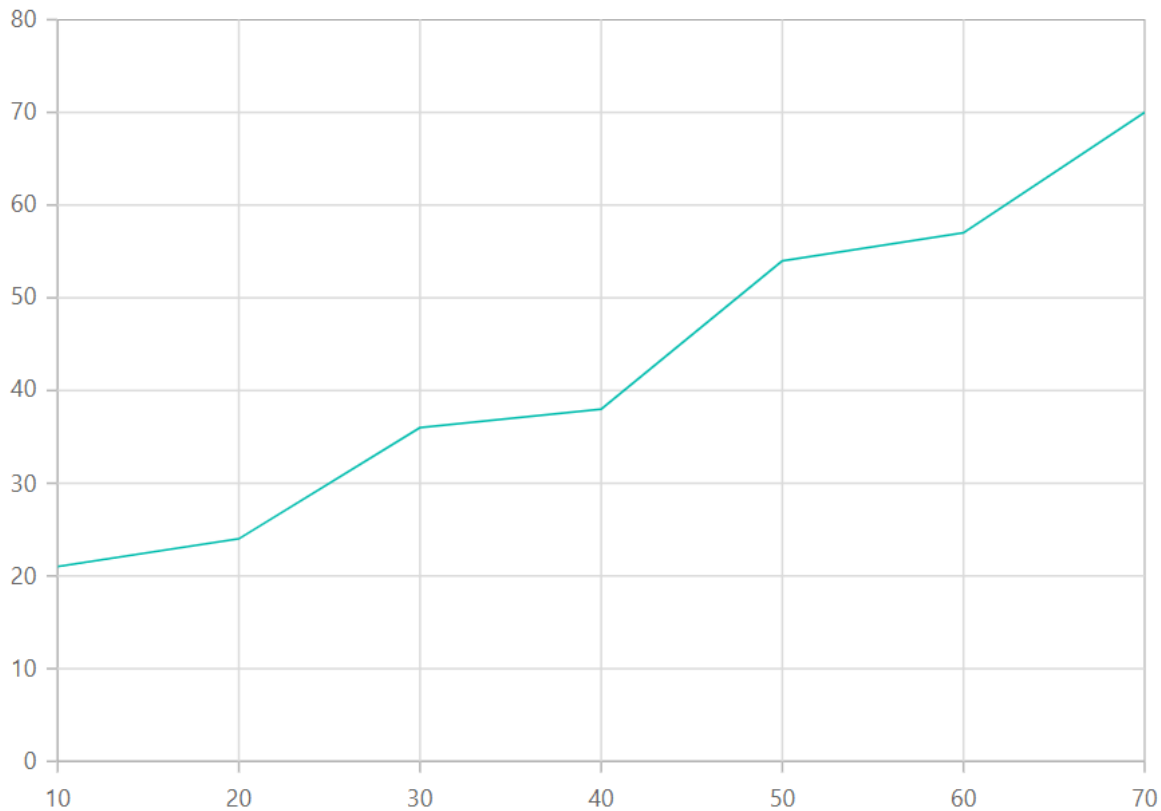
<!-- markdownlint-disable MD036 -->

Numeric Axis in Blazor Charts Component

Numeric axis can be used to represent numeric values in a chart. The [ValueType](#) of an axis is [Double](#) by default.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart>
<ChartSeriesCollection>
<ChartSeries DataSource="@Data" XName="XValue" YName="YValue">
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code{
public class ChartData
{
public double XValue { get; set; }
public double YValue { get; set; }
}
public List<ChartData> Data = new List<ChartData>
{
new ChartData { XValue = 10, YValue = 21 },
new ChartData { XValue = 20, YValue = 24 },
new ChartData { XValue = 30, YValue = 36 },
new ChartData { XValue = 40, YValue = 38 },
new ChartData { XValue = 50, YValue = 54 },
new ChartData { XValue = 60, YValue = 57 },
new ChartData { XValue = 70, YValue = 70 },
};
}
```



Range and Interval

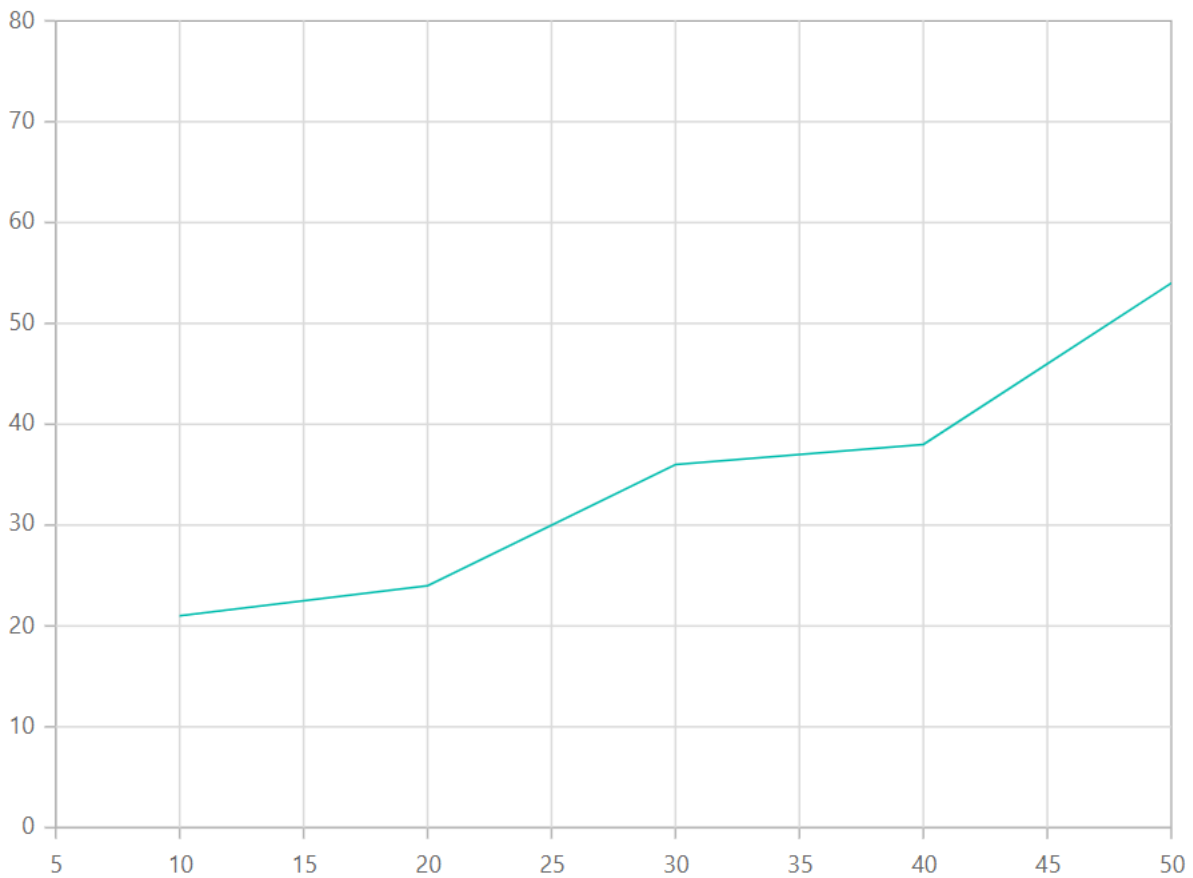
The axis range will be calculated automatically based on the provided data; however, the axis range can also be customized using [Minimum](#), [Maximum](#), and [Interval](#) properties.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart>
<ChartPrimaryXAxis Minimum="5" Maximum="50" Interval="2"/>
<ChartSeriesCollection>
<ChartSeries DataSource="@Data" XName="XValue" YName="YValue"/>
</ChartSeriesCollection>
</SfChart>
@code{
public class ChartData
{
public double XValue { get; set; }
public double YValue { get; set; }
}
public List<ChartData> Data = new List<ChartData>
{
new ChartData { XValue = 10, YValue = 21 },
new ChartData { XValue = 20, YValue = 24 },
new ChartData { XValue = 30, YValue = 36 },
new ChartData { XValue = 40, YValue = 38 },
new ChartData { XValue = 50, YValue = 54 },
new ChartData { XValue = 60, YValue = 57 },
new ChartData { XValue = 70, YValue = 70 },
}
```



```
};  
}
```



Range Padding

The [RangePadding](#) property can be used to apply padding to the minimum and maximum extremes of range. The following types of padding are supported by the numeric axis:

- None
- Round
- Additional
- Normal
- Auto

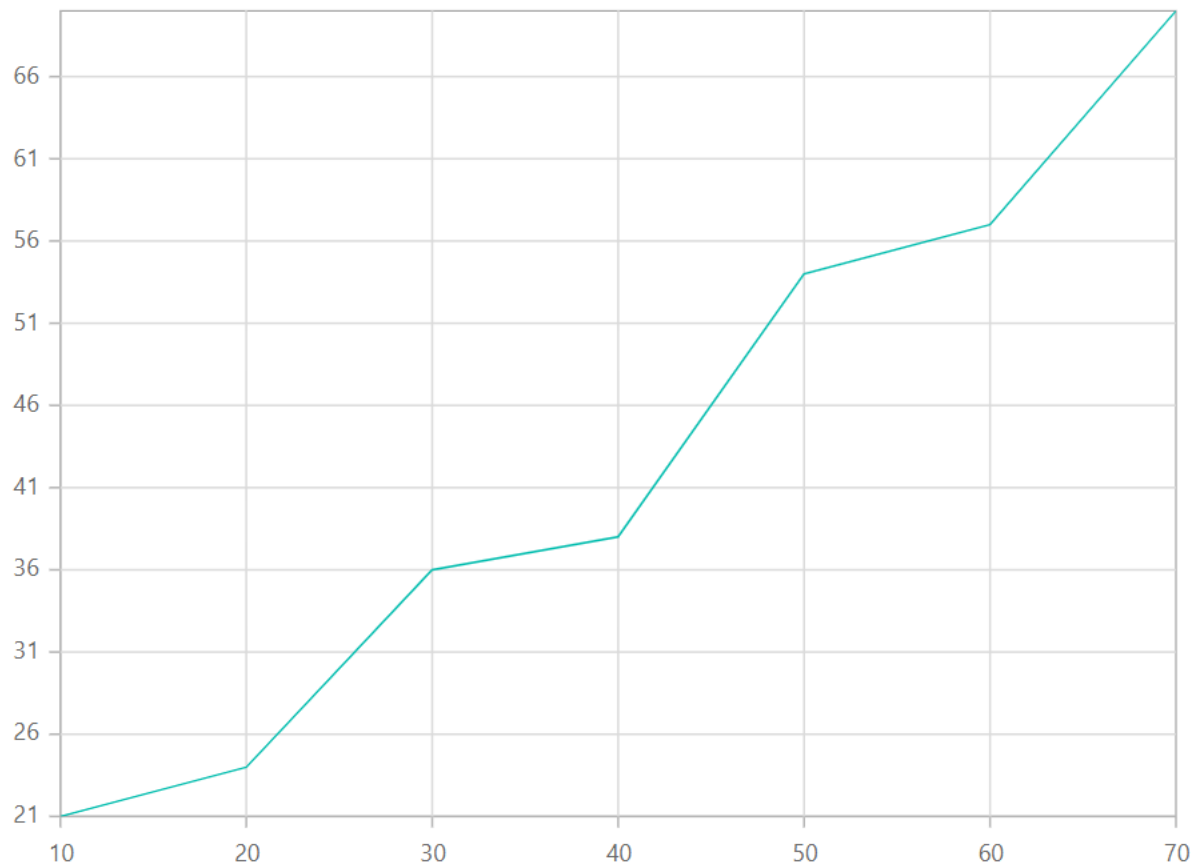
Numeric - None

When the [RangePadding](#) is set to **None**, the minimum and maximum of an axis is based on the data.

ASPX-CS

```
@using Syncfusion.Blazor.Charts  
<SfChart>  
<ChartPrimaryYAxis RangePadding="ChartRangePadding.None"/>  
<ChartSeriesCollection>  
<ChartSeries DataSource="@Data" XName="XValue" YName="YValue"/>  
</ChartSeriesCollection>
```

```
</SfChart>
@code{
public class ChartData
{
public double XValue { get; set; }
public double YValue { get; set; }
}
public List<ChartData> Data = new List<ChartData>
{
new ChartData { XValue = 10, YValue = 21 },
new ChartData { XValue = 20, YValue = 24 },
new ChartData { XValue = 30, YValue = 36 },
new ChartData { XValue = 40, YValue = 38 },
new ChartData { XValue = 50, YValue = 54 },
new ChartData { XValue = 60, YValue = 57 },
new ChartData { XValue = 70, YValue = 70 },
};
}
```

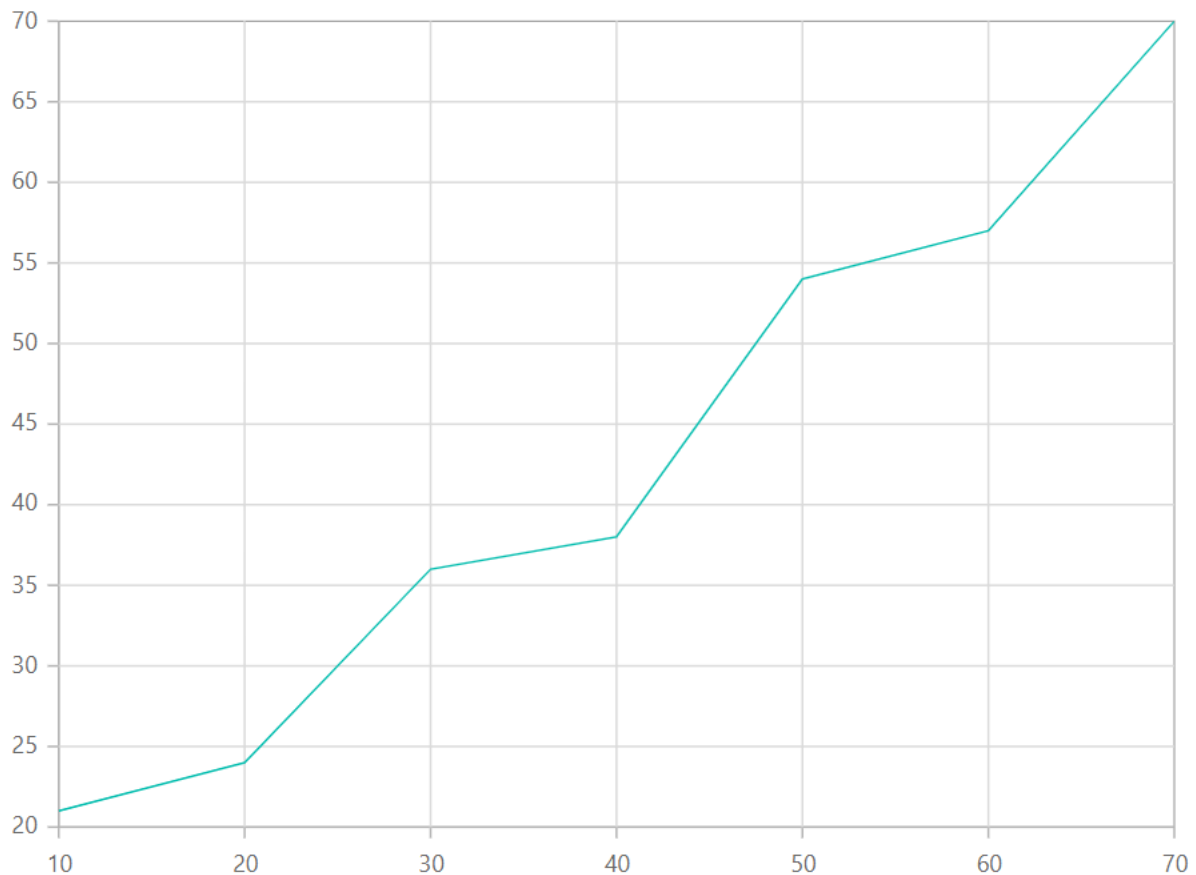


Numeric - Round

When the [RangePadding](#) is set to **Round**, the minimum and maximum will be rounded to the nearest possible value divisible by interval. For example, when the minimum is 3.5 and the interval is 1, then the minimum will be rounded to 3.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart>
<ChartPrimaryYAxis RangePadding="ChartRangePadding.Round"/>
<ChartSeriesCollection>
<ChartSeries DataSource="@Data" XName="XValue" YName="YValue"/>
</ChartSeriesCollection>
</SfChart>
@code{
public class ChartData
{
public double XValue { get; set; }
public double YValue { get; set; }
}
public List<ChartData> Data = new List<ChartData>
{
new ChartData { XValue = 10, YValue = 21 },
new ChartData { XValue = 20, YValue = 24 },
new ChartData { XValue = 30, YValue = 36 },
new ChartData { XValue = 40, YValue = 38 },
new ChartData { XValue = 50, YValue = 54 },
new ChartData { XValue = 60, YValue = 57 },
new ChartData { XValue = 70, YValue = 70 },
};
}
```

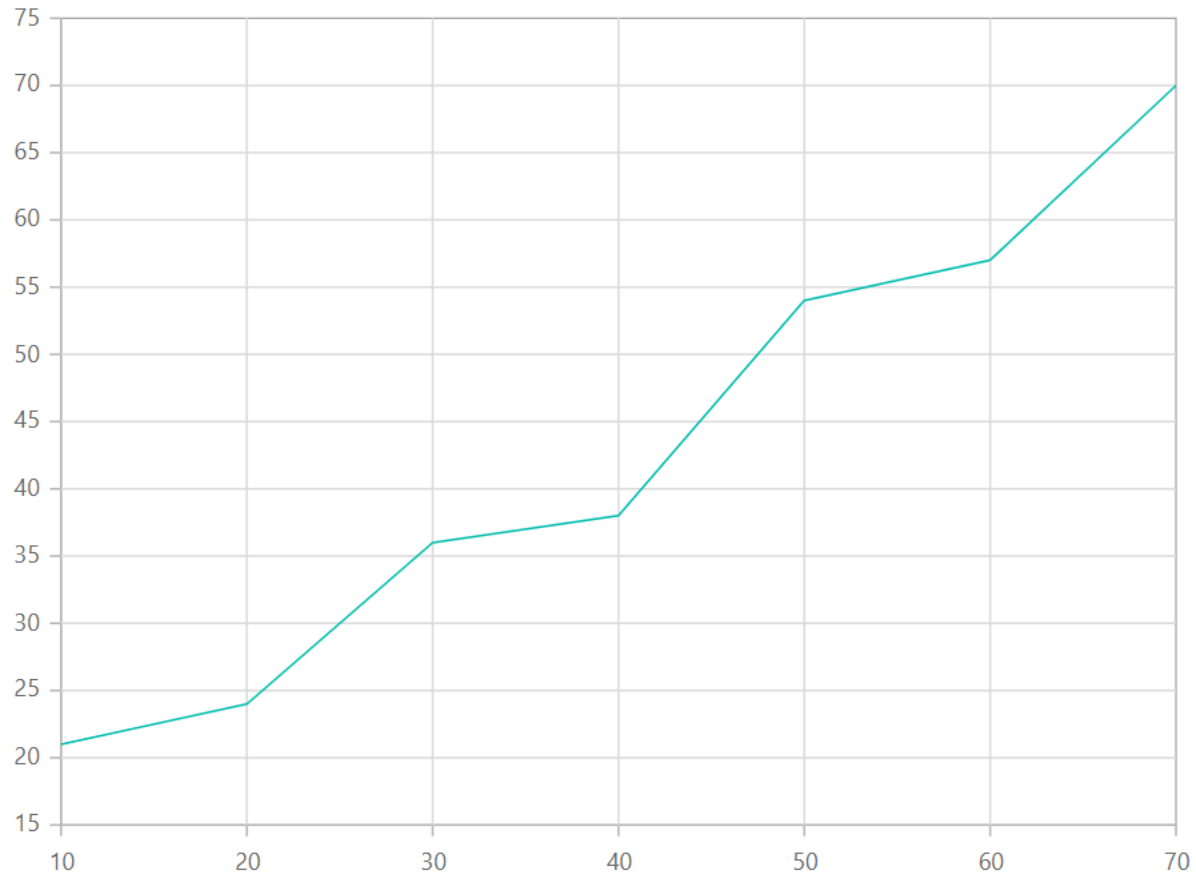


Numeric - Additional

When the [RangePadding](#) is set to **Additional**, interval of an axis will be padded to the minimum and maximum of the axis.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart>
<ChartPrimaryYAxis RangePadding="ChartRangePadding.Additional"/>
<ChartSeriesCollection>
<ChartSeries DataSource="@Data" XName="XValue" YName="YValue"/>
</ChartSeriesCollection>
</SfChart>
@code{
public class ChartData
{
public double XValue { get; set; }
public double YValue { get; set; }
}
public List<ChartData> Data = new List<ChartData>
{
new ChartData { XValue = 10, YValue = 21 },
new ChartData { XValue = 20, YValue = 24 },
new ChartData { XValue = 30, YValue = 36 },
new ChartData { XValue = 40, YValue = 38 },
new ChartData { XValue = 50, YValue = 54 },
new ChartData { XValue = 60, YValue = 57 },
new ChartData { XValue = 70, YValue = 70 },
};
}
```



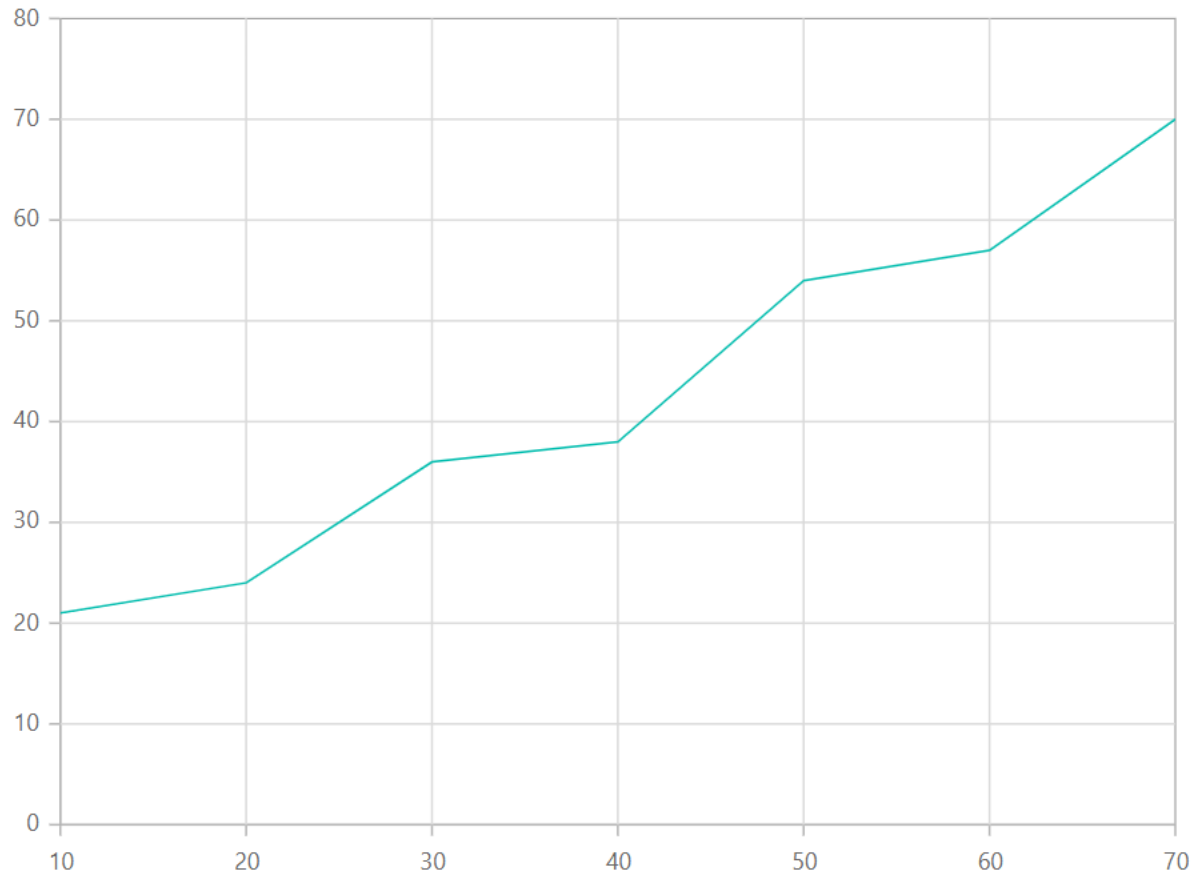
Numeric - Normal

When the [RangePadding](#) is set to **Normal**, padding is applied to the axis based on default range calculation.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart>
<ChartPrimaryYAxis RangePadding="ChartRangePadding.Normal"/>
<ChartSeriesCollection>
<ChartSeries DataSource="@Data" XName="XValue" YName="YValue"/>
</ChartSeriesCollection>
</SfChart>
@code{
public class ChartData
{
public double XValue { get; set; }
public double YValue { get; set; }
}
public List<ChartData> Data = new List<ChartData>
{
new ChartData { XValue = 10, YValue = 21 },
new ChartData { XValue = 20, YValue = 24 },
new ChartData { XValue = 30, YValue = 36 },
new ChartData { XValue = 40, YValue = 38 },
new ChartData { XValue = 50, YValue = 54 },
}
```

```
new ChartData { XValue = 60, YValue = 57 },  
new ChartData { XValue = 70, YValue = 70 },  
};  
}
```



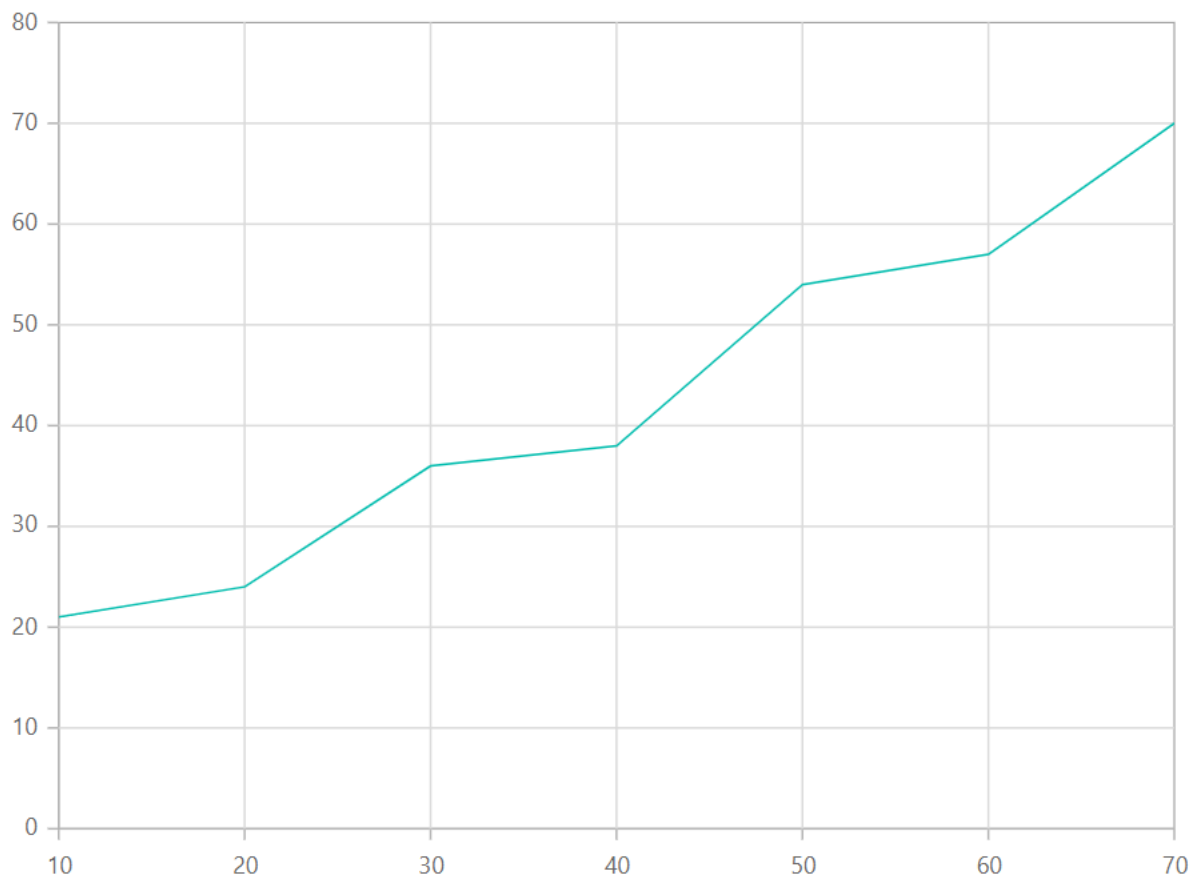
Numeric - Auto

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.

ASPX-CS

```
@using Syncfusion.Blazor.Charts  
<SfChart>  
<ChartPrimaryYAxis RangePadding="ChartRangePadding.Auto"/>  
<ChartPrimaryXAxis RangePadding="ChartRangePadding.Auto"/>  
<ChartSeriesCollection>  
<ChartSeries DataSource="@Data" XName="XValue" YName="YValue"/>  
</ChartSeriesCollection>  
</SfChart>  
@code{  
public class ChartData  
{  
    public double XValue { get; set; }  
    public double YValue { get; set; }  
}
```

```
public List<ChartData> Data = new List<ChartData>
{
    new ChartData { XValue = 10, YValue = 21 },
    new ChartData { XValue = 20, YValue = 24 },
    new ChartData { XValue = 30, YValue = 36 },
    new ChartData { XValue = 40, YValue = 38 },
    new ChartData { XValue = 50, YValue = 54 },
    new ChartData { XValue = 60, YValue = 57 },
    new ChartData { XValue = 70, YValue = 70 },
};
```



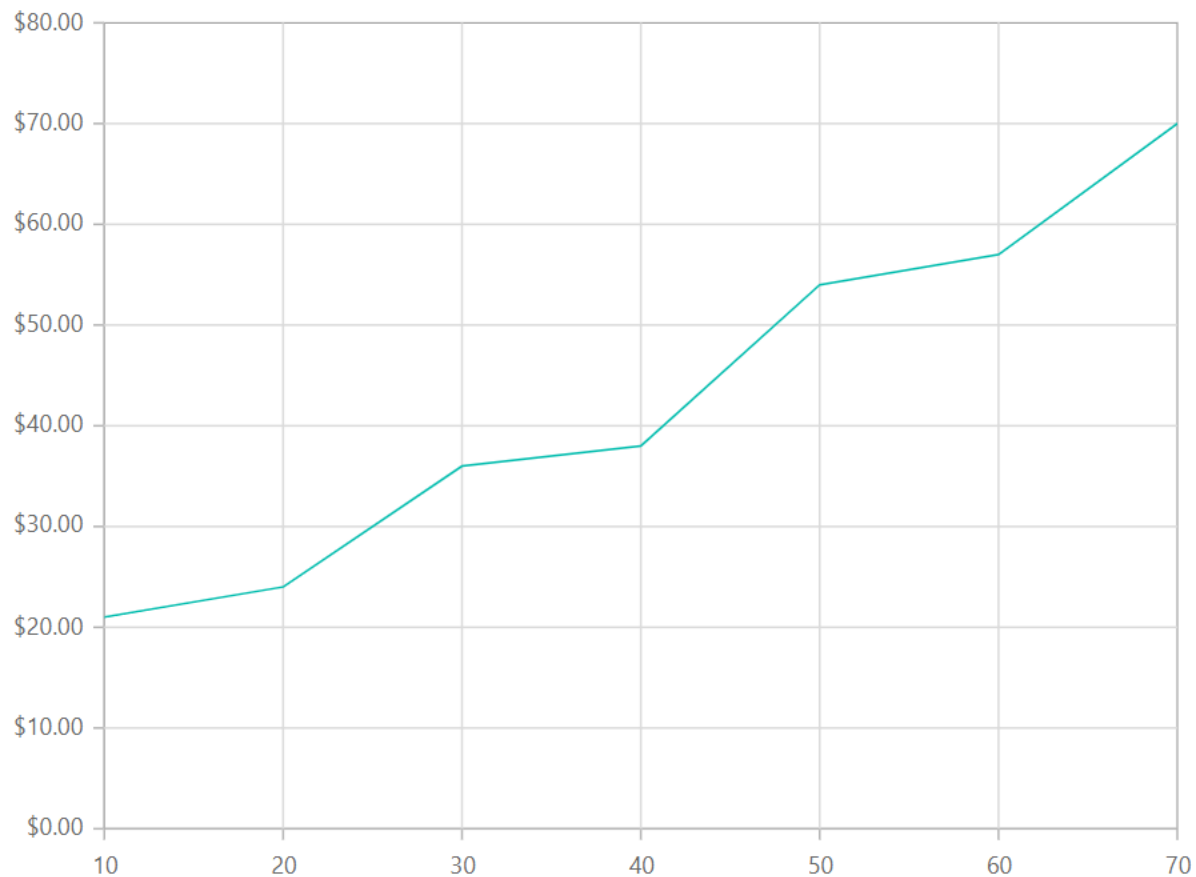
Label Format

Using the [LabelFormat](#) property on an axis, it is possible to format the numeric labels to all globalize formats.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart Title="Sales Comparison">
  <ChartPrimaryYAxis LabelFormat="c"/>
  <ChartSeriesCollection>
    <ChartSeries DataSource="@Data" XName="X" YName="Y"
      Type="ChartSeriesType.Column"/>
  </ChartSeriesCollection>
</SfChart>
```

```
@code{
public class ChartData
{
    public double X { get; set; }
    public double Y { get; set; }
}
public List<ChartData> Data = new List<ChartData>
{
    new ChartData{ X= 10, Y=7000 },
    new ChartData{ X= 20, Y= 1000 },
    new ChartData{ X= 30, Y= 12000 },
    new ChartData{ X= 40, Y= 14000 },
    new ChartData{ X= 50, Y= 11000 },
    new ChartData{ X= 60, Y= 5000 },
    new ChartData{ X= 70, Y= 7300 },
    new ChartData{ X= 80, Y= 9000 },
    new ChartData{ X= 90, Y= 12000 },
    new ChartData{ X= 100, Y= 14000 },
    new ChartData{ X= 110, Y= 11000 },
    new ChartData{ X= 120, Y= 5000 }
};
}
```



The table below shows the results of applying various commonly used label formats to numeric data.

<!-- markdownlint-disable MD033 -->

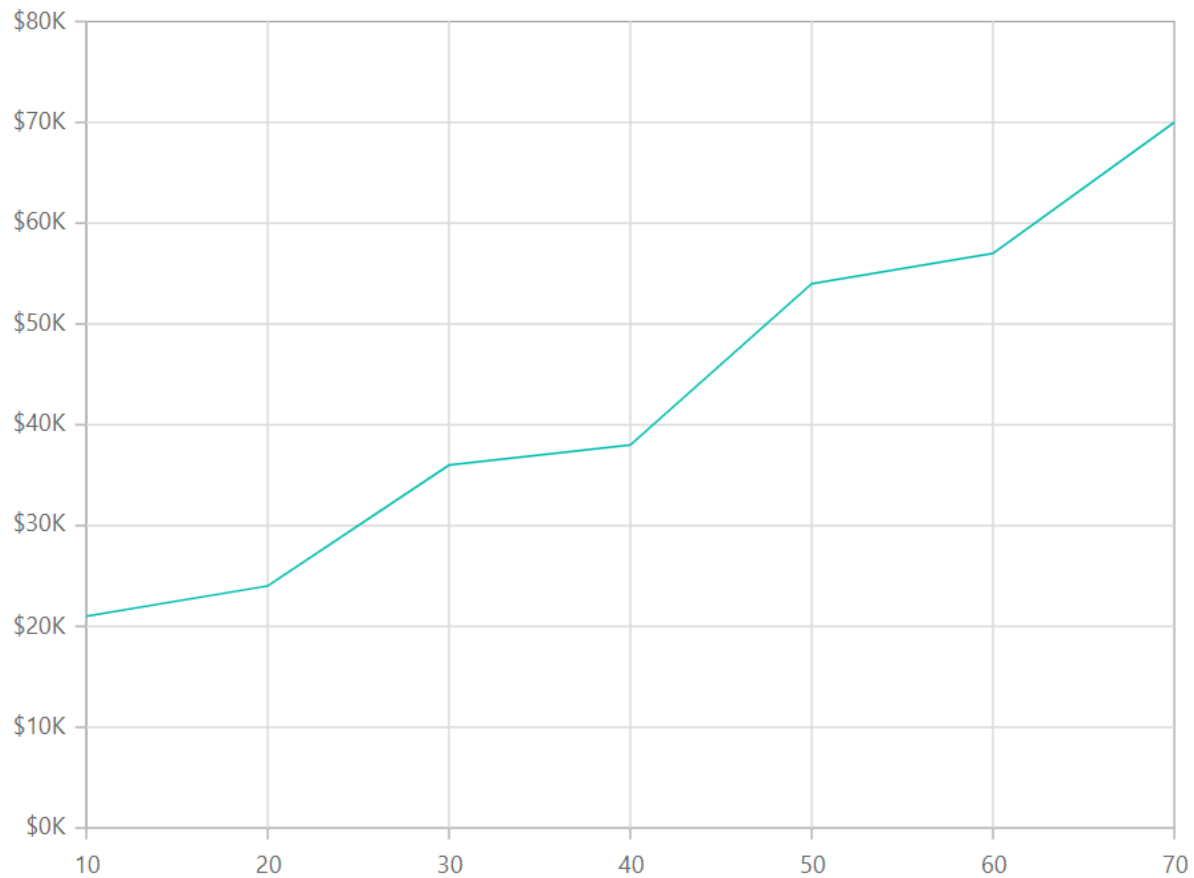
Label Value	Label Format property value	Result	Description
1000	n1	1000.0	The number is rounded to 1 decimal place.
1000	n2	1000.00	The number is rounded to 2 decimal place.
1000	n3	1000.000	The number is rounded to 3 decimal place.
0.01	p1	1.0%	The number is converted to percentage with 1 decimal place.
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.

Custom Label Format

Axis also supports custom label format using placeholders such as {value}K, where the value represents the axis label, for example, 20K.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart>
<ChartPrimaryYAxis LabelFormat="{value}K"
RangePadding="ChartRangePadding.Auto"/>
<ChartPrimaryXAxis RangePadding="ChartRangePadding.Auto"/>
<ChartSeriesCollection>
<ChartSeries DataSource="@Data" XName="XValue" YName="YValue"/>
</ChartSeriesCollection>
</SfChart>
@code{
public class ChartData
{
public double XValue { get; set; }
public double YValue { get; set; }
}
public List<ChartData> Data = new List<ChartData>
{
new ChartData { XValue = 10, YValue = 21 },
new ChartData { XValue = 20, YValue = 24 },
new ChartData { XValue = 30, YValue = 36 },
new ChartData { XValue = 40, YValue = 38 },
new ChartData { XValue = 50, YValue = 54 },
new ChartData { XValue = 60, YValue = 57 },
new ChartData { XValue = 70, YValue = 70 },
};
}
```



Refer to our [Blazor Charts](#) feature tour page for its groundbreaking feature representations and also explore our [Blazor Chart Example](#) to know various chart types and how to represent time-dependent data, showing trends at equal intervals.

See Also

- [Data label](#)
- [Tooltip](#)
- [Marker](#)

<!-- markdownlint-disable MD036 -->

DateTime Axis in Blazor Charts Component

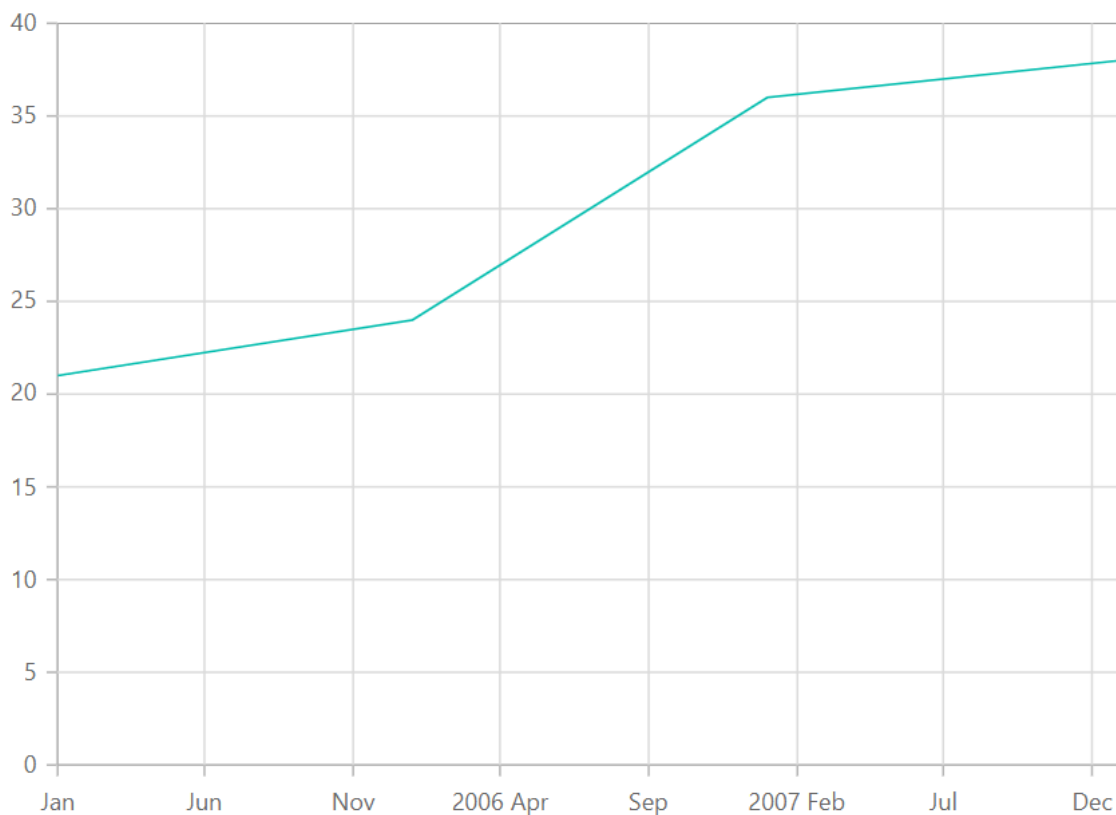
DateTime Axis

The [DateTime](#) axis uses a date time scale and displays date time values as axis labels in the format specified.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart>
  <ChartPrimaryXAxis ValueTypes="Syncfusion.Blazor.Charts.ValueType.DateTime">
  </ChartPrimaryXAxis>
```

```
<ChartSeriesCollection>
<ChartSeries DataSource="@WeatherReports" XName="XValue" YName="YValue"
Type="ChartSeriesType.Line">
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code{
public class ChartData
{
public DateTime XValue { get; set;}
public double YValue {get; set;}
}
public List<ChartData> WeatherReports = new List<ChartData>
{
new ChartData { XValue = new DateTime(2005, 01, 01), YValue = 21 },
new ChartData { XValue = new DateTime(2006, 01, 01), YValue = 24 },
new ChartData { XValue = new DateTime(2007, 01, 01), YValue = 36 },
new ChartData { XValue = new DateTime(2008, 01, 01), YValue = 38 },
};
}
```

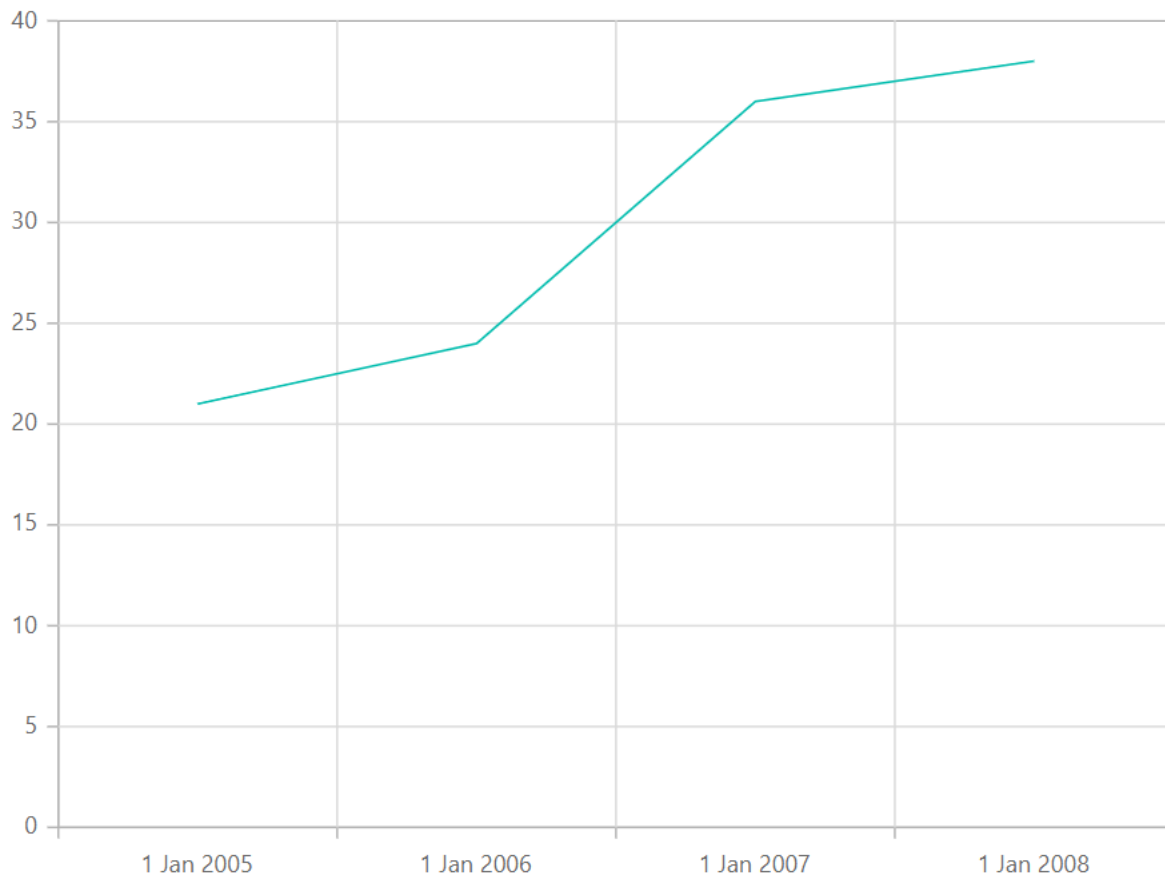


DateTime Category Axis

The [DateTime Category](#) axis is used to display date-time values with non-linear intervals. For example, the business days alone can be represented in a week here.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart>
<ChartPrimaryXAxis Format="d MMM yyyy"
ValueType="Syncfusion.Blazor.Charts.ValueType.DateTimeCategory">
</ChartPrimaryXAxis>
<ChartSeriesCollection>
<ChartSeries DataSource="@WeatherReports" XName="XValue" YName="YValue"
Type="ChartSeriesType.Line">
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code{
public class ChartData
{
public DateTime XValue { get; set; }
public double YValue { get; set; }
}
public List<ChartData> WeatherReports = new List<ChartData>
{
new ChartData { XValue = new DateTime(2005, 01, 01), YValue = 21 },
new ChartData { XValue = new DateTime(2006, 01, 01), YValue = 24 },
new ChartData { XValue = new DateTime(2007, 01, 01), YValue = 36 },
new ChartData { XValue = new DateTime(2008, 01, 01), YValue = 38 },
};
}
```

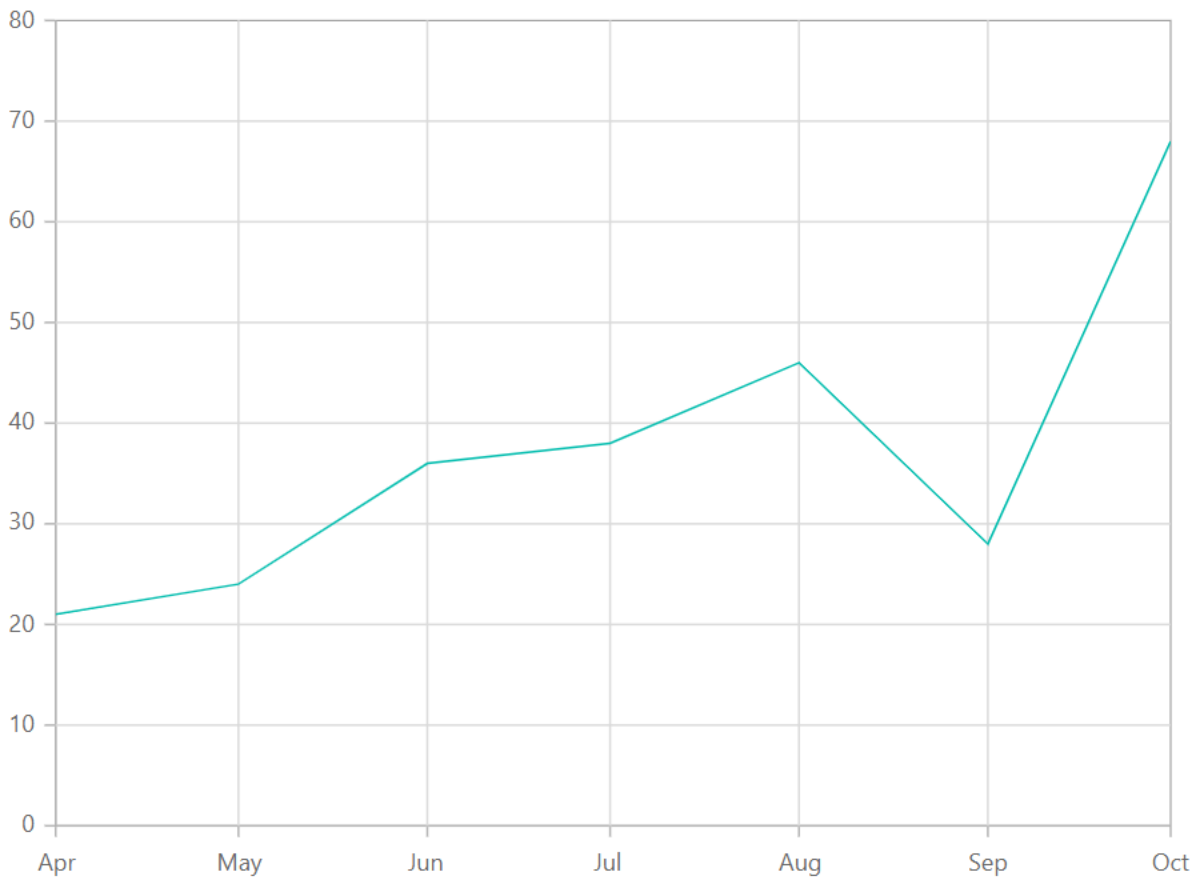


Range

The axis range will be calculated automatically based on the provided data; however, the axis range can also be customized using [Minimum](#), [Maximum](#), and [Interval](#) properties.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart>
<ChartPrimaryXAxis IntervalType="IntervalType.Years"
ValueType="Syncfusion.Blazor.Charts.ValueType.DateTime">
</ChartPrimaryXAxis>
<ChartSeriesCollection>
<ChartSeries DataSource="@WeatherReports" XName="XValue" YName="YValue"
Type="ChartSeriesType.Line">
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code{
public class ChartData
{
public DateTime XValue { get; set; }
public double YValue { get; set; }
}
public List<ChartData> WeatherReports = new List<ChartData>
{
new ChartData { XValue = new DateTime(2016, 4, 1), YValue = 21 },
new ChartData { XValue = new DateTime(2016, 5, 1), YValue = 24 },
new ChartData { XValue = new DateTime(2016, 6, 1), YValue = 36 },
new ChartData { XValue = new DateTime(2016, 7, 1), YValue = 38 },
new ChartData { XValue = new DateTime(2016, 8, 1), YValue = 46 },
new ChartData { XValue = new DateTime(2016, 9, 1), YValue = 28 },
new ChartData { XValue = new DateTime(2016, 10, 1), YValue = 68 }
};
}
```



Interval Customization

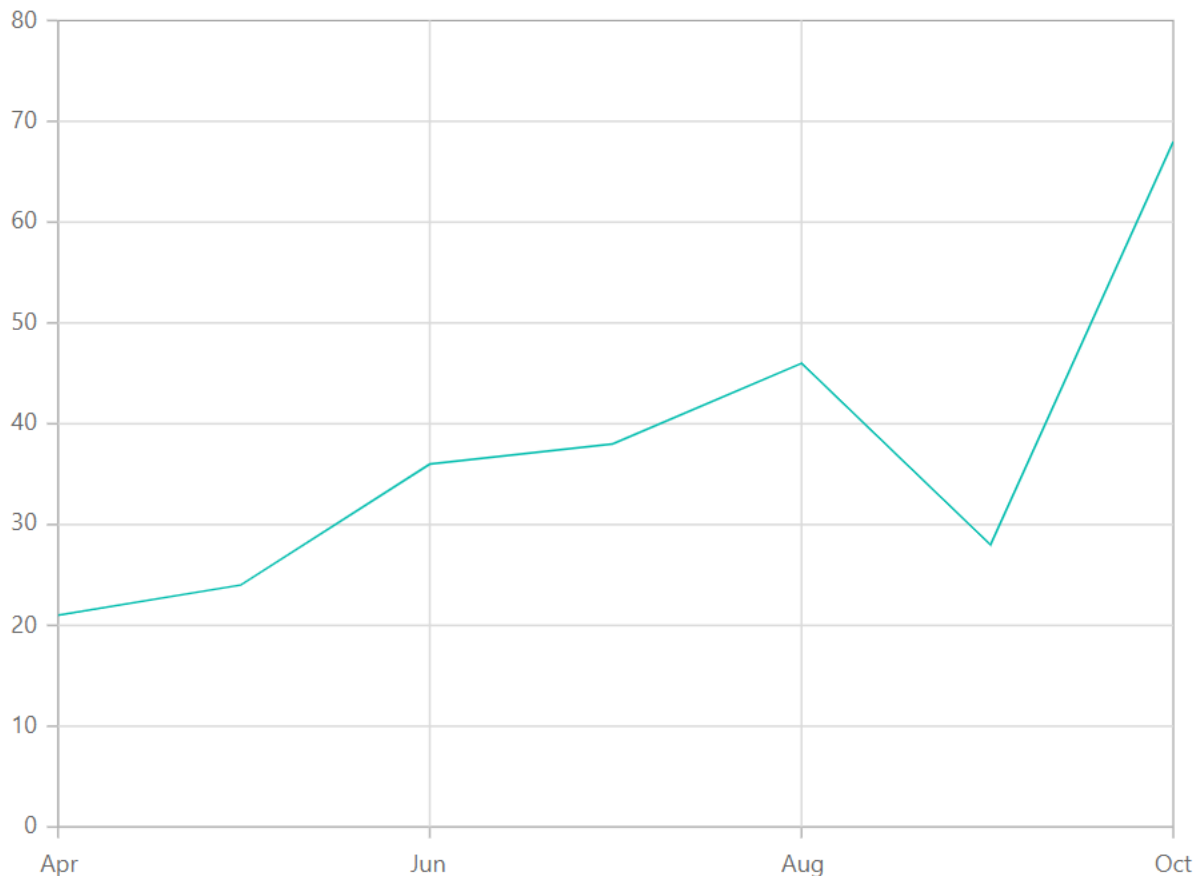
The [Interval](#) and [IntervalType](#) properties of the [Axis](#) can be used to customize date time intervals. When interval is set to **2** and interval type is set to **Years**, it considers 2 years to be the interval. The following interval types are supported by the DateTime axis:

- Auto
- Years
- Months
- Days
- Hours
- Minutes
- Seconds

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart>
<ChartPrimaryXAxis Interval="2" IntervalType="IntervalType.Months"
ValueType="Syncfusion.Blazor.Charts.ValueType.DateTime">
</ChartPrimaryXAxis>
<ChartSeriesCollection>
<ChartSeries DataSource="@WeatherReports" XName="XValue" YName="YValue"
Type="ChartSeriesType.Line">
</ChartSeries>
```

```
</ChartSeriesCollection>
</SfChart>
@code{
public class ChartData
{
public DateTime XValue { get; set; }
public double YValue { get; set; }
}
public List<ChartData> WeatherReports = new List<ChartData>
{
new ChartData { XValue = new DateTime(2016, 4, 1), YValue = 21 },
new ChartData { XValue = new DateTime(2016, 5, 1), YValue = 24 },
new ChartData { XValue = new DateTime(2016, 6, 1), YValue = 36 },
new ChartData { XValue = new DateTime(2016, 7, 1), YValue = 38 },
new ChartData { XValue = new DateTime(2016, 8, 1), YValue = 46 },
new ChartData { XValue = new DateTime(2016, 9, 1), YValue = 28 },
new ChartData { XValue = new DateTime(2016, 10, 1), YValue = 68 }
};
}
```



Applying padding to the Range

The [RangePadding](#) property can be used to apply padding to the minimum and maximum extremes of range. The following types of padding are supported by the DateTime axis:

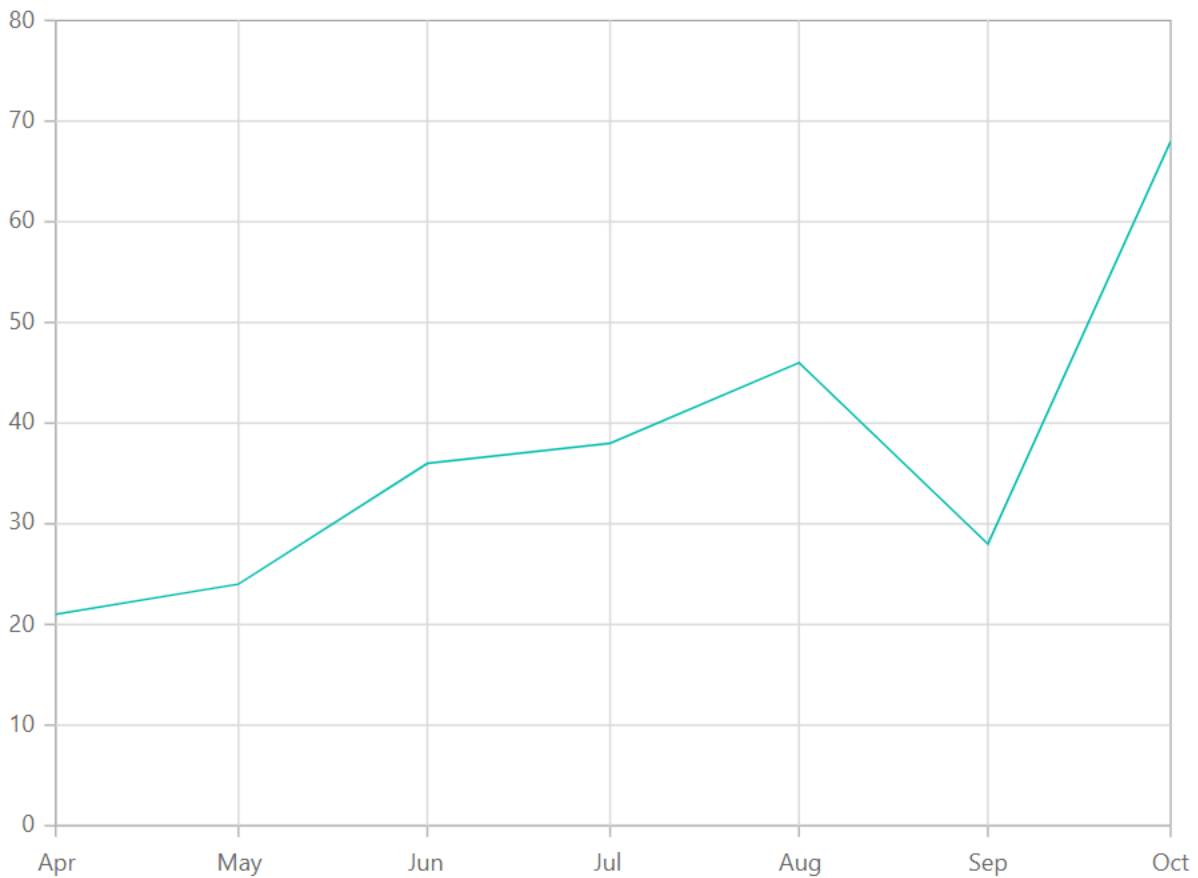
- None
- Round
- Additional

DateTime - None

When the [RangePadding](#) is set to **None**, the minimum and maximum of the axis is based on the data.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart>
<ChartPrimaryXAxis RangePadding="ChartRangePadding.None"
ValueType="Syncfusion.Blazor.Charts.ValueType.DateTime">
</ChartPrimaryXAxis>
<ChartSeriesCollection>
<ChartSeries DataSource="@WeatherReports" XName="XValue" YName="YValue"
Type="ChartSeriesType.Line">
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code{
public class ChartData
{
public DateTime XValue { get; set; }
public double YValue { get; set; }
}
public List<ChartData> WeatherReports = new List<ChartData>
{
new ChartData { XValue = new DateTime(2005, 01, 01), YValue = 21 },
new ChartData { XValue = new DateTime(2006, 01, 01), YValue = 24 },
new ChartData { XValue = new DateTime(2007, 01, 01), YValue = 36 },
new ChartData { XValue = new DateTime(2008, 01, 01), YValue = 38 },
};
}
```

DateTime - Round

When the [RangePadding](#) property is set to **Round**, the minimum and maximum will be rounded to the nearest possible value divisible by interval.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart>
<ChartPrimaryXAxis RangePadding="ChartRangePadding.Round"
ValueType="Syncfusion.Blazor.Charts.ValueType.DateTime">
</ChartPrimaryXAxis>
<ChartSeriesCollection>
<ChartSeries DataSource="@WeatherReports" XName="XValue" YName="YValue"
Type="ChartSeriesType.Line">
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code{
public class ChartData
{
public DateTime XValue { get; set; }
public double YValue { get; set; }
}
public List<ChartData> WeatherReports = new List<ChartData>
{
new ChartData { XValue = new DateTime(2005, 01, 01), YValue = 21 },
```

```
new ChartData { XValue = new DateTime(2006, 01, 01), YValue = 24 },
new ChartData { XValue = new DateTime(2007, 01, 01), YValue = 36 },
new ChartData { XValue = new DateTime(2008, 01, 01), YValue = 38 },
};
}
```

DateTime - Additional

When the [RangePadding](#) property is set to **Additional**, the interval of an axis will be padded to the minimum and maximum of the axis.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart>
<ChartPrimaryXAxis RangePadding="ChartRangePadding.Additional"
ValueType="Syncfusion.Blazor.Charts.ValueType.DateTime"/>
<ChartSeriesCollection>
<ChartSeries DataSource="@WeatherReports" XName="@nameof(ChartData.XValue)"
YName=@nameof(ChartData.YValue) />
</ChartSeriesCollection>
</SfChart>
@code{
public class ChartData
{
public DateTime XValue { get; set; }
public double YValue { get; set; }
}
public List<ChartData> WeatherReports = new List<ChartData>
{
new ChartData { XValue = new DateTime(2005, 01, 01), YValue = 21 },
new ChartData { XValue = new DateTime(2006, 01, 01), YValue = 24 },
new ChartData { XValue = new DateTime(2007, 01, 01), YValue = 36 },
new ChartData { XValue = new DateTime(2008, 01, 01), YValue = 38 },
};
}
```

Label Format

Using the [LabelFormat](#) property on an axis, it is possible to format and parse the date to all globalize formats.

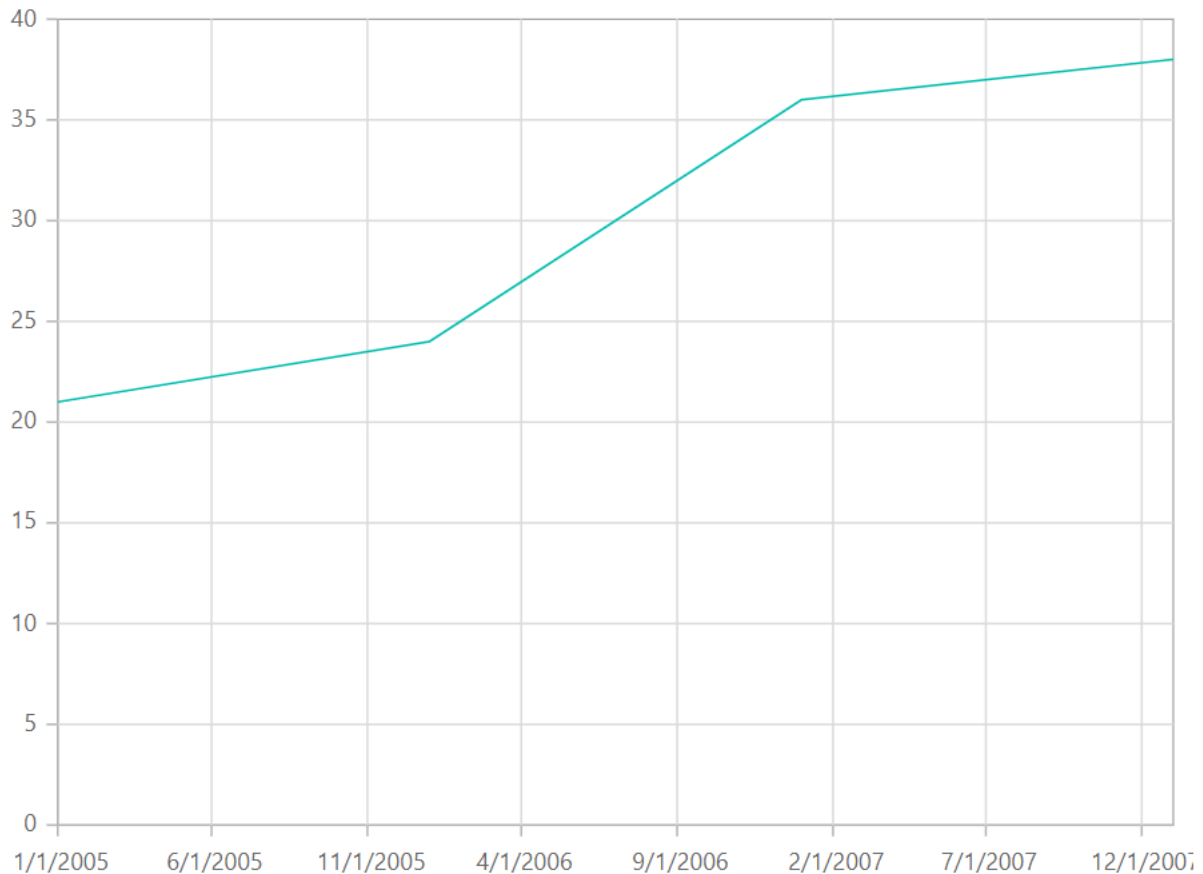
ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart>
<ChartPrimaryXAxis LabelFormat="d"
ValueType="Syncfusion.Blazor.Charts.ValueType.DateTime"/>
<ChartSeriesCollection>
<ChartSeries DataSource="@WeatherReports" XName="XValue" YName="YValue" />
</ChartSeriesCollection>
</SfChart>
@code{
public class ChartData
{
public DateTime XValue { get; set; }
public double YValue { get; set; }
}
```

```

}
public List<ChartData> WeatherReports = new List<ChartData>
{
    new ChartData { XValue = new DateTime(2005, 01, 01), YValue = 21 },
    new ChartData { XValue = new DateTime(2006, 01, 01), YValue = 24 },
    new ChartData { XValue = new DateTime(2007, 01, 01), YValue = 36 },
    new ChartData { XValue = new DateTime(2008, 01, 01), YValue = 38 },
};
}

```



The table below shows the results of applying various popular date and 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.

<!-- markdownlint-disable MD033 -->

Refer to our [Blazor Charts](#) feature tour page for its groundbreaking feature representations and also explore our [Blazor Chart Example](#) to know various chart types and how to represent time-dependent data, showing trends at equal intervals.

See Also

- [Data label](#)
- [Tooltip](#)
- [Marker](#)

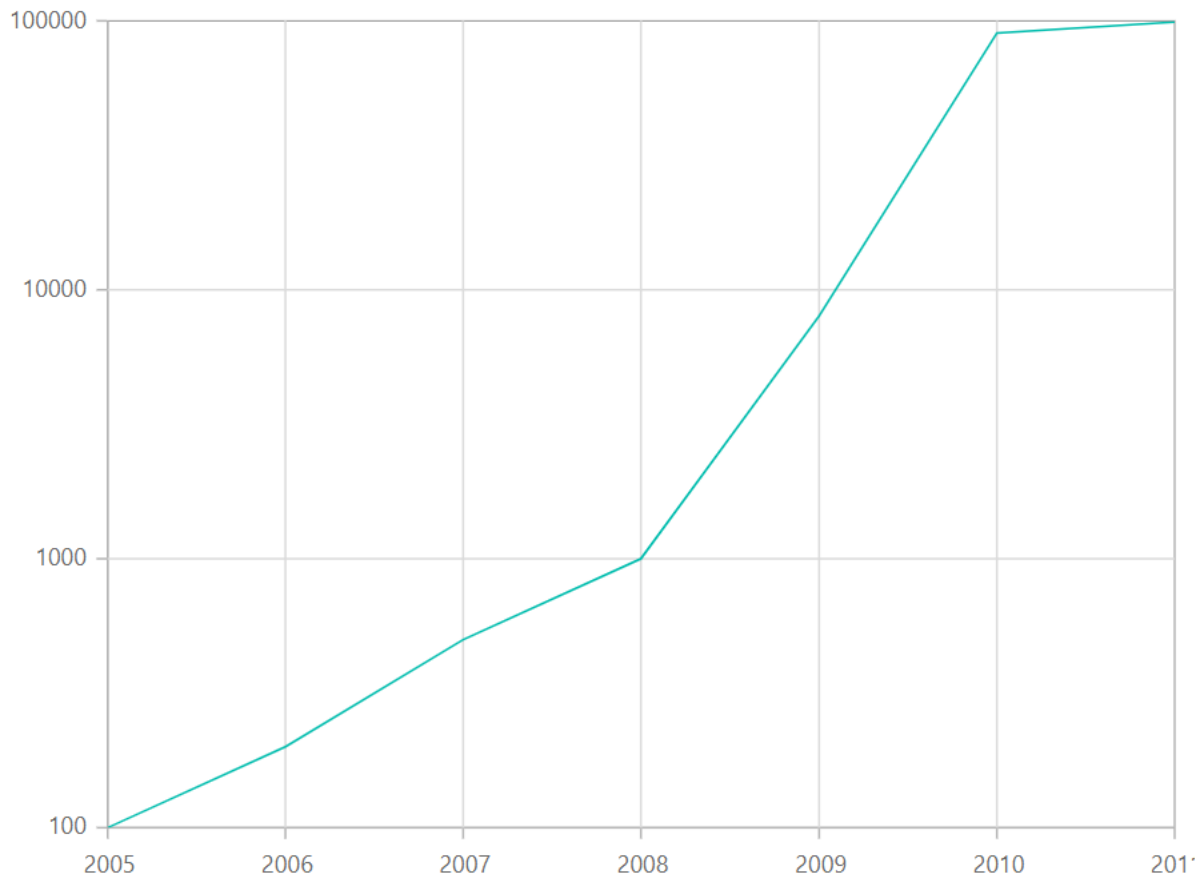
Logarithmic Axis in Blazor Charts Component

<!-- markdownlint-disable MD033 -->

When data contains numeric values in both the lower order of magnitude (eg: 10^{-6}) and the upper order of magnitude (eg: 10^6), a logarithmic axis is highly useful in visualizing it.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart>
<ChartPrimaryXAxis ValueType="Syncfusion.Blazor.Charts.ValueType.DateTime"/>
<ChartPrimaryYAxis
ValueType="Syncfusion.Blazor.Charts.ValueType.Logarithmic"/>
<ChartSeriesCollection>
<ChartSeries DataSource="@Data" XName="XValue" YName="YValue" />
</ChartSeriesCollection>
</SfChart>
@code{
public class ChartData
{
public DateTime XValue { get; set; }
public double YValue { get; set; }
}
public List<ChartData> Data = new List<ChartData>
{
new ChartData { XValue = new DateTime(2005, 01, 01), YValue = 100 },
new ChartData { XValue = new DateTime(2006, 01, 01), YValue = 200 },
new ChartData { XValue = new DateTime(2007, 01, 01), YValue = 500 },
new ChartData { XValue = new DateTime(2008, 01, 01), YValue = 1000 },
new ChartData { XValue = new DateTime(2009, 01, 01), YValue = 8000 },
new ChartData { XValue = new DateTime(2010, 01, 01), YValue = 90000 },
new ChartData { XValue = new DateTime(2011, 01, 01), YValue = 99000 },
};
}
```



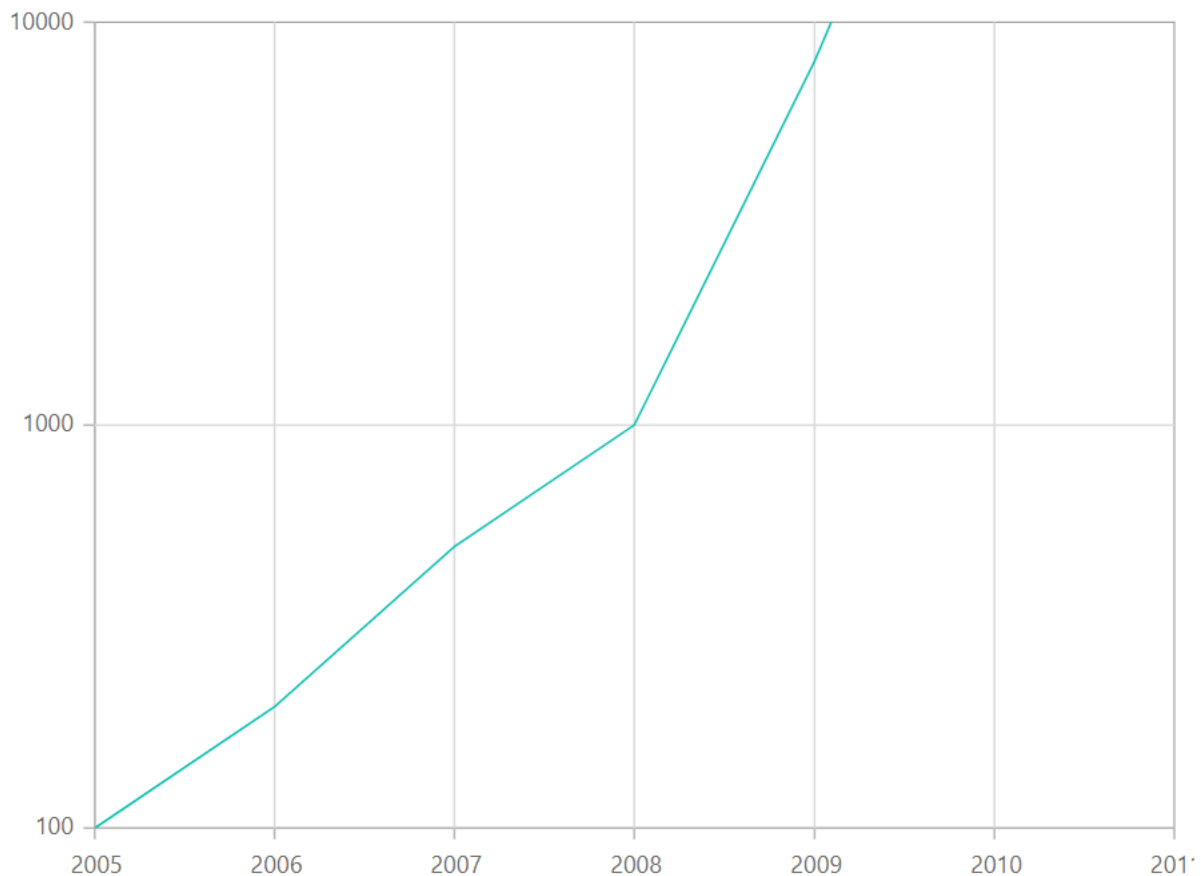
Range

The axis range will be calculated automatically based on the provided data; however, the axis range can also be customized using [Minimum](#), [Maximum](#) and [Interval](#) properties.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart>
<ChartPrimaryXAxis ValueType="Syncfusion.Blazor.Charts.ValueType.DateTime"/>
<ChartPrimaryYAxis Minimum="100" Maximum="10000"
ValueType="Syncfusion.Blazor.Charts.ValueType.Logarithmic"/>
<ChartSeriesCollection>
<ChartSeries DataSource="@Data" XName="XValue" YName="YValue"/>
</ChartSeriesCollection>
</SfChart>
@code{
public class ChartData
{
public DateTime XValue { get; set; }
public double YValue { get; set; }
}
public List<ChartData> Data = new List<ChartData>
{
new ChartData { XValue = new DateTime(2005, 01, 01), YValue = 100 },
new ChartData { XValue = new DateTime(2006, 01, 01), YValue = 200 },
new ChartData { XValue = new DateTime(2007, 01, 01), YValue = 500 },
new ChartData { XValue = new DateTime(2008, 01, 01), YValue = 1000 },
}
```

```
new ChartData { XValue = new DateTime(2009, 01, 01), YValue = 8000 },
new ChartData { XValue = new DateTime(2010, 01, 01), YValue = 90000 },
new ChartData { XValue = new DateTime(2011, 01, 01), YValue = 99000 },
};
}
```



Logarithmic Base

Logarithmic base can be customized using the [LogBase](#) property of the axis. When the LogBase is 5, the axis values are 5^{-2} , 5^{-1} , 5^0 , 5^1 , 5^2 and so on.

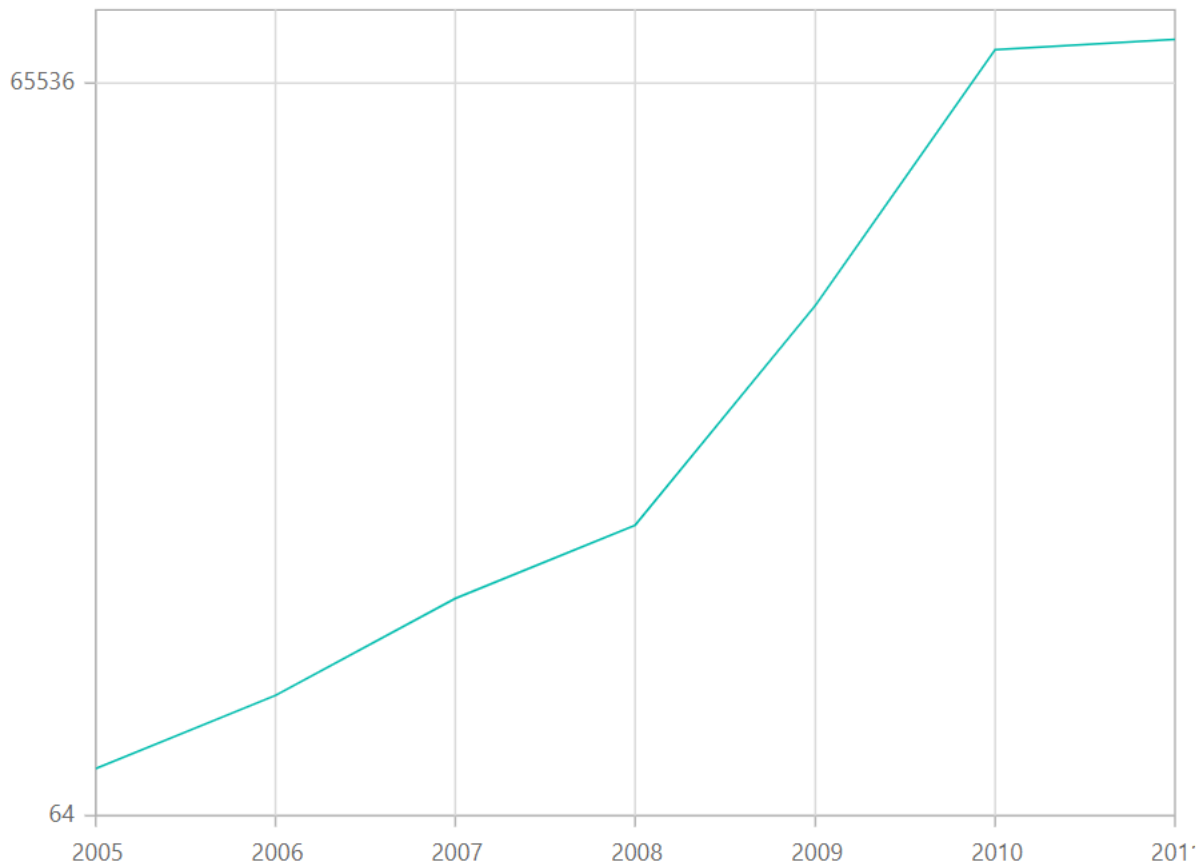
ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart>
<ChartPrimaryXAxis ValueType="Syncfusion.Blazor.Charts.ValueType.DateTime"/>
<ChartPrimaryYAxis LogBase="2"
ValueType="Syncfusion.Blazor.Charts.ValueType.Logarithmic"/>
<ChartSeriesCollection>
<ChartSeries DataSource="@Data" XName="XValue" YName="YValue" />
</ChartSeriesCollection>
</SfChart>
@code{
public class ChartData
{
public DateTime XValue { get; set; }
}
```

```

public double YValue { get; set; }
}
public List<ChartData> Data = new List<ChartData>
{
    new ChartData { XValue = new DateTime(2005, 01, 01), YValue = 100 },
    new ChartData { XValue = new DateTime(2006, 01, 01), YValue = 200 },
    new ChartData { XValue = new DateTime(2007, 01, 01), YValue = 500 },
    new ChartData { XValue = new DateTime(2008, 01, 01), YValue = 1000 },
    new ChartData { XValue = new DateTime(2009, 01, 01), YValue = 8000 },
    new ChartData { XValue = new DateTime(2010, 01, 01), YValue = 90000 },
    new ChartData { XValue = new DateTime(2011, 01, 01), YValue = 99000 },
};
}

```



Logarithmic Interval

The interval can be customized using the [Interval](#) property of the logarithmic axis. When the logarithmic base is **10** and logarithmic interval is **2**, then the axis labels are placed at an interval of **10²**. The default value of the interval is **1**.

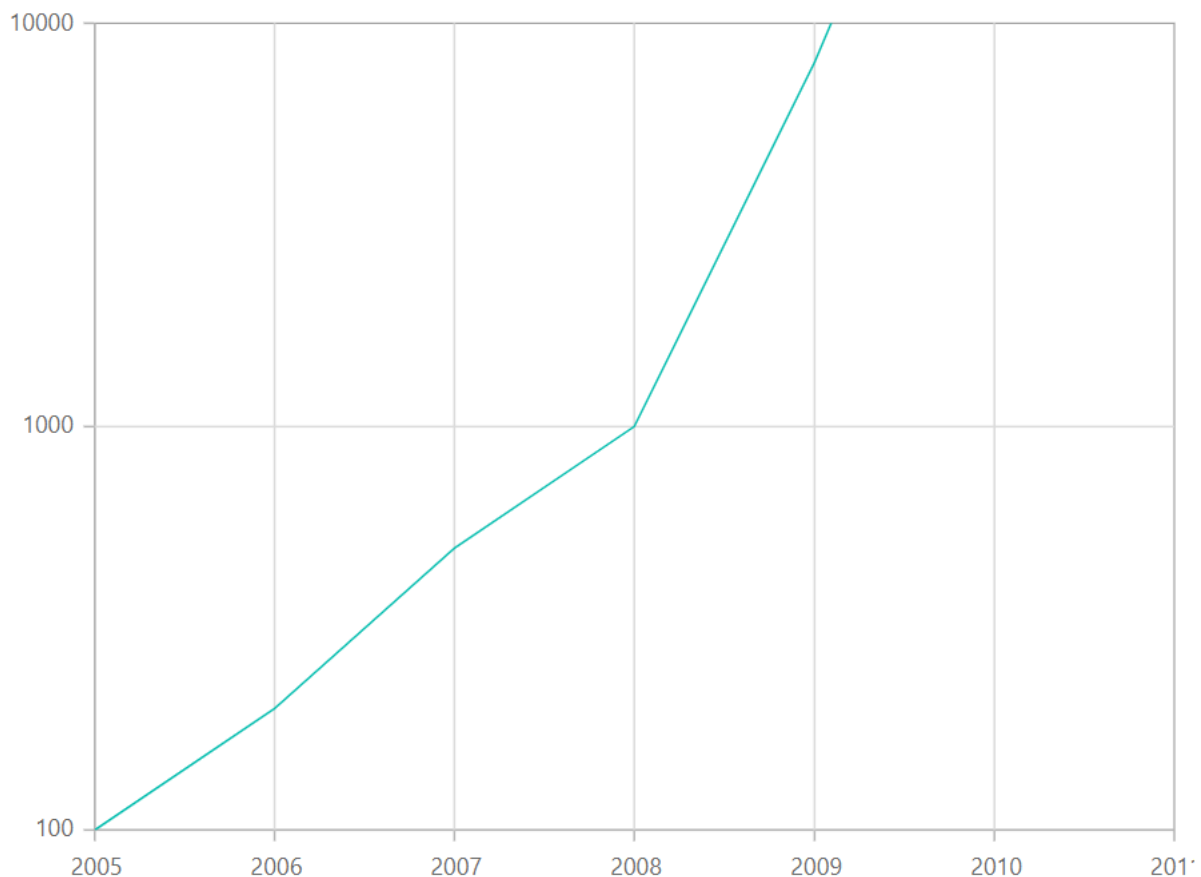
ASPX-CS

```

@using Syncfusion.Blazor.Charts
<SfChart>
<ChartPrimaryXAxis Value="Syncfusion.Blazor.Charts.ValueType.DateTime"/>
<ChartPrimaryYAxis Interval="2" LogBase="2"
Value="Syncfusion.Blazor.Charts.ValueType.Logarithmic"/>

```

```
<ChartSeriesCollection>
<ChartSeries DataSource="@Data" XName="XValue" YName="YValue" />
</ChartSeriesCollection>
</SfChart>
@code{
public class ChartData
{
public DateTime XValue { get; set; }
public double YValue { get; set; }
}
public List<ChartData> Data = new List<ChartData>
{
new ChartData { XValue = new DateTime(2005, 01, 01), YValue = 100 },
new ChartData { XValue = new DateTime(2006, 01, 01), YValue = 200 },
new ChartData { XValue = new DateTime(2007, 01, 01), YValue = 500 },
new ChartData { XValue = new DateTime(2008, 01, 01), YValue = 1000 },
new ChartData { XValue = new DateTime(2009, 01, 01), YValue = 8000 },
new ChartData { XValue = new DateTime(2010, 01, 01), YValue = 90000 },
new ChartData { XValue = new DateTime(2011, 01, 01), YValue = 99000 },
};
}
```

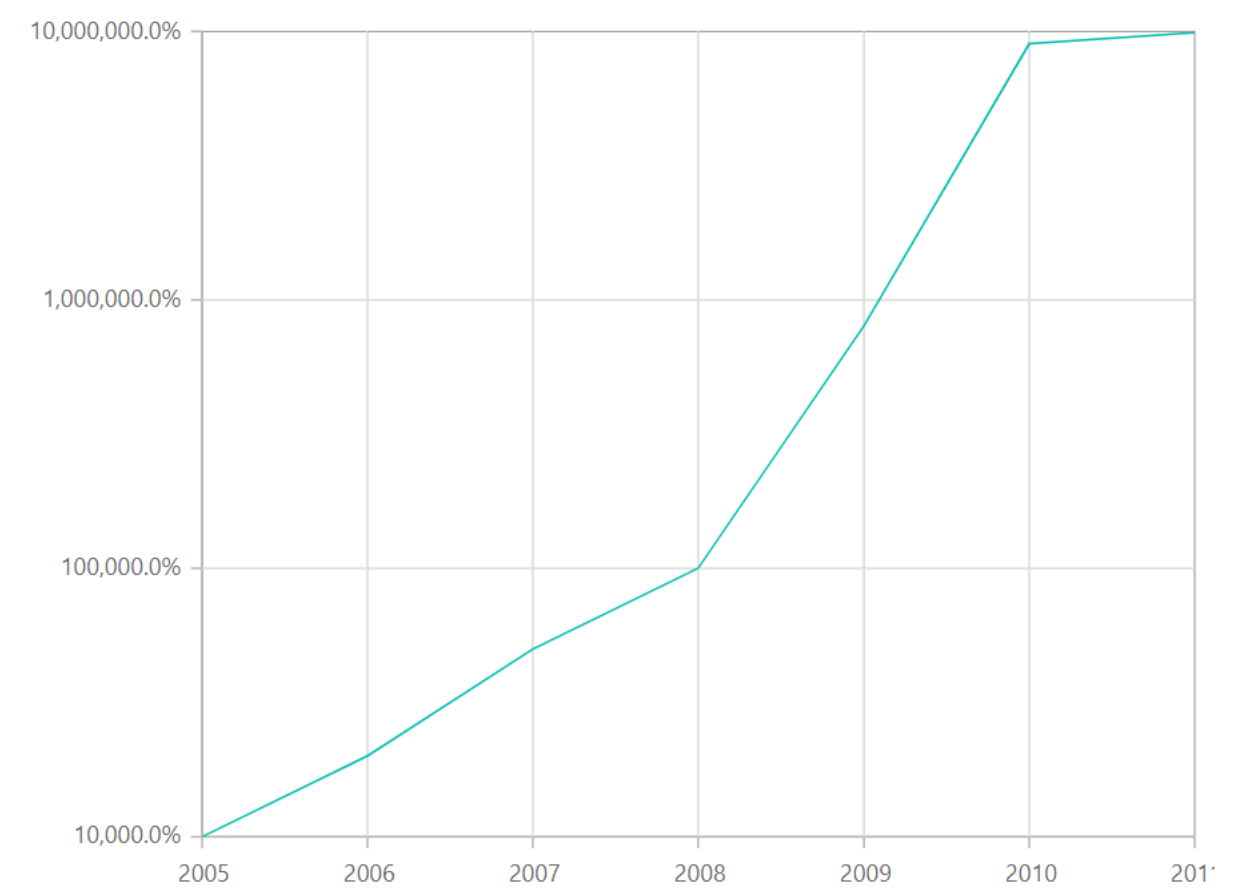


Label Format

Using the [LabelFormat](#) property on an axis, it is possible to format the logarithmic labels to all globalize formats.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart>
<ChartPrimaryXAxis ValueType="Syncfusion.Blazor.Charts.ValueType.DateTime"/>
<ChartPrimaryYAxis LabelFormat="P1"
ValueType="Syncfusion.Blazor.Charts.ValueType.Logarithmic"/>
<ChartSeriesCollection>
<ChartSeries DataSource="@Data" XName="XValue" YName="YValue" />
</ChartSeriesCollection>
</SfChart>
@code{
public class ChartData
{
public DateTime XValue { get; set; }
public double YValue { get; set; }
}
public List<ChartData> Data = new List<ChartData>
{
new ChartData { XValue = new DateTime(2005, 01, 01), YValue = 100 },
new ChartData { XValue = new DateTime(2006, 01, 01), YValue = 200 },
new ChartData { XValue = new DateTime(2007, 01, 01), YValue = 500 },
new ChartData { XValue = new DateTime(2008, 01, 01), YValue = 1000 },
new ChartData { XValue = new DateTime(2009, 01, 01), YValue = 8000 },
new ChartData { XValue = new DateTime(2010, 01, 01), YValue = 90000 },
new ChartData { XValue = new DateTime(2011, 01, 01), YValue = 99000 },
};
}
```



The table below shows the results of applying some commonly used label formats to logarithmic values.
<!-- markdownlint-disable MD033 -->

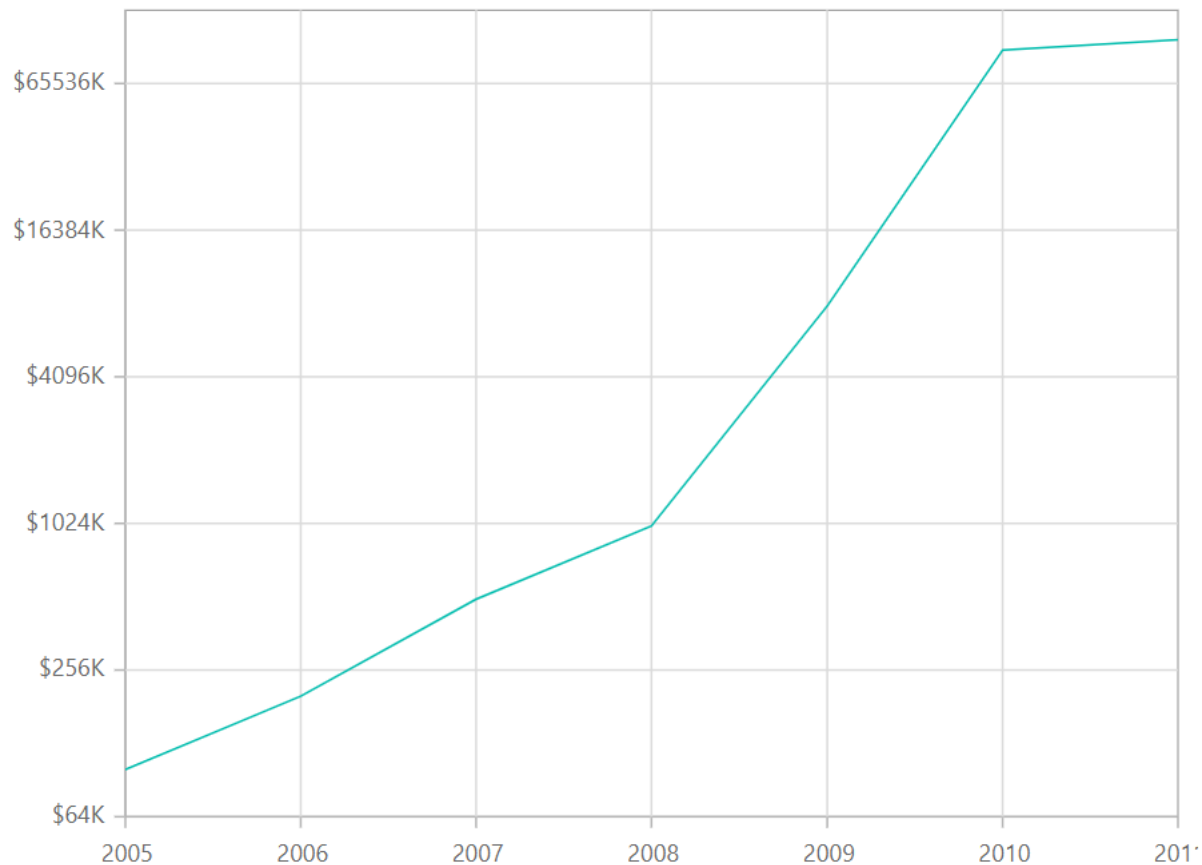
Label Value	Label Format property value	Result	Description
1000	n1	1000.0	The value is rounded to 1 decimal place.
1000	n2	1000.00	The value is rounded to 2 decimal place.
1000	n3	1000.000	The value is rounded to 3 decimal place.
0.01	p1	1.0%	The value is converted to percentage with 1 decimal place.
0.01	p2	1.00%	The value is converted to percentage with 2 decimal place.
0.01	p3	1.000%	The value 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.

Custom Label Format

Axis also supports custom label format using placeholders such as {value}K, where the value represents the axis label, for example, 200K.

ASPX-CS

```
<SfChart>
<ChartPrimaryXAxis ValueType="Syncfusion.Blazor.Charts.ValueType.DateTime"/>
<ChartPrimaryYAxis
ValueType="Syncfusion.Blazor.Charts.ValueType.Logarithmic"
LabelFormat="{value}K" RangePadding="ChartRangePadding.Auto"/>
<ChartSeriesCollection>
<ChartSeries DataSource="@Data" XName="XValue" YName="YValue" />
</ChartSeriesCollection>
</SfChart>
@code{
public class ChartData
{
public DateTime XValue { get; set; }
public double YValue { get; set; }
}
public List<ChartData> Data = new List<ChartData>
{
new ChartData { XValue = new DateTime(2005, 01, 01), YValue = 100 },
new ChartData { XValue = new DateTime(2006, 01, 01), YValue = 200 },
new ChartData { XValue = new DateTime(2007, 01, 01), YValue = 500 },
new ChartData { XValue = new DateTime(2008, 01, 01), YValue = 1000 },
new ChartData { XValue = new DateTime(2009, 01, 01), YValue = 8000 },
new ChartData { XValue = new DateTime(2010, 01, 01), YValue = 90000 },
new ChartData { XValue = new DateTime(2011, 01, 01), YValue = 99000 },
};
}
```



Refer to our [Blazor Charts](#) feature tour page for its groundbreaking feature representations and also explore our [Blazor Chart Example](#) to know various chart types and how to represent time-dependent data, showing trends at equal intervals.

See Also

- [Data label](#)
- [Tooltip](#)
- [Marker](#)

Axis Labels in Blazor Charts Component

Smart Axis Labels

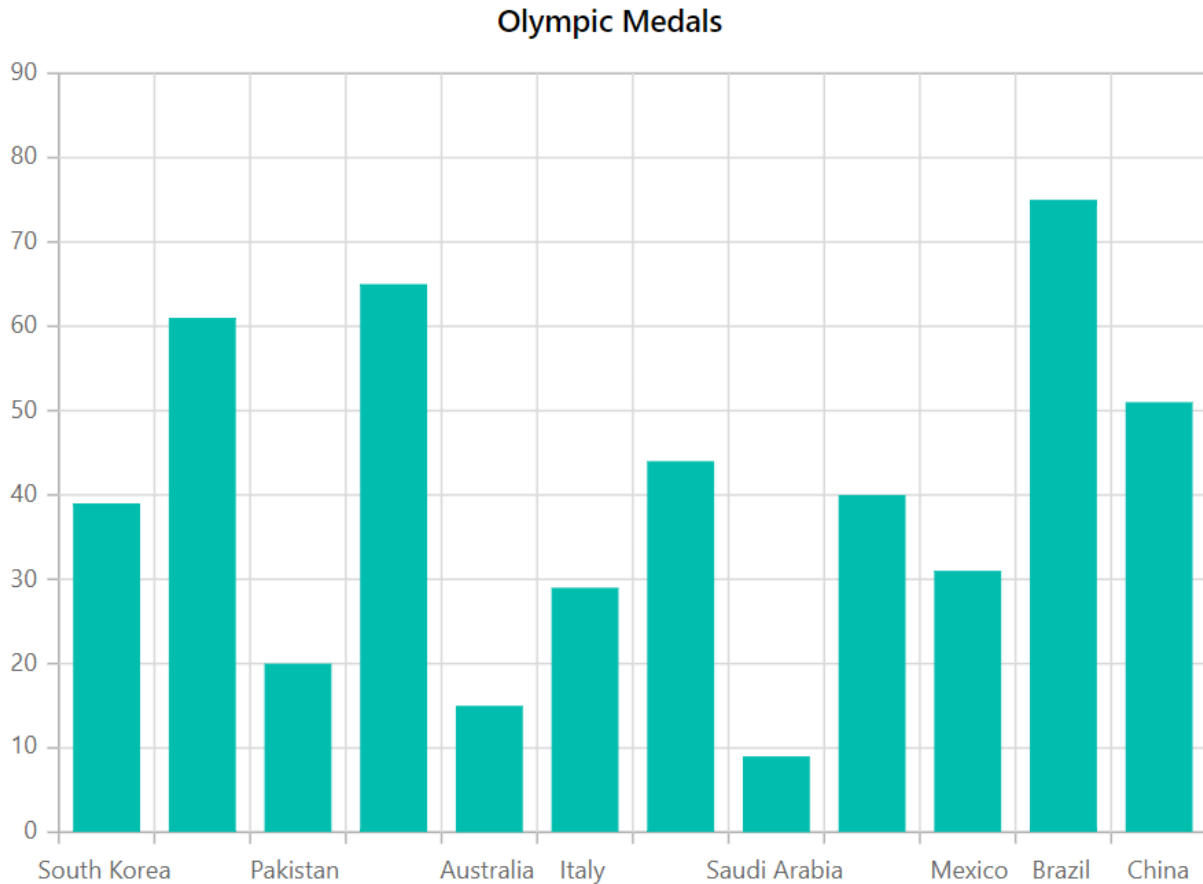
When the axis labels overlap, the [LabelIntersectAction](#) property in the axis can be used to intelligently arrange them.

Case 1: When [LabelIntersectAction](#) is set to **Hide**.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart Title="Olympic Medals" >
  <ChartPrimaryXAxis ValueType="Syncfusion.Blazor.Charts.ValueType.Category"
    LabelIntersectAction="LabelIntersectAction.Hide" />
  <ChartSeriesCollection>
```

```
<ChartSeries DataSource="@MedalDetails" XName="X" YName="Y"
Type="ChartSeriesType.Column" />
</ChartSeriesCollection>
</SfChart>
@code{
public class ChartData
{
public string X { get; set; }
public double Y { get; set; }
}
public List<ChartData> MedalDetails = new List<ChartData>
{
new ChartData { X= "South Korea", Y= 39 },
new ChartData { X= "India", Y= 61 },
new ChartData { X= "Pakistan", Y= 20 },
new ChartData { X= "Germany", Y= 65 },
new ChartData { X= "Australia", Y= 15 },
new ChartData { X= "Italy", Y= 29 },
new ChartData { X= "United Kingdom", Y= 44 },
new ChartData { X= "Saudi Arabia", Y= 9 },
new ChartData { X= "Russia", Y= 40 },
new ChartData { X= "Mexico", Y= 31 },
new ChartData { X= "Brazil", Y= 75 },
new ChartData { X= "China", Y= 51 }
};
}
```



Case 2: When [LabelIntersectAction](#) is set to **Rotate45**.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart Title="Olympic Medals">
  <ChartPrimaryXAxis ValueType="Syncfusion.Blazor.Charts.ValueType.Category"
    LabelIntersectAction="LabelIntersectAction.Rotate45" />
  <ChartSeriesCollection>
    <ChartSeries DataSource="@MedalDetails" XName="X" YName="Y"
      Type="ChartSeriesType.Column" />
  </ChartSeriesCollection>
</SfChart>

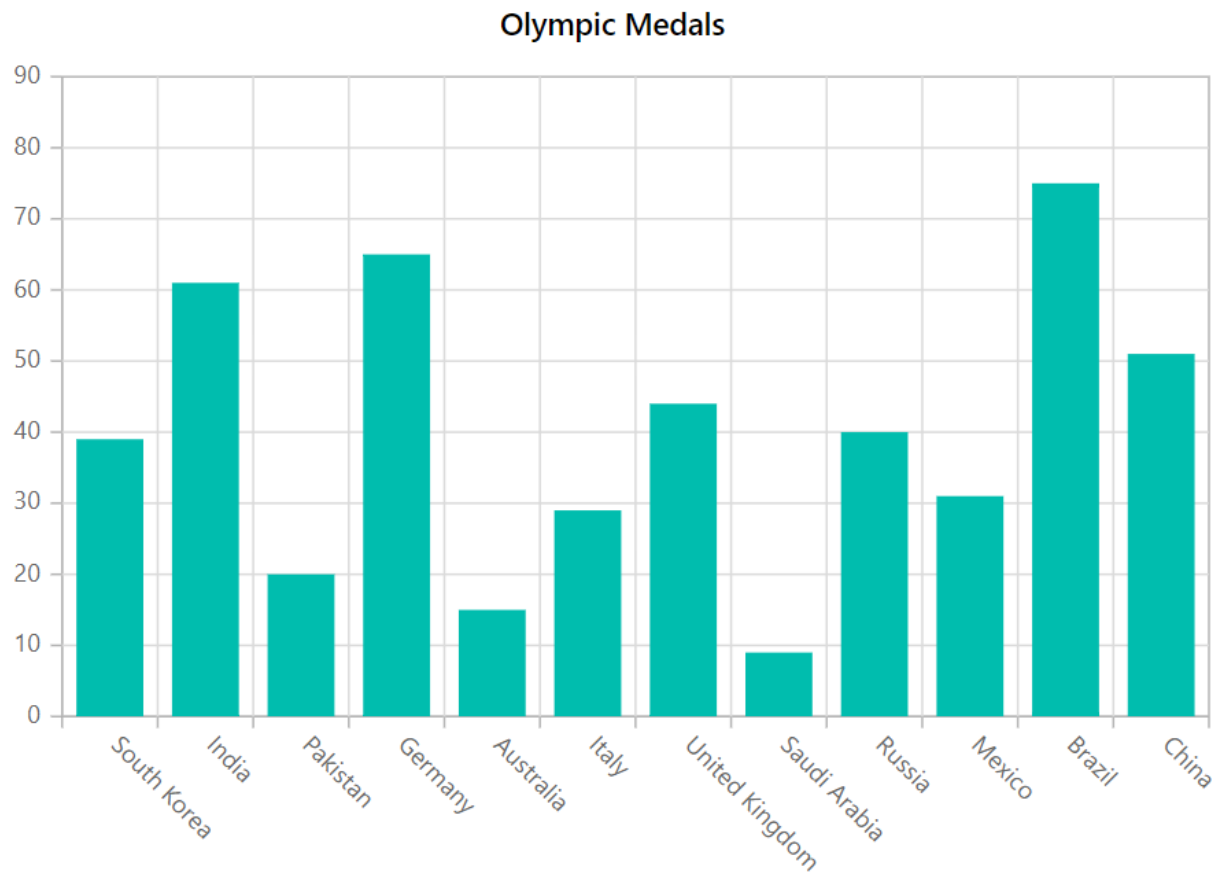
@code{
public class ChartData
{
    public string X { get; set; }
    public double Y { get; set; }
}

public List<ChartData> MedalDetails = new List<ChartData>
{
    new ChartData { X= "South Korea", Y= 39 },
    new ChartData { X= "India", Y= 61 },
    new ChartData { X= "Pakistan", Y= 20 },
    new ChartData { X= "Germany", Y= 65 },
    new ChartData { X= "Australia", Y= 15 },
    new ChartData { X= "Italy", Y= 29 },
}
```

```

new ChartData { X= "United Kingdom", Y= 44 },
new ChartData { X= "Saudi Arabia", Y= 9 },
new ChartData { X= "Russia", Y= 40 },
new ChartData { X= "Mexico", Y= 31 },
new ChartData { X= "Brazil", Y= 75 },
new ChartData { X= "China", Y= 51 }
};
}

```



Case 3: When [LabelIntersectAction](#) is set to **Rotate90**.

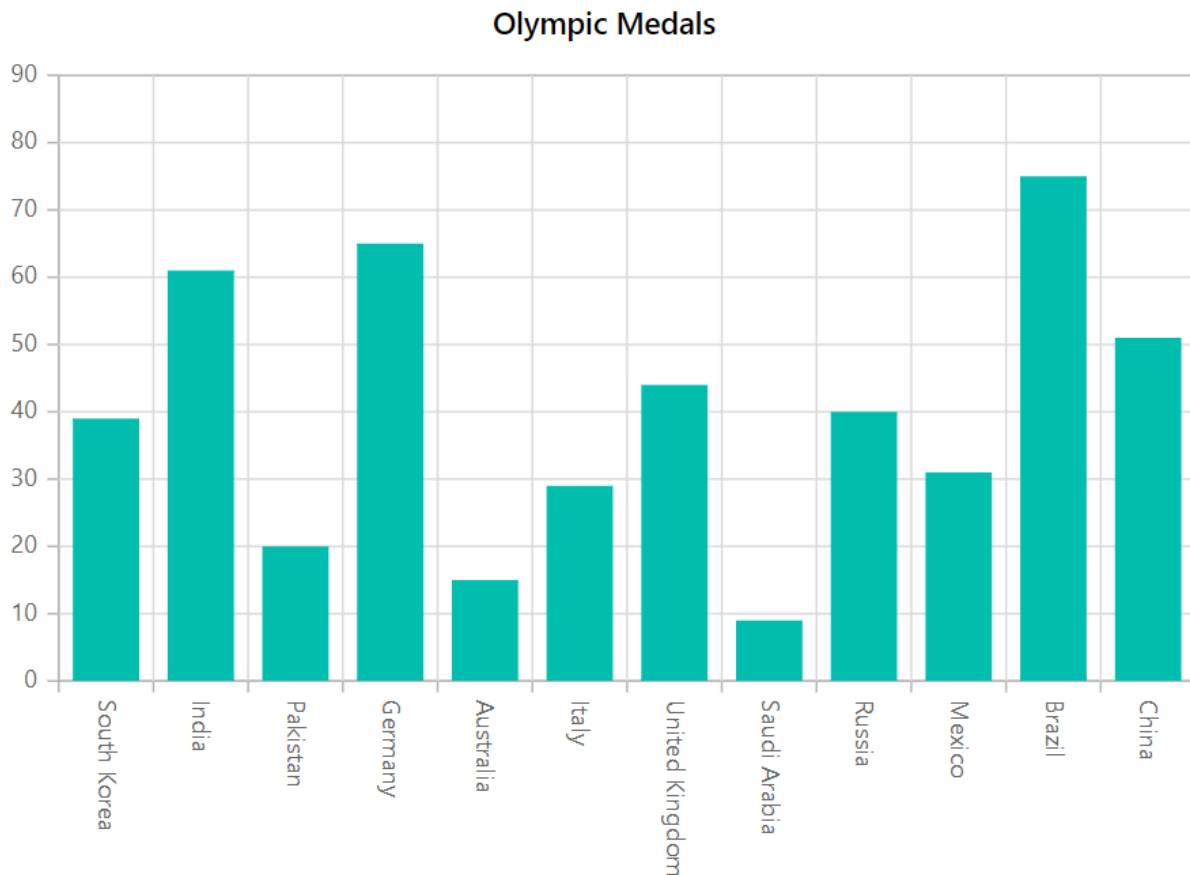
ASPX-CS

```

@using Syncfusion.Blazor.Charts
<SfChart Title="Olympic Medals">
  <ChartPrimaryXAxis ValueTypes="Syncfusion.Blazor.Charts.ValueType.Category"
    LabelIntersectAction="LabelIntersectAction.Rotate90" />
  <ChartSeriesCollection>
    <ChartSeries DataSource="@MedalDetails" XName="X" YName="Y"
      Type="ChartSeriesType.Column" />
  </ChartSeriesCollection>
</SfChart>
@code{
public class ChartData
{
public string X { get; set; }

```

```
public double Y { get; set; }  
}  
public List<ChartData> MedalDetails = new List<ChartData>  
{  
    new ChartData { X= "South Korea", Y= 39 },  
    new ChartData { X= "India", Y= 61 },  
    new ChartData { X= "Pakistan", Y= 20 },  
    new ChartData { X= "Germany", Y= 65 },  
    new ChartData { X= "Australia", Y= 15 },  
    new ChartData { X= "Italy", Y= 29 },  
    new ChartData { X= "United Kingdom", Y= 44 },  
    new ChartData { X= "Saudi Arabia", Y= 9 },  
    new ChartData { X= "Russia", Y= 40 },  
    new ChartData { X= "Mexico", Y= 31 },  
    new ChartData { X= "Brazil", Y= 75 },  
    new ChartData { X= "China", Y= 51 }  
};  
}
```



Axis Labels Positioning

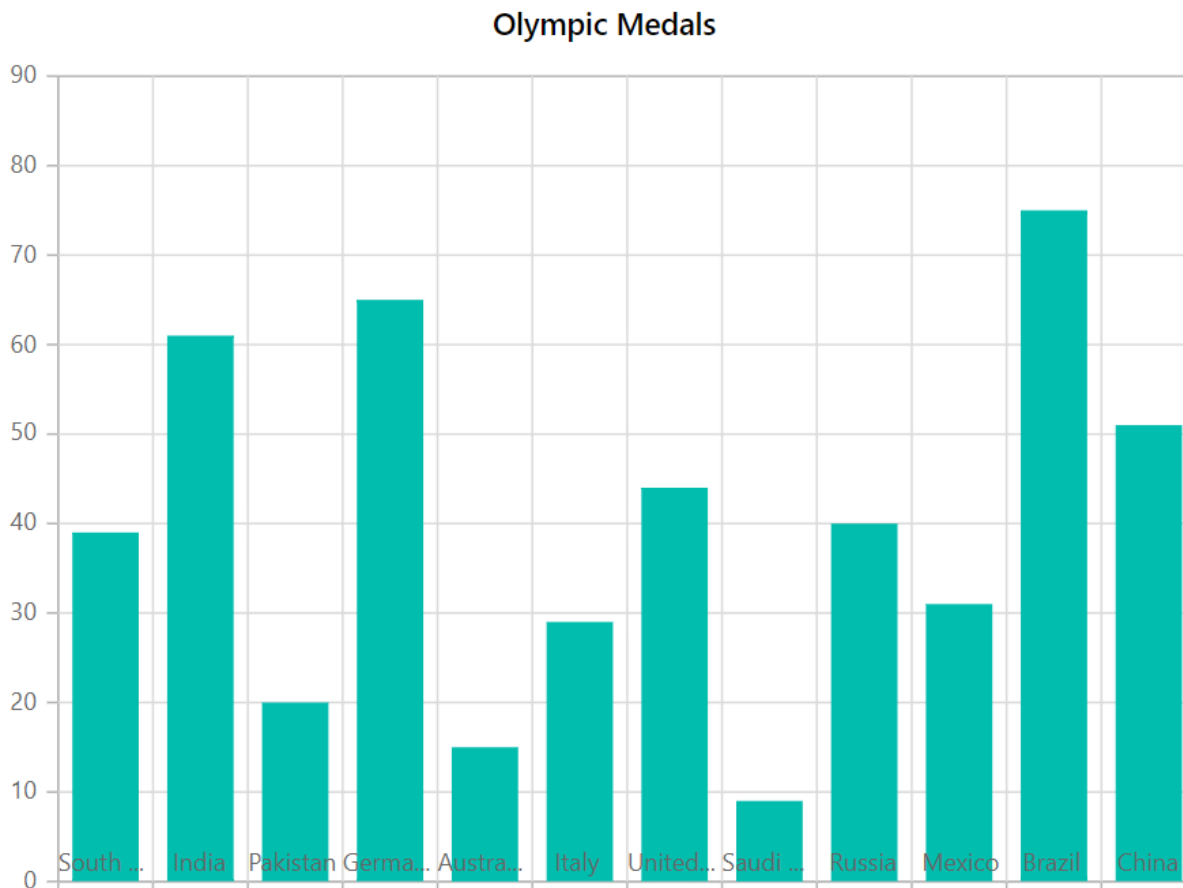
The axis labels can be put **Outside** of the axis line by default, however the [LabelPosition](#) property can also be used to position them **Inside** the axis line.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
```



```
<SfChart Title="Olympic Medals" >
<ChartPrimaryXAxis ValueType="Syncfusion.Blazor.Charts.ValueType.Category"
LabelPosition="AxisPosition.Inside"/>
<ChartSeriesCollection>
<ChartSeries DataSource="@MedalDetails" XName="X" YName="Y"
Type="ChartSeriesType.Column"/>
</ChartSeriesCollection>
</SfChart>
@code{
public class ChartData
{
public string X { get; set; }
public double Y { get; set; }
}
public List<ChartData> MedalDetails = new List<ChartData>
{
new ChartData { X= "South Korea", Y= 39 },
new ChartData { X= "India", Y= 61 },
new ChartData { X= "Pakistan", Y= 20 },
new ChartData { X= "Germany", Y= 65 },
new ChartData { X= "Australia", Y= 15 },
new ChartData { X= "Italy", Y= 29 },
new ChartData { X= "United Kingdom", Y= 44 },
new ChartData { X= "Saudi Arabia", Y= 9 },
new ChartData { X= "Russia", Y= 40 },
new ChartData { X= "Mexico", Y= 31 },
new ChartData { X= "Brazil", Y= 75 },
new ChartData { X= "China", Y= 51 }
};
}
```



Multilevel Labels

The [MultiLevelLabels](#) property allows to add any number of layers of labels to the axis. The following properties can be used to configure this property.

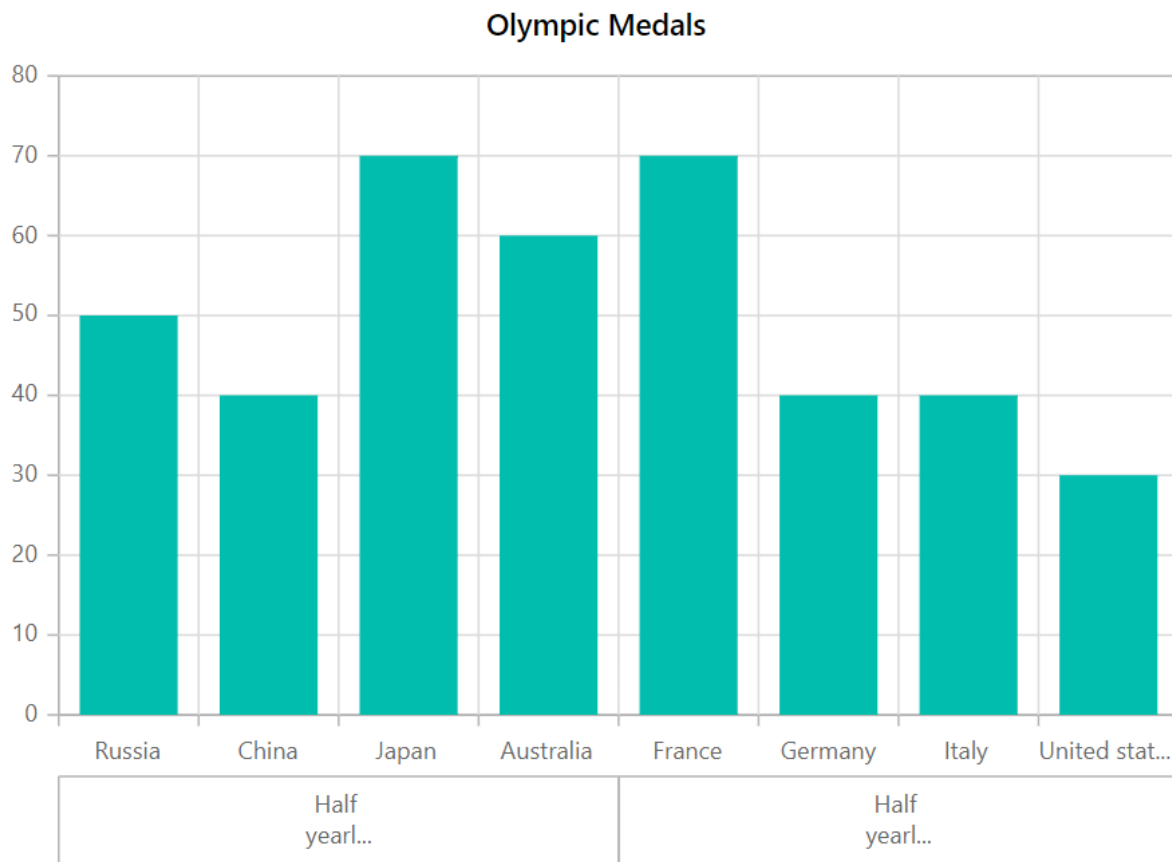
Categories

The [Start](#), [End](#), [Text](#), and [MaximumTextWidth](#) of multilevel labels can be customized using the [ChartCategories](#) which accepts the collections of [ChartCategory](#).

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart Title="Olympic Medals">
  <ChartPrimaryXAxis ValueTypes="Syncfusion.Blazor.Charts.ValueType.Category">
    <ChartMultiLevelLabels>
      <ChartMultiLevelLabel>
        <ChartCategories>
          <ChartCategory Start="-0.5" End="3.5" Text="Half yearly 1"
            MaximumTextWidth=50></ChartCategory>
          <ChartCategory Start="3.5" End="7.5" Text="Half yearly 2"
            MaximumTextWidth=50></ChartCategory>
        </ChartCategories>
      </ChartMultiLevelLabel>
    </ChartMultiLevelLabels>
  </ChartPrimaryXAxis>
  <ChartSeriesCollection>
```

```
<ChartSeries DataSource="@MedalDetails" XName="X" YName="Y"
Type="ChartSeriesType.Column">
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code{
public int start = 0, end = 30;
public class ChartData
{
public string X { get; set; }
public double Y { get; set; }
}
public List<ChartData> MedalDetails = new List<ChartData>
{
new ChartData { X= "Russia", Y= 50 },
new ChartData { X= "China", Y= 40 },
new ChartData { X= "Japan", Y= 70 },
new ChartData { X= "Australia", Y= 60 },
new ChartData { X= "France", Y= 70 },
new ChartData { X= "Germany", Y= 40 },
new ChartData { X= "Italy", Y= 40 },
new ChartData { X= "United states", Y= 30 },
};
}
```



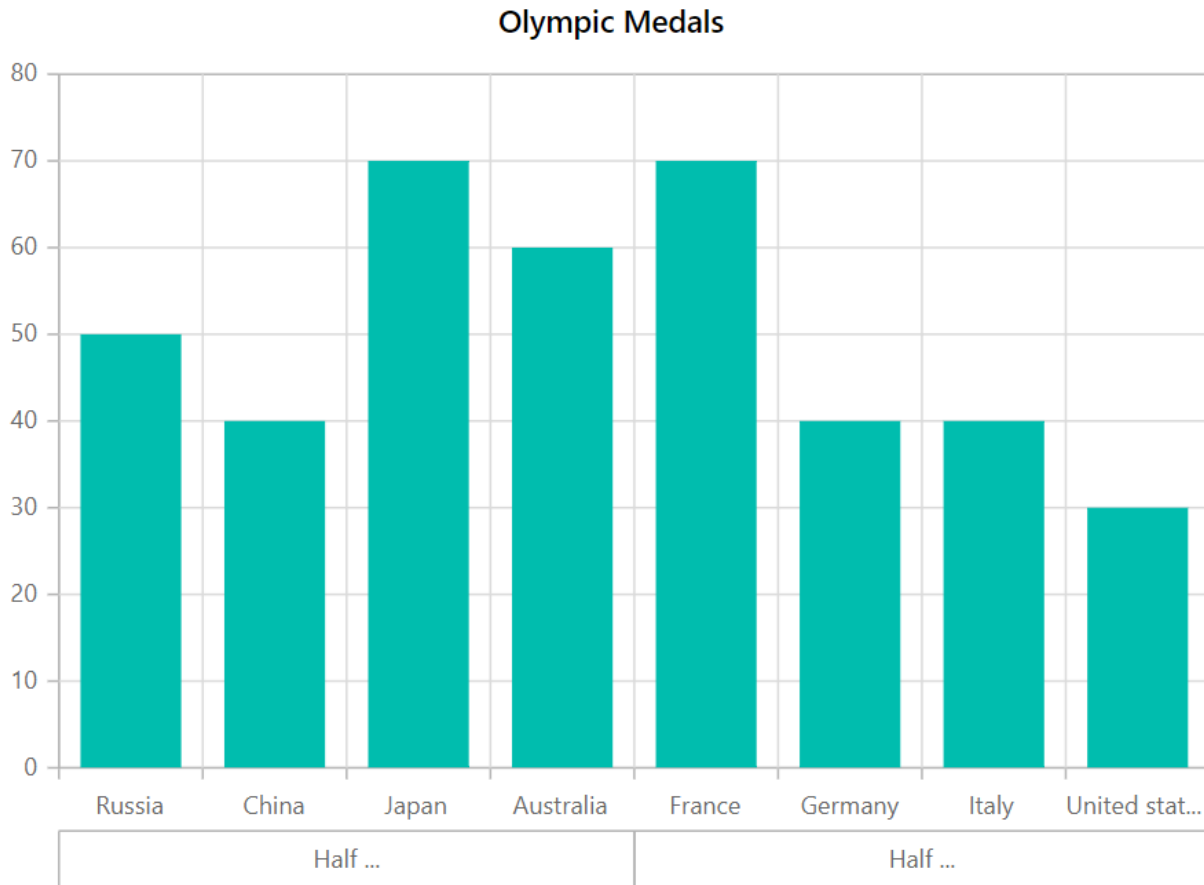
Overflow

Using the [Overflow](#) property, one can [Trim](#) or [Wrap](#) the multilevel labels.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart Title="Olympic Medals">
  <ChartPrimaryXAxis ValueType="Syncfusion.Blazor.Charts.ValueType.Category">
    <ChartMultiLevelLabels>
      <ChartMultiLevelLabel Overflow="TextOverflow.Trim">
        <ChartCategories>
          <ChartCategory Start="-0.5" End="3.5" Text="Half yearly 1"
            MaximumTextWidth=50></ChartCategory>
          <ChartCategory Start="3.5" End="7.5" Text="Half yearly 2"
            MaximumTextWidth=50></ChartCategory>
        </ChartCategories>
      </ChartMultiLevelLabel>
    </ChartMultiLevelLabels>
  </ChartPrimaryXAxis>
  <ChartSeriesCollection>
    <ChartSeries DataSource="@MedalDetails" XName="X" YName="Y"
      Type="ChartSeriesType.Column">
    </ChartSeries>
  </ChartSeriesCollection>
</SfChart>

@code{
public int start = 0, end = 30;
public class ChartData
{
public string X { get; set; }
public double Y { get; set; }
}
public List<ChartData> MedalDetails = new List<ChartData>
{
new ChartData { X= "Russia", Y= 50 },
new ChartData { X= "China", Y= 40 },
new ChartData { X= "Japan", Y= 70 },
new ChartData { X= "Australia", Y= 60 },
new ChartData { X= "France", Y= 70 },
new ChartData { X= "Germany", Y= 40 },
new ChartData { X= "Italy", Y= 40 },
new ChartData { X= "United states", Y= 30 },
};
}
```



Alignment

The [Alignment](#) property allows to place multilevel labels in a [Far](#), [Center](#), or [Near](#) location.

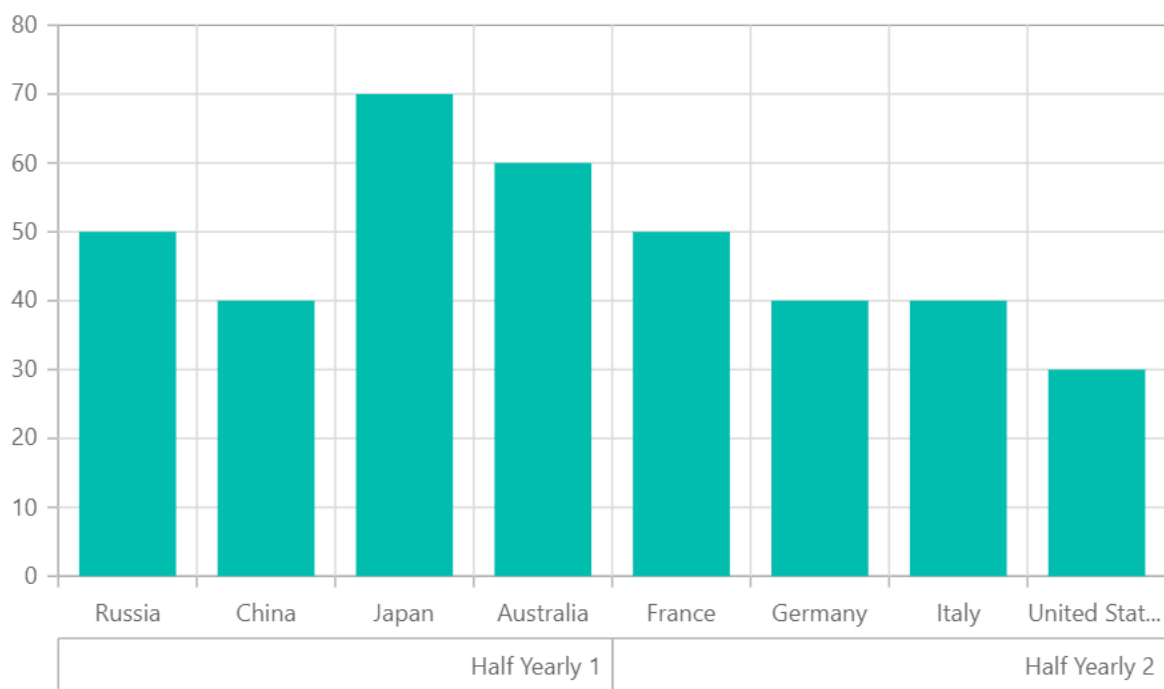
ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart Title="Olympic Medals">
  <ChartPrimaryXAxis ValueType="Syncfusion.Blazor.Charts.ValueType.Category">
    <ChartMultiLevelLabels>
      <ChartMultiLevelLabel Alignment="Alignment.Far">
        <ChartCategories>
          <ChartCategory Start="-0.5" End="3.5" Text="Half yearly 1"
            MaximumTextWidth=100></ChartCategory>
          <ChartCategory Start="3.5" End="7.5" Text="Half yearly 2"
            MaximumTextWidth=100></ChartCategory>
        </ChartCategories>
      </ChartMultiLevelLabel>
    </ChartMultiLevelLabels>
  </ChartPrimaryXAxis>
  <ChartSeriesCollection>
    <ChartSeries DataSource="@MedalDetails" XName="X" YName="Y"
      Type="ChartSeriesType.Column">
    </ChartSeries>
  </ChartSeriesCollection>
</SfChart>
@code{
  public int start = 0, end = 30;
```

```

public class ChartData
{
    public string X { get; set; }
    public double Y { get; set; }
}
public List<ChartData> MedalDetails = new List<ChartData>
{
    new ChartData { X= "Russia", Y= 50 },
    new ChartData { X= "China", Y= 40 },
    new ChartData { X= "Japan", Y= 70 },
    new ChartData { X= "Australia", Y= 60 },
    new ChartData { X= "France", Y= 70 },
    new ChartData { X= "Germany", Y= 40 },
    new ChartData { X= "Italy", Y= 40 },
    new ChartData { X= "United states", Y= 30 },
};
}

```



Text Customization

The [Size](#), [Color](#), [FontFamily](#), [FontWeight](#), [FontStyle](#), [Opacity](#), [TextAlignment](#) and [TextOverflow](#) properties can be customized using the [TextStyle](#) of multilevel labels.

ASPX-CS

```

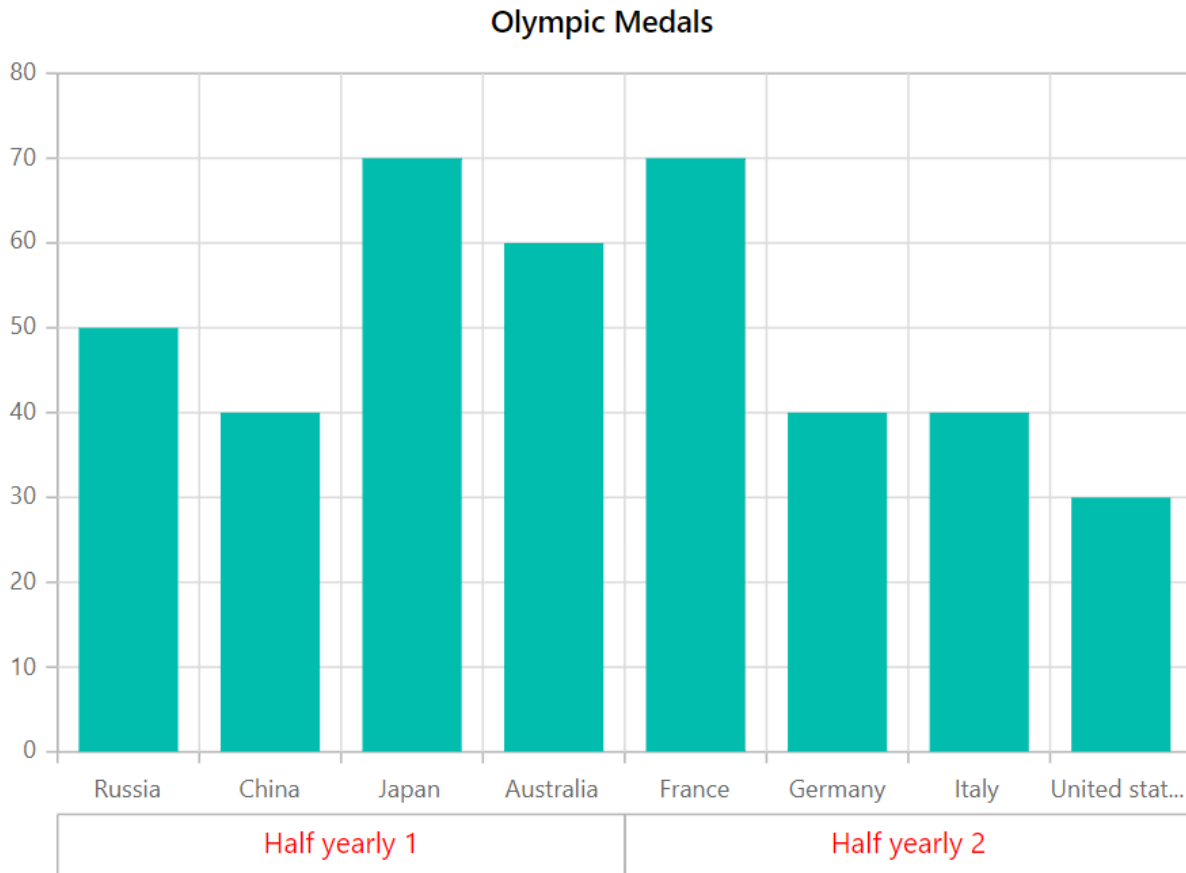
@using Syncfusion.Blazor.Charts
<SfChart Title="Olympic Medals">
    <ChartPrimaryXAxis Value="Syncfusion.Blazor.Charts.ValueType.Category">
        <ChartMultiLevelLabels>
            <ChartMultiLevelLabel>
                <ChartCategories>

```

```

<ChartCategory Start="-0.5" End="3.5" Text="Half yearly 1"
MaximumTextWidth=100></ChartCategory>
<ChartCategory Start="3.5" End="7.5" Text="Half yearly 2"
MaximumTextWidth=100></ChartCategory>
</ChartCategories>
<ChartAxisMultiLevelLabelTextStyle Size="14px" Color="red"/>
</ChartMultiLevelLabel>
</ChartMultiLevelLabels>
</ChartPrimaryXAxis>
<ChartSeriesCollection>
<ChartSeries DataSource="@MedalDetails" XName="X" YName="Y"
Type="ChartSeriesType.Column">
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code{
public int start = 0, end = 30;
public class ChartData
{
public string X { get; set; }
public double Y { get; set; }
}
public List<ChartData> MedalDetails = new List<ChartData>
{
new ChartData { X= "Russia", Y= 50 },
new ChartData { X= "China", Y= 40 },
new ChartData { X= "Japan", Y= 70 },
new ChartData { X= "Australia", Y= 60 },
new ChartData { X= "France", Y= 70 },
new ChartData { X= "Germany", Y= 40 },
new ChartData { X= "Italy", Y= 40 },
new ChartData { X= "United states", Y= 30 },
};
}

```



Border Customization

The [Width](#), [Color](#), and [Type](#) of the border can be customized using the [ChartAxisMultiLevelLabelBorder](#). [Rectangle](#), [Brace](#), [WithoutBorder](#), [WithoutTopBorder](#), [WithoutTopandBottomBorder](#), [Auto](#), and [CurlyBrace](#) are the different types of borders available.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart Title="Olympic Medals">
  <ChartPrimaryXAxis Value="Syncfusion.Blazor.Charts.ValueType.Category">
    <ChartMultiLevelLabels>
      <ChartMultiLevelLabel>
        <ChartCategories>
          <ChartCategory Start="-0.5" End="3.5" Text="Half yearly 1"
            MaximumTextWidth="50"></ChartCategory>
          <ChartCategory Start="3.5" End="7.5" Text="Half yearly 2"
            MaximumTextWidth="50"></ChartCategory>
        </ChartCategories>
        <ChartAxisMultiLevelLabelBorder Type="BorderType.Brace" Color="blue"
          Width=2></ChartAxisMultiLevelLabelBorder>
      </ChartMultiLevelLabel>
    </ChartMultiLevelLabels>
  </ChartPrimaryXAxis>
  <ChartSeriesCollection>
    <ChartSeries DataSource="@MedalDetails" XName="X" YName="Y"
      Type="ChartSeriesType.Column">

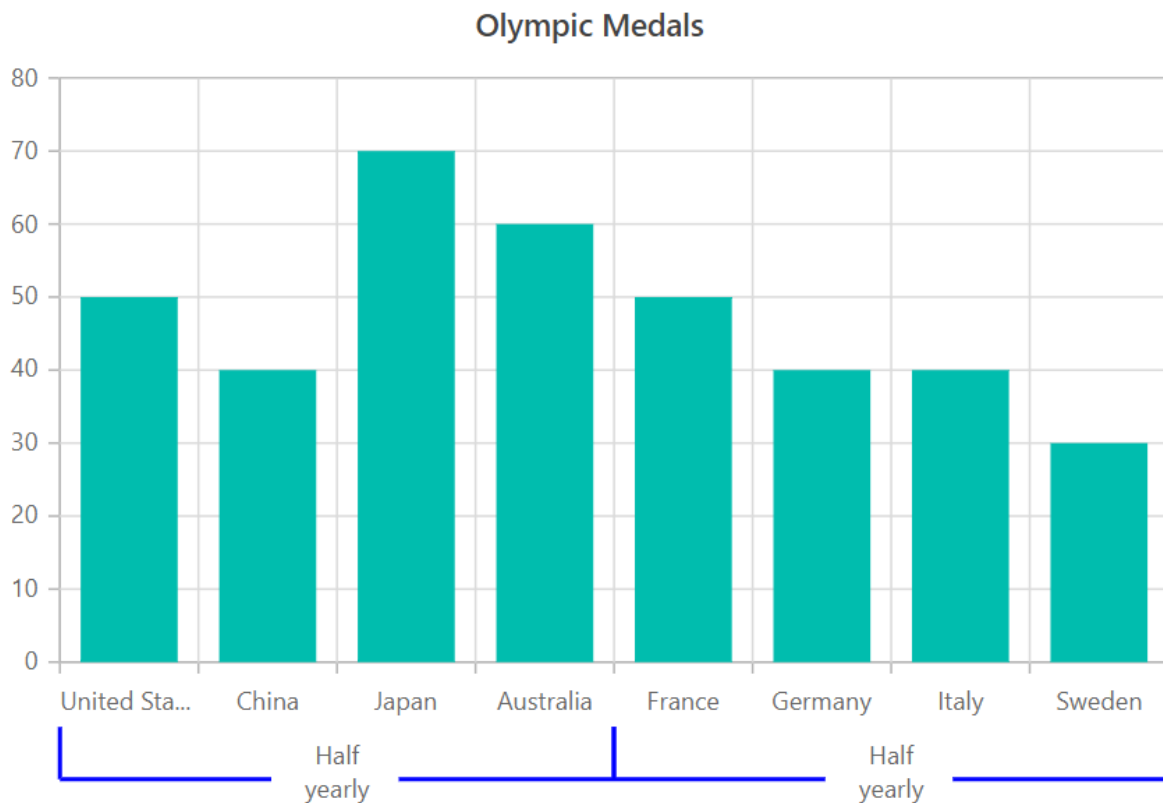
```



```

</ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code{
public int start = 0, end = 30;
public class ChartData
{
public string X { get; set; }
public double Y { get; set; }
}
public List<ChartData> MedalDetails = new List<ChartData>
{
new ChartData { X= "Russia", Y= 50 },
new ChartData { X= "China", Y= 40 },
new ChartData { X= "Japan", Y= 70 },
new ChartData { X= "Australia", Y= 60 },
new ChartData { X= "France", Y= 70 },
new ChartData { X= "Germany", Y= 40 },
new ChartData { X= "Italy", Y= 40 },
new ChartData { X= "United states", Y= 30 },
};
}

```

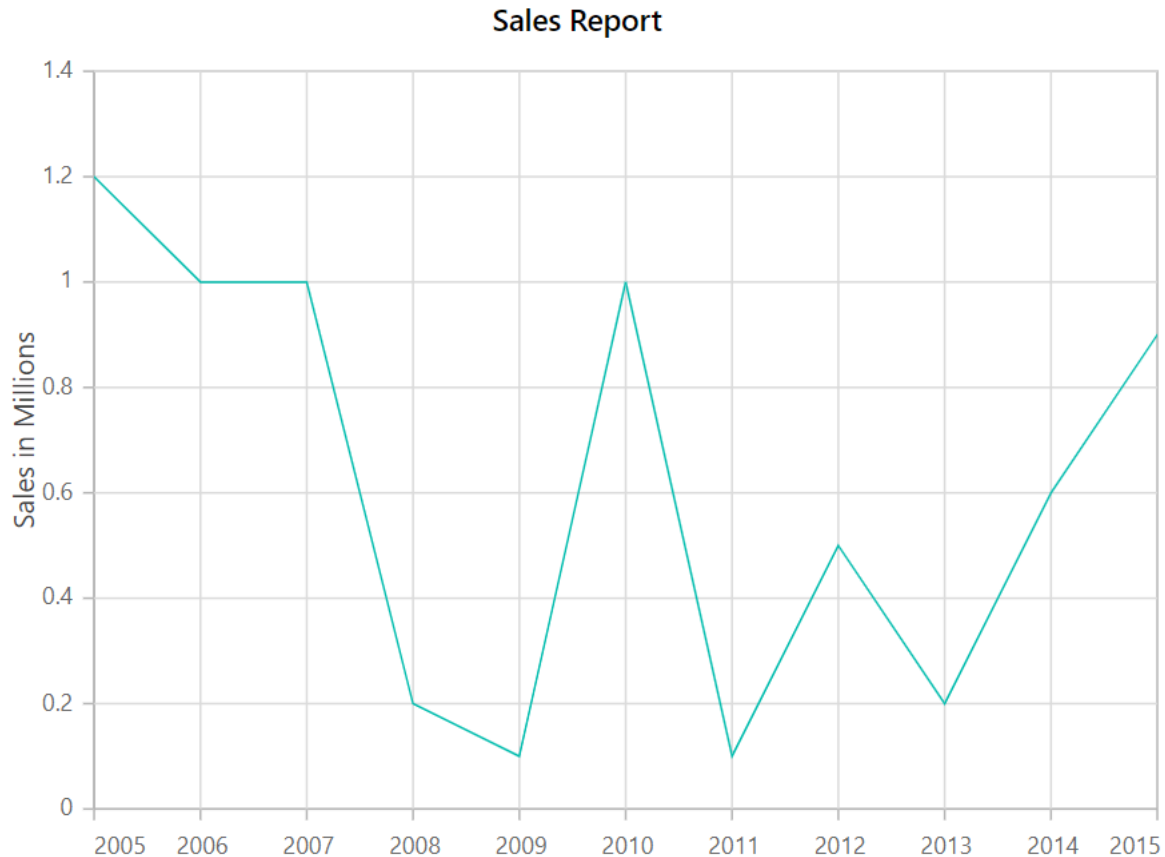


Edge Label Placement

The longer text labels at the axes edges may only be partially visible in the chart. To avoid this, utilize the [EdgeLabelPlacement](#) property, which moves or hides the label within the chart area for a better user experience using the [Shift](#) and [Hide](#) options. [None](#) will leave the text as it is.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart Title="Sales Report">
<ChartPrimaryXAxis LabelPlacement="LabelPlacement.OnTicks"
EdgeLabelPlacement="EdgeLabelPlacement.Shift"
ValueType="Syncfusion.Blazor.Charts.ValueType.Category">
<ChartAxisLabelStyle Size="18px" Color="red" />
</ChartPrimaryXAxis>
<ChartPrimaryYAxis Title="Sales in Millions" >
</ChartPrimaryYAxis>
<ChartSeriesCollection>
<ChartSeries DataSource="@MedalDetails" XName="X" YName="Y"
Type="ChartSeriesType.Line">
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code{
public class ChartData
{
public string X { get; set; }
public double Y { get; set; }
}
public List<ChartData> SalesReports = new List<ChartData>
{
new ChartData { X= "2005", Y= 1.2},
new ChartData { X= "2006", Y= 1 },
new ChartData { X= "2007", Y= 1 },
new ChartData { X= "2008", Y= 0.2},
new ChartData { X= "2009", Y= 0.1},
new ChartData { X= "2010", Y= 1 },
new ChartData { X= "2011", Y= 0.1},
new ChartData { X= "2012", Y= 0.5},
new ChartData { X= "2013", Y= 0.2},
new ChartData { X= "2014", Y= 0.6},
new ChartData { X= "2015", Y= 0.9},
};
}
```



Labels Customization

The label [Color](#) and [Size](#) can be customized to specify in [ChartAxisLabelStyle](#).

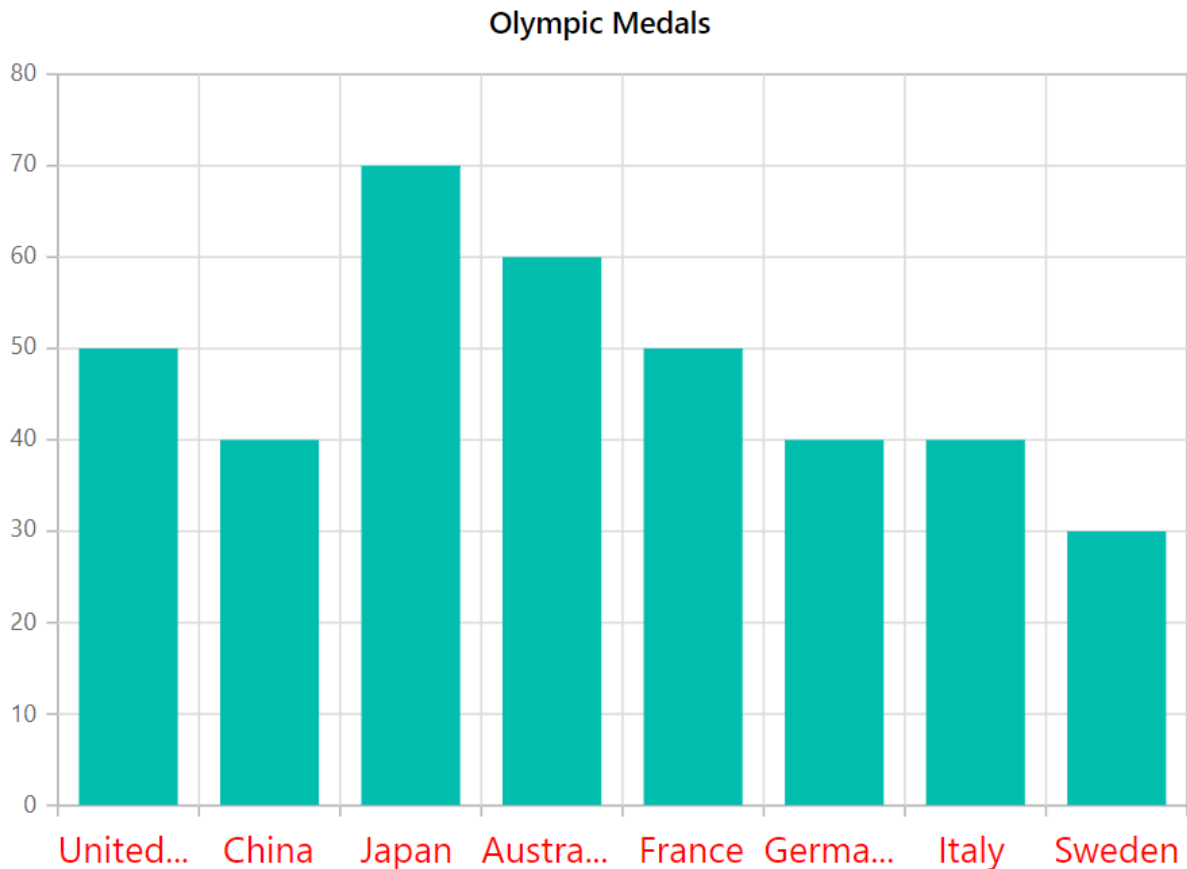
ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart Title="Olympic Medals">
  <ChartPrimaryXAxis ValueType="Syncfusion.Blazor.Charts.ValueType.Category">
    <ChartAxisLabelStyle Size="18px" Color="red" />
  </ChartPrimaryXAxis>
  <ChartSeriesCollection>
    <ChartSeries DataSource="@MedalDetails" XName="Country" YName="Gold"
      Type="ChartSeriesType.Column">
    </ChartSeries>
  </ChartSeriesCollection>
</SfChart>

@code{
public class ChartData
{
    public string Country { get; set; }
    public double Gold { get; set; }
}

public List<ChartData> MedalDetails = new List<ChartData>
{
    new ChartData{ Country= "United States", Gold=50 },
    new ChartData{ Country= "China", Gold=40 },
    new ChartData{ Country= "Japan", Gold=70 },
}
```

```
new ChartData{ Country= "Australia", Gold=60},
new ChartData{ Country= "France", Gold=50 },
new ChartData{ Country= "Germany", Gold=40 },
new ChartData{ Country= "Italy", Gold=40 },
new ChartData{ Country= "Sweden", Gold=30 }
};
}
```



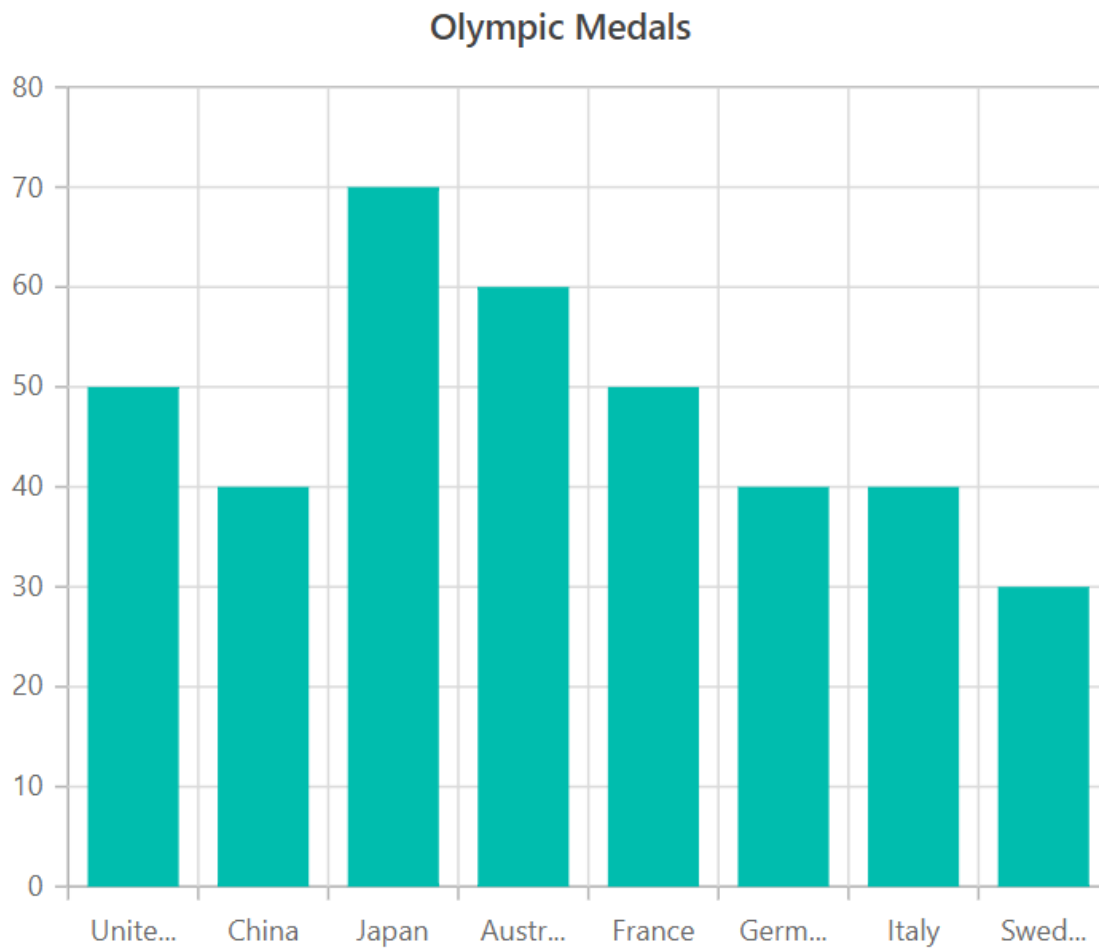
Trim Label

The label can be trimmed using the [EnableTrim](#) property, and the width of the label can be modified using the [MaximumLabelWidth](#) property in the axis. The default value of [MaximumLabelWidth](#) property is 34px.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart Title="Olympic Medals">
  <ChartPrimaryXAxis EnableTrim="true" MaximumLabelWidth="40"
    ValueType="Syncfusion.Blazor.Charts.ValueType.Category">
  </ChartPrimaryXAxis>
  <ChartSeriesCollection>
    <ChartSeries DataSource="@MedalDetails" XName="Country" YName="Gold"
      Type="ChartSeriesType.Column">
    </ChartSeries>
  </ChartSeriesCollection>
</SfChart>
```

```
@code{
public class ChartData
{
public string Country { get; set; }
public double Gold { get; set; }
}
public List<ChartData> MedalDetails = new List<ChartData>
{
new ChartData{ Country= "United States", Gold=50 },
new ChartData{ Country= "China", Gold=40 },
new ChartData{ Country= "Japan", Gold=70 },
new ChartData{ Country= "Australia", Gold=60},
new ChartData{ Country= "France", Gold=50 },
new ChartData{ Country= "Germany", Gold=40 },
new ChartData{ Country= "Italy", Gold=40 },
new ChartData{ Country= "Sweden", Gold=30 }
};
}
```

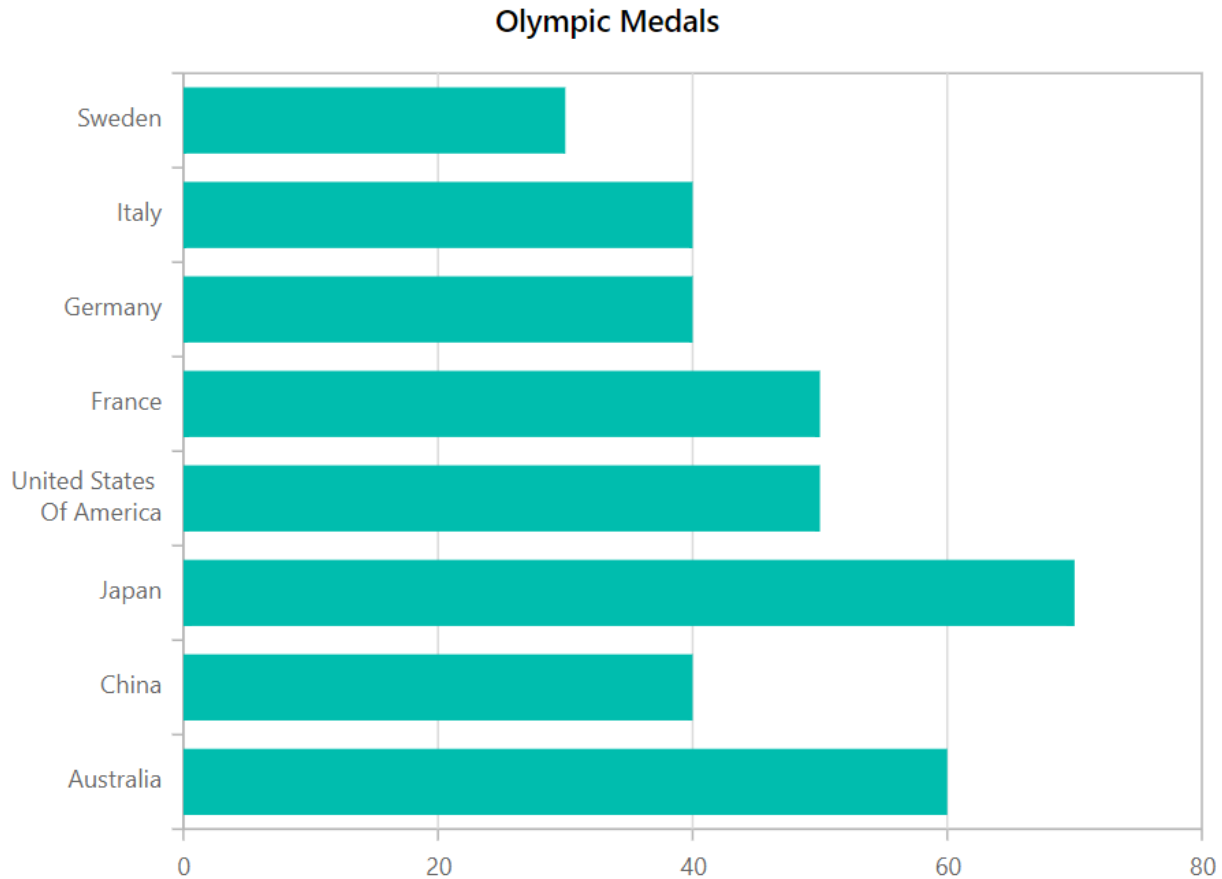


Line Break

The `
` tag can be used to separate the long axis label into multiple lines.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart Title="Olympic Medals">
<ChartPrimaryXAxis ValueType="Syncfusion.Blazor.Charts.ValueType.Category">
<ChartAxisMajorGridLines Width="0"></ChartAxisMajorGridLines>
</ChartPrimaryXAxis>
<ChartSeriesCollection>
<ChartSeries DataSource="@MedalDetails" XName="Country" YName="Gold"
Type="ChartSeriesType.Bar">
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code{
public class ChartData
{
public string Country { get; set; }
public double Gold { get; set; }
}
public List<ChartData> MedalDetails = new List<ChartData>
{
new ChartData{ Country= "Australia", Gold=60},
new ChartData{ Country= "China", Gold=40 },
new ChartData{ Country= "Japan", Gold=70 },
new ChartData{ Country= "United States<br>Of America", Gold=50 },
new ChartData{ Country= "France", Gold=50 },
new ChartData{ Country= "Germany", Gold=40 },
new ChartData{ Country= "Italy", Gold=40 },
new ChartData{ Country= "Sweden", Gold=30 }
};
}
```



Label Format

Learn more about axis label format in-relation to axis types from the pages below.

- [Numeric Label Format](#)
- [DateTime Label Format](#)
- [Custom Label Format](#)

Refer to our [Blazor Charts](#) feature tour page for its groundbreaking feature representations and also explore our [Blazor Chart Example](#) to know various chart types and how to represent time-dependent data, showing trends at equal intervals.

See Also

- [Data Label](#)
- [Tooltip](#)
- [Marker](#)

Axis Customization in Blazor Charts Component

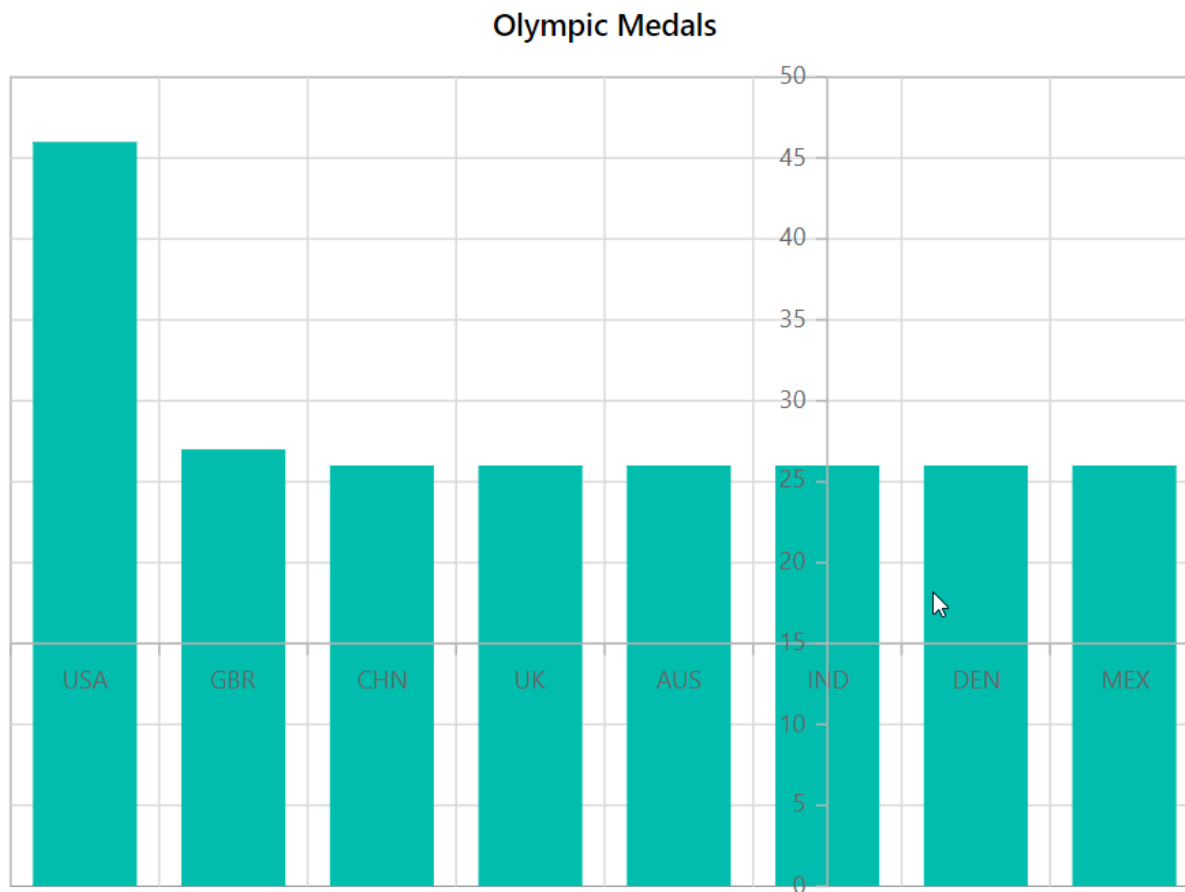
Axis Crossing

An axis can be positioned in the chart area using [CrossesAt](#) and [CrossesInAxis](#) properties. The [CrossesAt](#) property specifies the values (numeric, datetime or logarithmic) at which the axis line has to be

intersected with the vertical axis or vice-versa, and the [CrossesInAxis](#) property specifies the axis name with which the axis line has to be crossed.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart Title="Olympic Medals">
  <ChartPrimaryXAxis ValueType="Syncfusion.Blazor.Charts.ValueType.Category"
  CrossesAt="15"/>
  <ChartPrimaryYAxis CrossesAt="5"/>
  <ChartSeriesCollection>
    <ChartSeries DataSource="@MedalDetails" XName="X" YName="YValue"
    Type="ChartSeriesType.Column"/>
  </ChartSeriesCollection>
</SfChart>
@code{
public class ChartData
{
public string X { get; set; }
public double YValue { get; set; }
}
public List<ChartData> MedalDetails = new List<ChartData>
{
new ChartData { X= "USA", YValue= 46 },
new ChartData { X= "GBR", YValue= 27 },
new ChartData { X= "CHN", YValue= 26 },
new ChartData { X= "UK", YValue= 26 },
new ChartData { X= "AUS", YValue= 26 },
new ChartData { X= "IND", YValue= 26 },
new ChartData { X= "DEN", YValue= 26 },
new ChartData { X= "MEX", YValue= 26 },
};
}
```

Title

A title can be added to the axis using [Title](#) property to provide quick information to the user about the data plotted in the axis. The title text can be customized using [ChartAxisTitleStyle](#) of the axis.

ASPX-CS

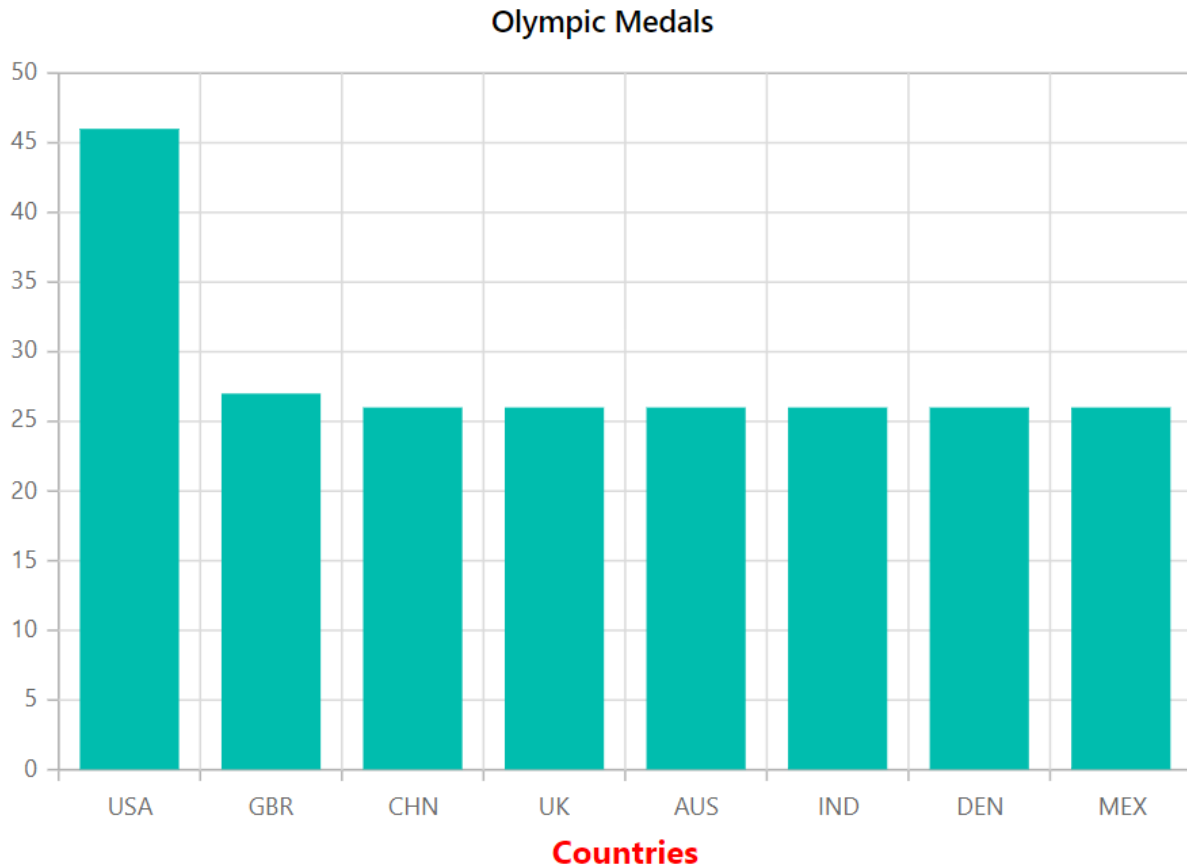
```
@using Syncfusion.Blazor.Charts
<SfChart Title="Olympic Medals">
  <ChartPrimaryXAxis Title="Countries"
    ValueType="Syncfusion.Blazor.Charts.ValueType.Category">
    <ChartAxisTitleStyle Size="16px" Color="red" FontFamily="Segoe UI"
      FontWeight="bold"/>
  </ChartPrimaryXAxis>
  <ChartSeriesCollection>
    <ChartSeries DataSource="@MedalDetails" XName="X" YName="YValue"
      Type="ChartSeriesType.Column"/>
  </ChartSeriesCollection>
</SfChart>

@code{
  public class ChartData
  {
    public string X { get; set; }
    public double YValue { get; set; }
  }
  public List<ChartData> MedalDetails = new List<ChartData>
  {
```

```

new ChartData { X= "USA", YValue= 46 },
new ChartData { X= "GBR", YValue= 27 },
new ChartData { X= "CHN", YValue= 26 },
new ChartData { X= "UK", YValue= 26 },
new ChartData { X= "AUS", YValue= 26 },
new ChartData { X= "IND", YValue= 26 },
new ChartData { X= "DEN", YValue= 26 },
new ChartData { X= "MEX", YValue= 26 },
};
}

```



Tick Lines

The width, color and size of the minor and major tick lines can be customized using [MajorTickLines](#) and [MinorTickLines](#) properties in the axis.

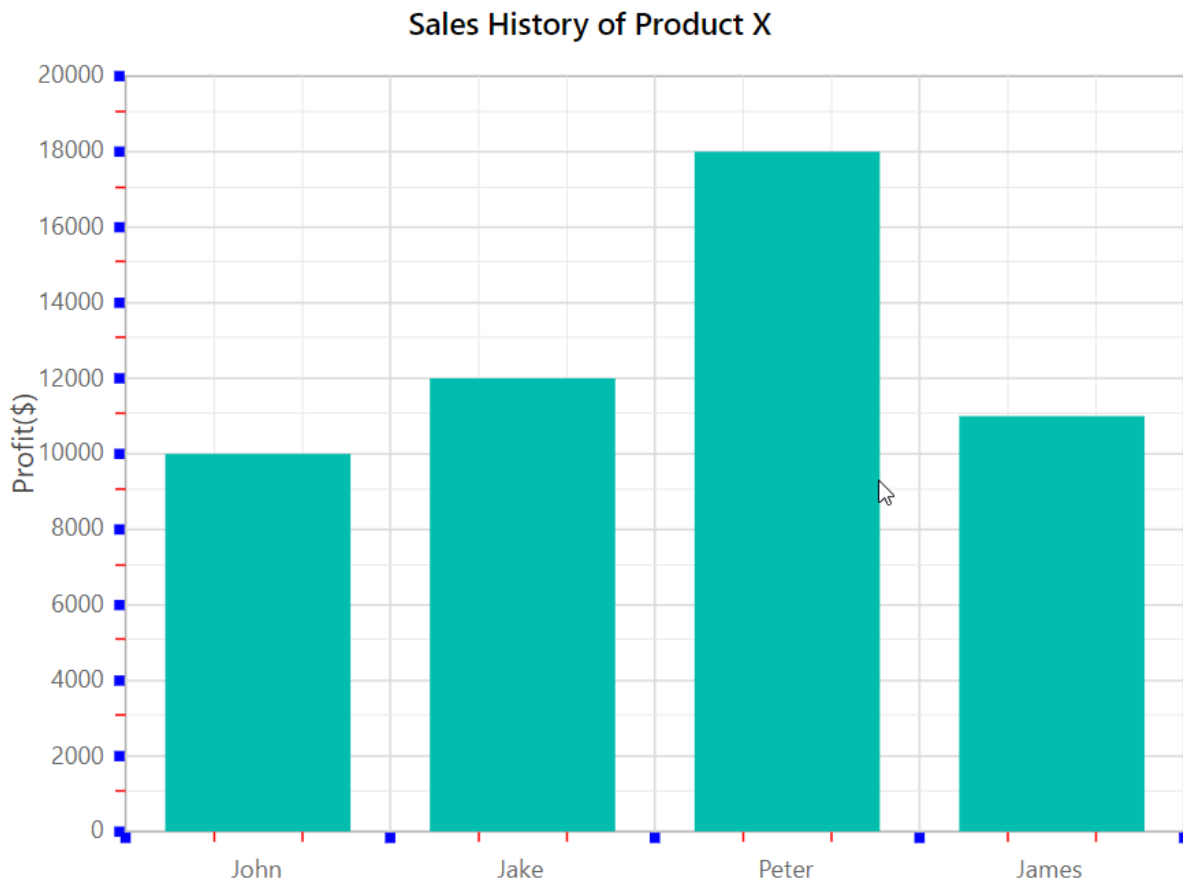
ASPX-CS

```

@using Syncfusion.Blazor.Charts
<SfChart Title="Sales History of Product X">
<ChartPrimaryXAxis ValueType="Syncfusion.Blazor.Charts.ValueType.Category"
MinorTicksPerInterval="2">
<ChartAxisMajorTickLines Width="5" Color="blue"/>
<ChartAxisMinorTickLines Width="1" Color="red"/>
</ChartPrimaryXAxis>
<ChartPrimaryYAxis Title="Profit ($)" MinorTicksPerInterval="1">
<ChartAxisMajorTickLines Width="5" Color="blue"/>

```

```
<ChartAxisMinorTickLines Width="1" Color="red"/>
</ChartPrimaryYAxis>
<ChartSeriesCollection>
<ChartSeries DataSource="@SalesData" XName="X" YName="YValue"
Type="ChartSeriesType.Column"/>
</ChartSeriesCollection>
</SfChart>
@code{
public class ChartData
{
public string X { get; set; }
public double YValue { get; set; }
}
public List<ChartData> SalesData = new List<ChartData>
{
new ChartData { X= "John", YValue= 10000 },
new ChartData { X= "Jake", YValue= 12000 },
new ChartData { X= "Peter", YValue= 18000 },
new ChartData { X= "James", YValue= 11000 }
};
}
```

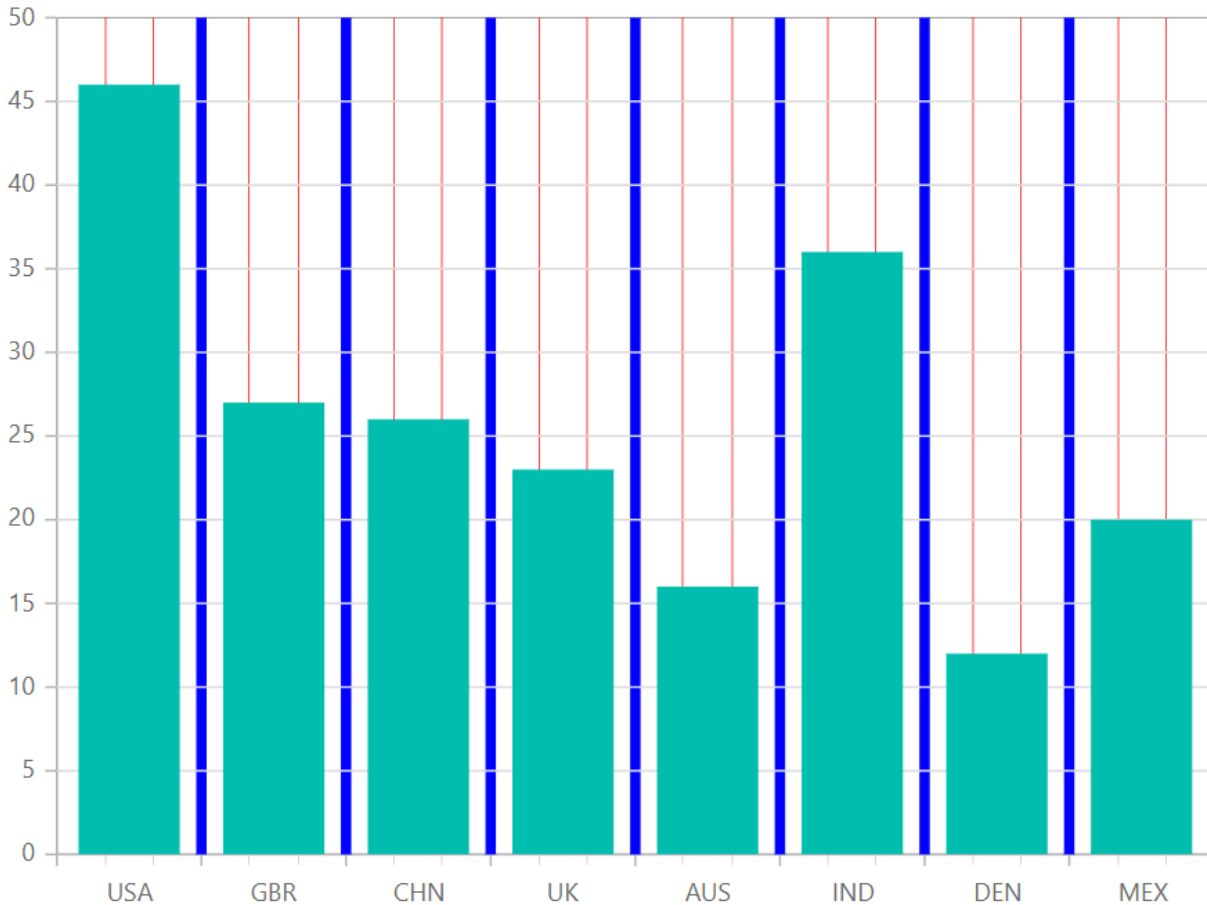


Grid Lines Customization

The width, color and dash array of the minor and major grid lines can be customized using [MajorGridLines](#) and [MinorGridLines](#) properties in the axis.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart>
<ChartPrimaryXAxis ValueType="Syncfusion.Blazor.Charts.ValueType.Category"
MinorTicksPerInterval="2">
<ChartAxisMajorGridLines Width="5" Color="blue"/>
<ChartAxisMinorGridLines Width="0.5" Color="red"/>
</ChartPrimaryXAxis>
<ChartSeriesCollection>
<ChartSeries DataSource="@MedalDetails" XName="X" YName="YValue"
Type="ChartSeriesType.Column"/>
</ChartSeriesCollection>
</SfChart>
@code{
public class ChartData
{
public string X { get; set; }
public double YValue { get; set; }
}
public List<ChartData> MedalDetails = new List<ChartData>
{
new ChartData { X= "USA", YValue= 46 },
new ChartData { X= "GBR", YValue= 27 },
new ChartData { X= "CHN", YValue= 26 },
new ChartData { X= "UK", YValue= 23 },
new ChartData { X= "AUS", YValue= 16 },
new ChartData { X= "IND", YValue= 36 },
new ChartData { X= "DEN", YValue= 12 },
new ChartData { X= "MEX", YValue= 20 },
};
}
```



Multiple Axis

The [ChartAxes](#) is a secondary axis collection that can be used to add "n" number of axes to the chart in addition to the basic X and Y axis. By mapping with the axis unique name, series can be linked to it.

ASPX-CS

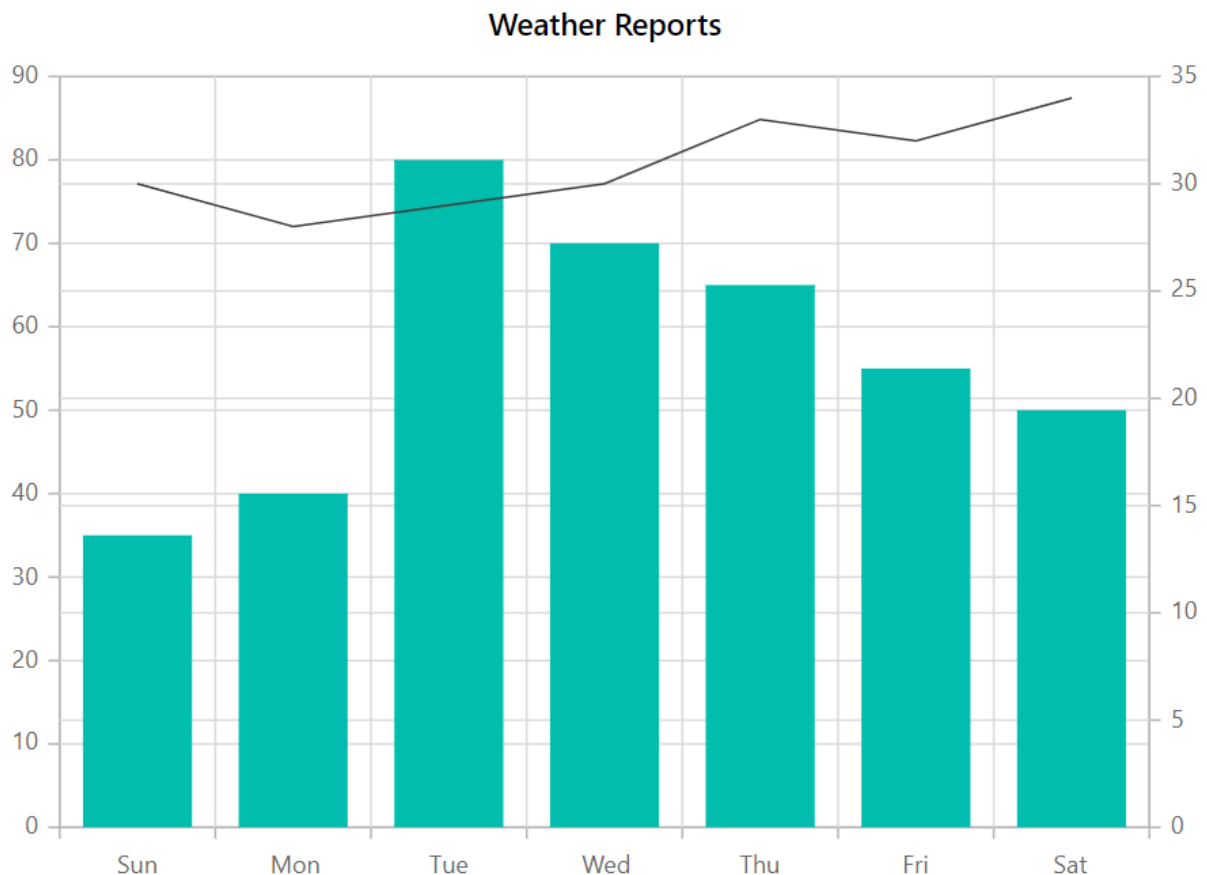
```
@using Syncfusion.Blazor.Charts
<SfChart Title="Weather Reports">
  <ChartPrimaryXAxis ValueType="Syncfusion.Blazor.Charts.ValueType.Category"/>
  <ChartAxes>
    <ChartAxis Name="YAxis" OpposedPosition="true"/>
  </ChartAxes>
  <ChartSeriesCollection>
    <ChartSeries DataSource="@WeatherReports" XName="X" YName="Y"
      Type="ChartSeriesType.Column"/>
    <ChartSeries DataSource="@WeatherReports" XName="X" YName="Y1"
      YAxisName="YAxis"/>
  </ChartSeriesCollection>
</SfChart>

@code{
  public class ChartData
  {
    public string X { get; set; }
    public double Y { get; set; }
    public double Y1 { get; set; }
  }
}
```

```

}
public List<ChartData> WeatherReports = new List<ChartData>
{
    new ChartData { X = "Sun", Y = 35, Y1 = 30 },
    new ChartData { X = "Mon", Y = 40, Y1 = 28 },
    new ChartData { X = "Tue", Y = 80, Y1 = 29 },
    new ChartData { X = "Wed", Y = 70, Y1 = 30 },
    new ChartData { X = "Thu", Y = 65, Y1 = 33 },
    new ChartData { X = "Fri", Y = 55, Y1 = 32 },
    new ChartData { X = "Sat", Y = 50, Y1 = 34 }
};
}

```



See also

- [Mixed Chart](#)
- [Multiple Panes](#)

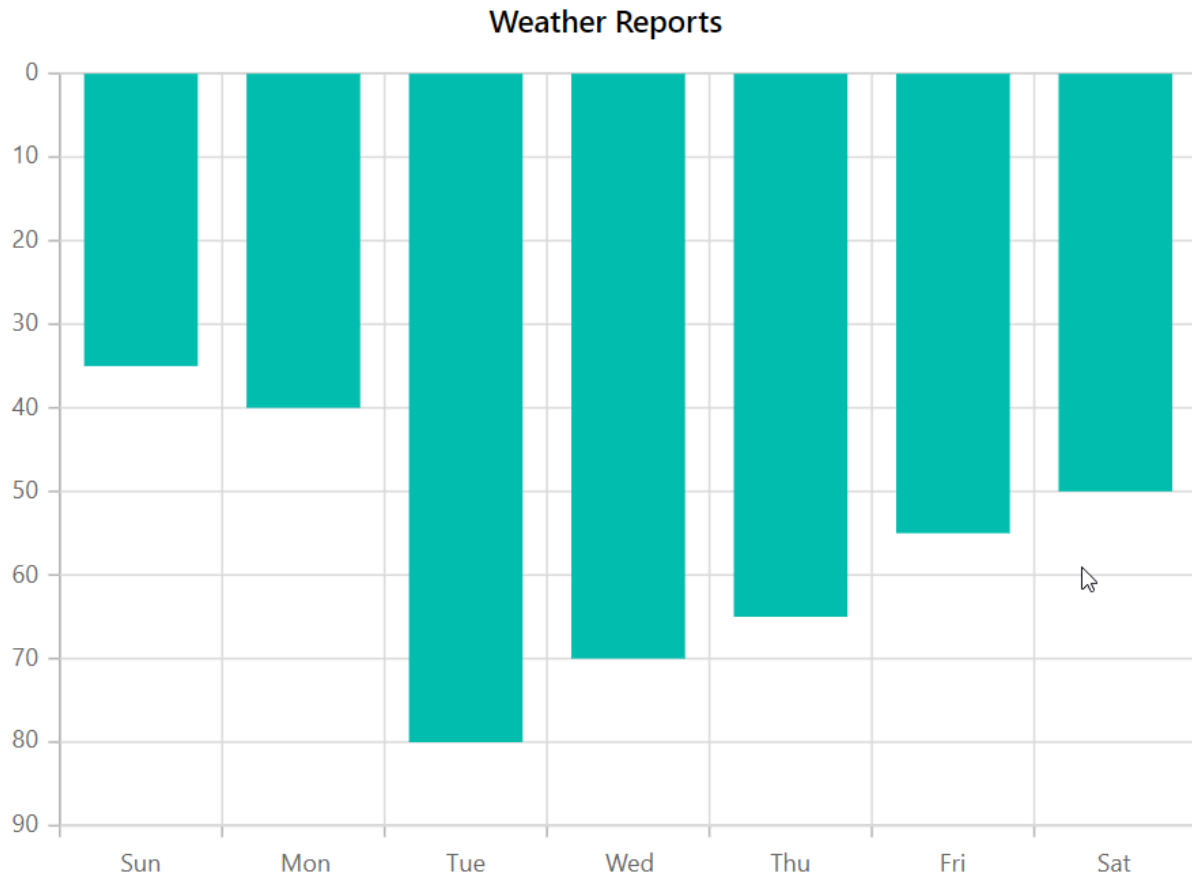
Inversed Axis

<!-- markdownlint-disable MD033 -->

When an axis is inversed, the greatest value on the axis moves closer to the origin, and vice versa. To invert an axis, set the [IsInversed](#) property to **true**.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart Title="Weather Reports">
  <ChartPrimaryXAxis ValueType="Syncfusion.Blazor.Charts.ValueType.Category"
  />
  <ChartPrimaryYAxis IsInversed="true" />
  <ChartSeriesCollection>
    <ChartSeries DataSource="@WeatherReports" XName="X" YName="Y"
    Type="ChartSeriesType.Column" />
  </ChartSeriesCollection>
</SfChart>
@code{
public class ChartData
{
public string X { get; set; }
public double Y { get; set; }
}
public List<ChartData> WeatherReports = new List<ChartData>
{
new ChartData { X = "Sun", Y = 35 },
new ChartData { X = "Mon", Y = 40 },
new ChartData { X = "Tue", Y = 80 },
new ChartData { X = "Wed", Y = 70 },
new ChartData { X = "Thu", Y = 65 },
new ChartData { X = "Fri", Y = 55 },
new ChartData { X = "Sat", Y = 50 }
};
}
```



Opposed Position

To place an axis in the opposite position of its original position, set its [OpposedPosition](#) property to **true**. It's similar to right-to-left (RTL) support.

ASPX-CS

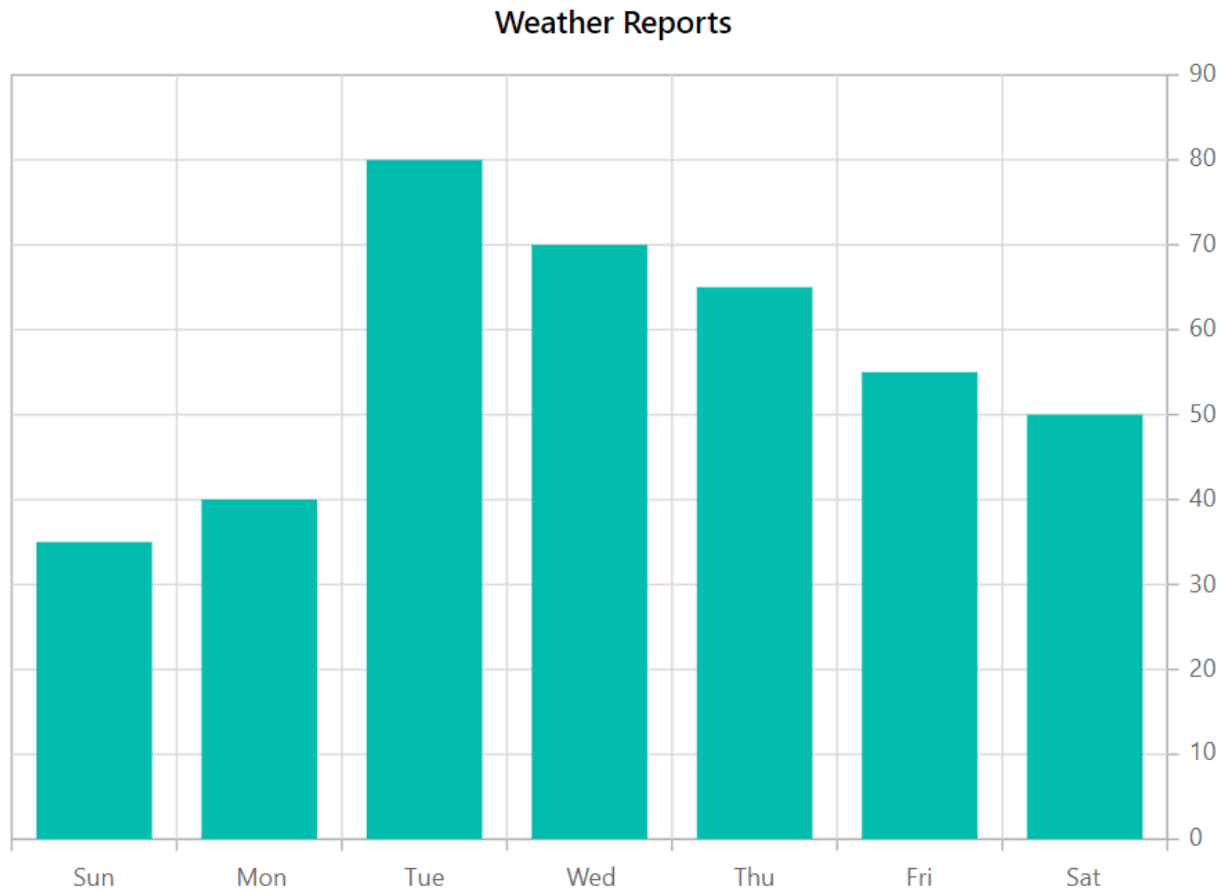
```
@using Syncfusion.Blazor.Charts
<SfChart Title="Weather Reports">
  <ChartPrimaryXAxis ValueType="Syncfusion.Blazor.Charts.ValueType.Category"/>
  <ChartPrimaryYAxis OpposedPosition="true"/>
  <ChartSeriesCollection>
    <ChartSeries DataSource="@WeatherReports" XName="X" YName="Y"
      Type="ChartSeriesType.Column"/>
  </ChartSeriesCollection>
</SfChart>

@code{
public class ChartData
{
    public string X { get; set; }
    public double Y { get; set; }
}

public List<ChartData> WeatherReports = new List<ChartData>
{
    new ChartData { X = "Sun", Y = 35 },
    new ChartData { X = "Mon", Y = 40 },
    new ChartData { X = "Tue", Y = 80 },
}
```



```
new ChartData { X = "Wed", Y = 70 },  
new ChartData { X = "Thu", Y = 65 },  
new ChartData { X = "Fri", Y = 55 },  
new ChartData { X = "Sat", Y = 50 }  
};  
}
```



Refer to our [Blazor Charts](#) feature tour page for its groundbreaking feature representations and also explore our [Blazor Chart Example](#) to know various chart types and how to represent time-dependent data, showing trends at equal intervals.

See Also

- [Data Label](#)
- [Tooltip](#)
- [Marker](#)

<!-- markdownlint-disable MD036 -->

[Stripline in Blazor Charts Component](#)

<!-- markdownlint-disable MD036 -->

Horizontal Striplines

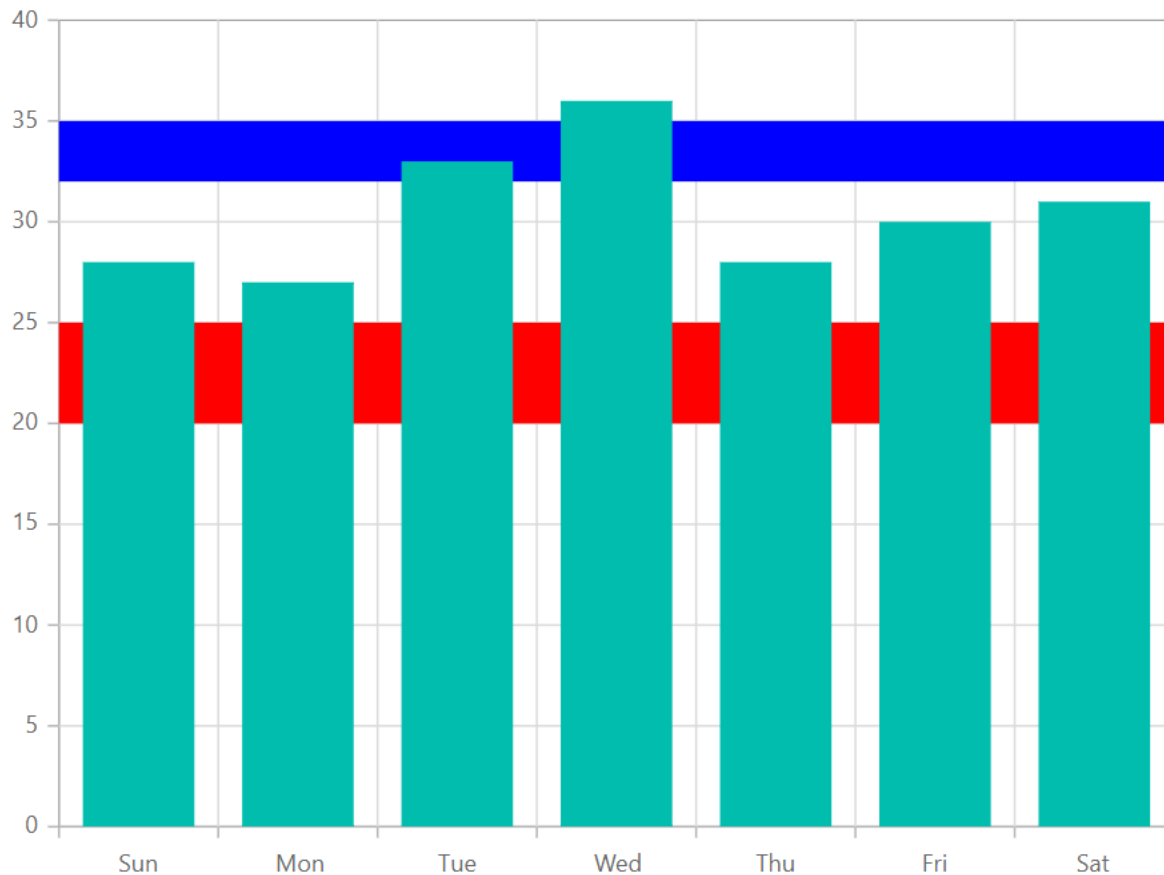
By adding the [ChartStripline](#) on the vertical axis, one can create a horizontal stripline. Striplines are drawn in the provided start-to-end range, and an axis can have multiple striplines.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart>
  <ChartPrimaryXAxis ValueType="Syncfusion.Blazor.Charts.ValueType.Category"/>
  <ChartPrimaryYAxis>
    <ChartStriplines>
      <ChartStripline Start="20" End="25" Color="red"/>
      <ChartStripline Start="32" End="35" Color="blue"/>
    </ChartStriplines>
  </ChartPrimaryYAxis>
  <ChartSeriesCollection>
    <ChartSeries Type="ChartSeriesType.Column" DataSource="@WeatherReports"
      XName="X" YName="Y">
    </ChartSeries>
  </ChartSeriesCollection>
</SfChart>

@code{
public class ChartData
{
    public string X { get; set; }
    public double Y { get; set; }
}

public List<ChartData> WeatherReports = new List<ChartData>
{
    new ChartData { X = "Sun", Y = 28 },
    new ChartData { X = "Mon", Y = 27 },
    new ChartData { X = "Tue", Y = 33 },
    new ChartData { X = "Wed", Y = 36 },
    new ChartData { X = "Thu", Y = 28 },
    new ChartData { X = "Fri", Y = 30 },
    new ChartData { X = "Sat", Y = 31 }
};
}
```



Vertical Striplines

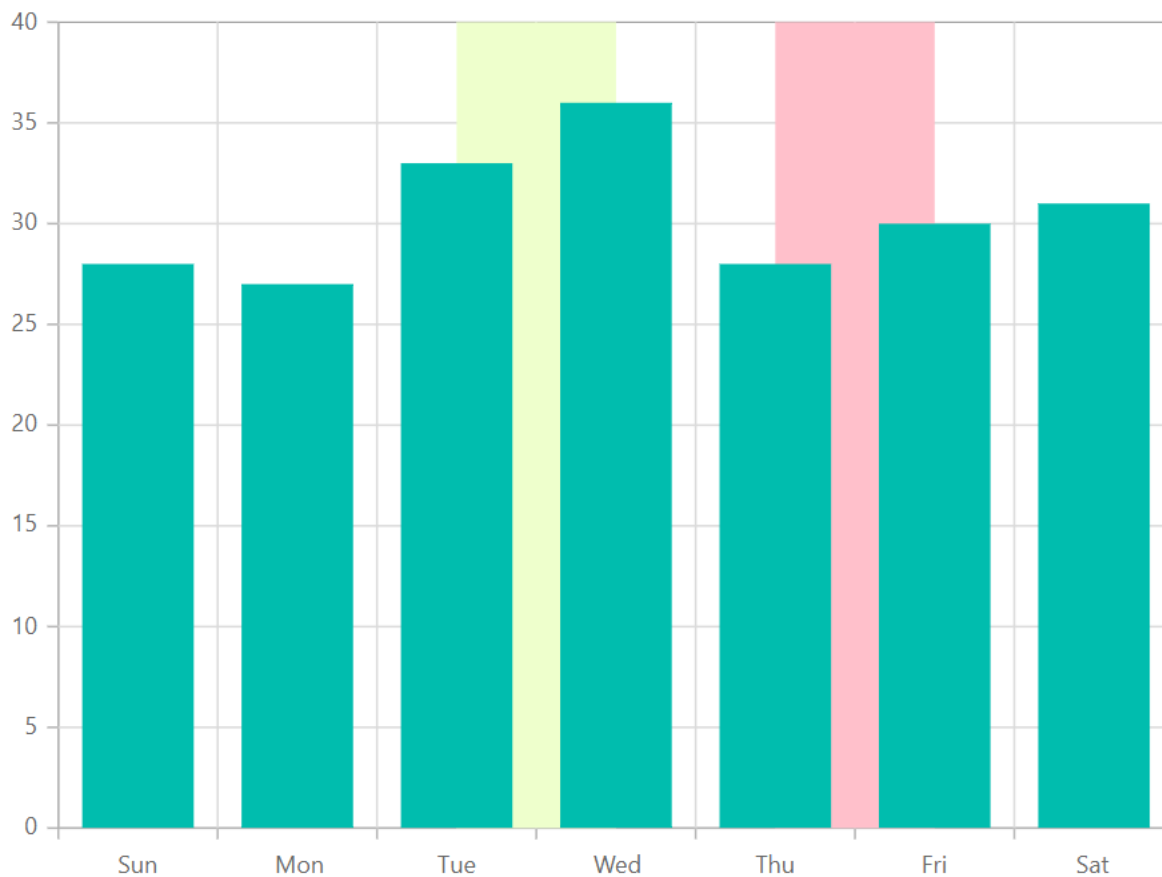
By adding the [ChartStripline](#) on the horizontal axis, one can create a vertical stripline. Striplines are drawn in the provided start-to-end range, and an axis can have multiple striplines.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart>
  <ChartPrimaryXAxis ValueType="Syncfusion.Blazor.Charts.ValueType.Category">
    <ChartStriplines>
      <ChartStripline Start="2" End="3" Color="#EEFFCC" />
      <ChartStripline Start="4" End="5" Color="pink" />
    </ChartStriplines>
  </ChartPrimaryXAxis>
  <ChartSeriesCollection>
    <ChartSeries Type="ChartSeriesType.Column" DataSource="@WeatherReports"
      XName="X" YName="Y">
    </ChartSeries>
  </ChartSeriesCollection>
</SfChart>

@code{
public class ChartData
{
  public string X { get; set; }
  public double Y { get; set; }
}
public List<ChartData> WeatherReports = new List<ChartData>
```

```
{
new ChartData { X = "Sun", Y = 28 },
new ChartData { X = "Mon", Y = 27 },
new ChartData { X = "Tue", Y = 33 },
new ChartData { X = "Wed", Y = 36 },
new ChartData { X = "Thu", Y = 28 },
new ChartData { X = "Fri", Y = 30 },
new ChartData { X = "Sat", Y = 31 }
};
}
```



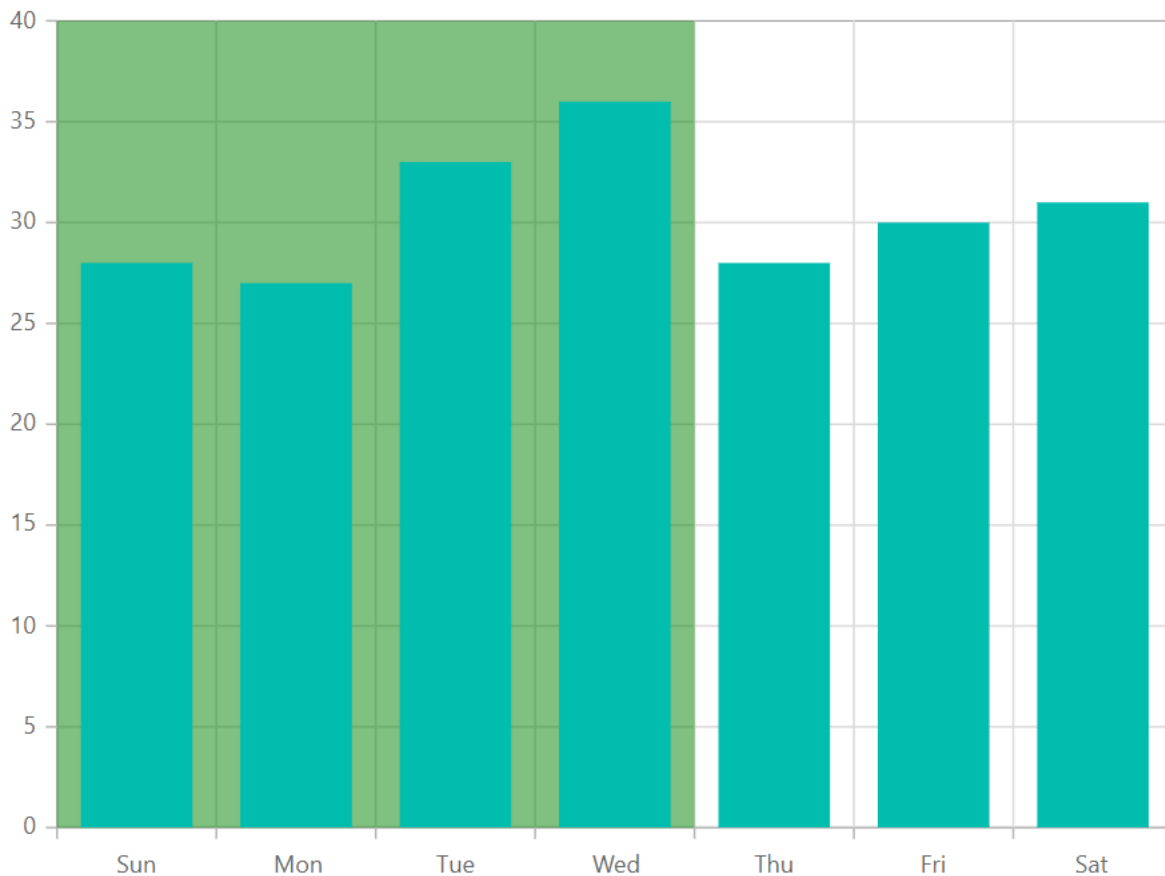
Striplines Customization

The [Start](#) property in a stripline can be used to customize the starting value in that stripline. The [End](#) property customizes the end value in the same way. Both [Size](#) and [Border](#) properties can be used to customize the stripline's size and border. The [ZIndex](#) property can be used to alter the order of the stripline, determining whether it should be drawn behind or over the series elements.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart>
<ChartPrimaryXAxis ValueType="Syncfusion.Blazor.Charts.ValueType.Category">
<ChartStriplines>
<ChartStripline StartFromAxis="true" Size="4" ZIndex="ZIndex.Behind"
Opacity="0.5" Color="green"/>
</ChartStriplines>
</ChartPrimaryXAxis>
</SfChart>
```

```
</ChartStriplines>
</ChartPrimaryXAxis>
<ChartSeriesCollection>
<ChartSeries Type="ChartSeriesType.Column" DataSource="@WeatherReports"
XName="X" YName="Y">
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code{
public class ChartData
{
public string X { get; set; }
public double Y { get; set; }
}
public List<ChartData> WeatherReports = new List<ChartData>
{
new ChartData { X = "Sun", Y = 28 },
new ChartData { X = "Mon", Y = 27 },
new ChartData { X = "Tue", Y = 33 },
new ChartData { X = "Wed", Y = 36 },
new ChartData { X = "Thu", Y = 28 },
new ChartData { X = "Fri", Y = 30 },
new ChartData { X = "Sat", Y = 31 }
};
}
```

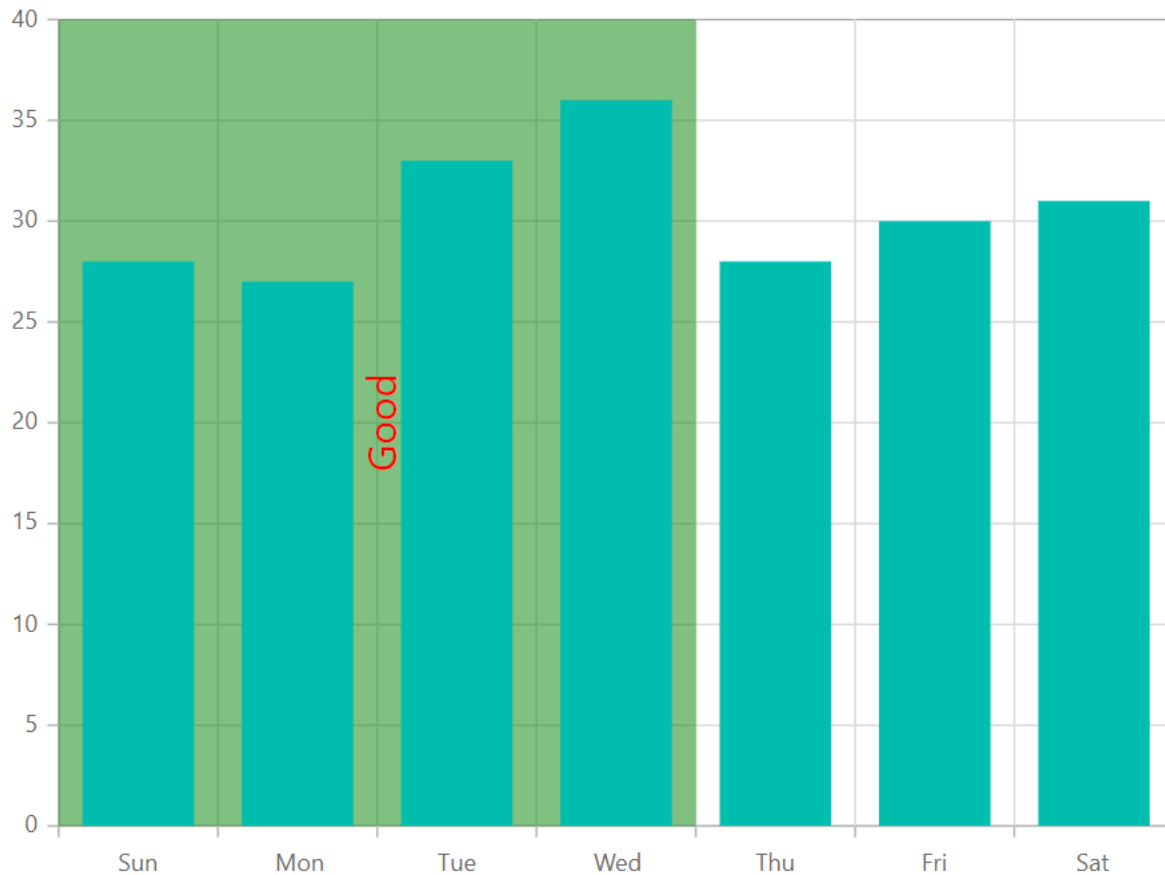


Text Customization

[TextStyle](#) and [Rotation](#) properties can be used to customize and rotate the text presented in a stripline. The [HorizontalAlignment](#) and [VerticalAlignment](#) properties can be used to customize the horizontal and vertical alignment of the stripline text.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart>
<ChartPrimaryXAxis ValueType="Syncfusion.Blazor.Charts.ValueType.Category">
<ChartStriplines>
<ChartStripline StartFromAxis="true" Size="4" ZIndex="ZIndex.Behind"
Opacity="0.5" Color="green" Text="Good"
HorizontalAlignment="Anchor.Middle" VerticalAlignment="Anchor.Middle">
<ChartStriplineTextStyle Size="20px" Color="red"/>
</ChartStripline>
</ChartStriplines>
</ChartPrimaryXAxis>
<ChartSeriesCollection>
<ChartSeries Type="ChartSeriesType.Column" DataSource="@WeatherReports"
XName="X" YName="Y">
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code{
public class ChartData
{
public string X { get; set; }
public double Y { get; set; }
}
public List<ChartData> WeatherReports = new List<ChartData>
{
new ChartData { X = "Sun", Y = 28 },
new ChartData { X = "Mon", Y = 27 },
new ChartData { X = "Tue", Y = 33 },
new ChartData { X = "Wed", Y = 36 },
new ChartData { X = "Thu", Y = 28 },
new ChartData { X = "Fri", Y = 30 },
new ChartData { X = "Sat", Y = 31 }
};
}
```



Refer to our [Blazor Charts](#) feature tour page for its groundbreaking feature representations and also explore our [Blazor Chart Example](#) to know various chart types and how to represent time-dependent data, showing trends at equal intervals.

See Also

- [Data label](#)
- [Tooltip](#)
- [Marker](#)

Multiple Panes in Blazor Charts Component

The chart area can be divided into multiple panes using [Rows](#) and [Columns](#) settings.

Rows

Use the chart's [Rows](#) property to divide the chart area vertically into any number of rows.

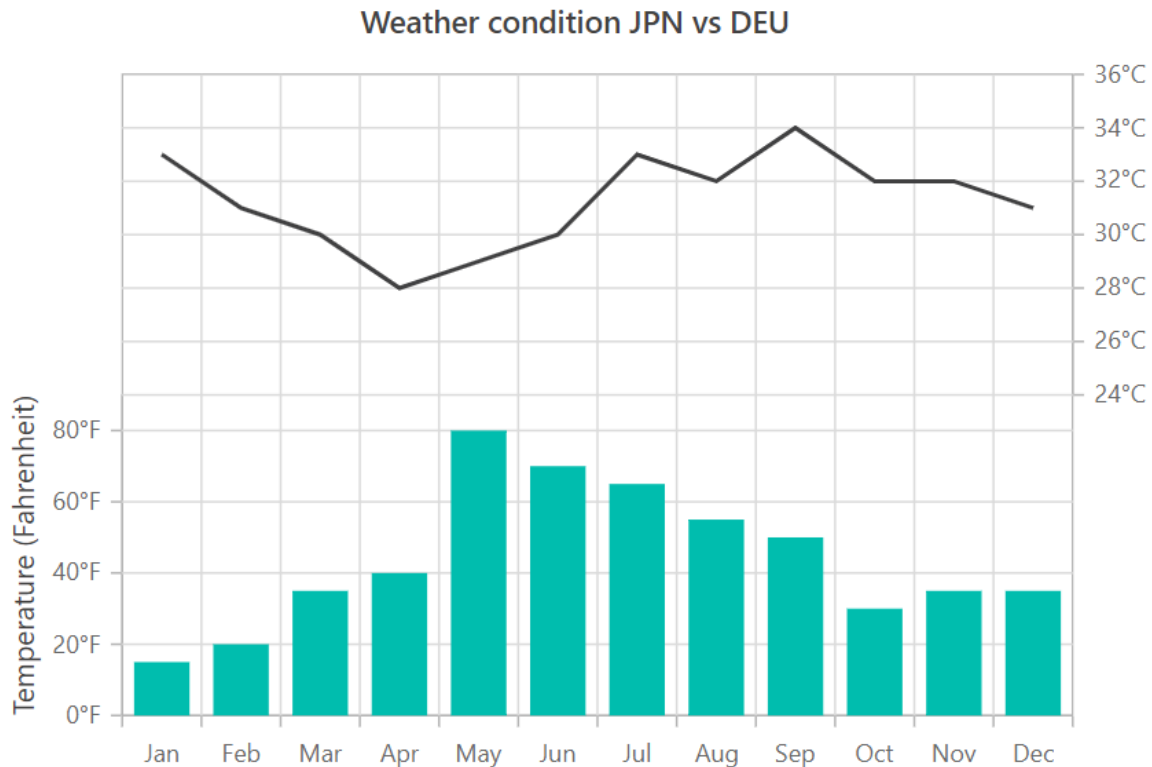
- The [Height](#) property can be used to allocate space for each row. The value can be expressed either in percentage or pixel.
- To bind a vertical axis to a specific row, set the axis's [RowIndex](#) property to that row's index.
- The bottom line of each row can be customized by specified in [ChartBorder](#).

ASPX-CS

```

@using Syncfusion.Blazor.Charts
<SfChart Title="Weather condition JPN vs DEU">
<ChartPrimaryXAxis ValueTypes="Syncfusion.Blazor.Charts.ValueType.Category"/>
<ChartPrimaryYAxis Title="Temperature (Fahrenheit)" LabelFormat="{value}°F"
Minimum="0" Maximum="90" Interval="20"/>
<ChartRows>
<ChartRow Height="50%"/>
<ChartRow Height="50%"/>
</ChartRows>
<ChartAxes>
<ChartAxis Minimum="24" Maximum="36" Interval="2" OpposedPosition="true"
RowIndex="1" Name="YAxis" LabelFormat="{value}°C"/>
</ChartAxes>
<ChartSeriesCollection>
<ChartSeries DataSource="@WeatherReports" XName="X" YName="Y"
Type="ChartSeriesType.Column"/>
<ChartSeries DataSource="@WeatherReports" XName="X" YName="Y1"
YAxisName="YAxis"/>
</ChartSeriesCollection>
</SfChart>
@code{
public class ChartData
{
public string X { get; set; }
public double Y { get; set; }
public double Y1 { get; set; }
}
public List<ChartData> WeatherReports = new List<ChartData>
{
new ChartData{ X= "Jan", Y= 15, Y1= 33 },
new ChartData{ X= "Feb", Y= 20, Y1= 31 },
new ChartData{ X= "Mar", Y= 35, Y1= 30 },
new ChartData{ X= "Apr", Y= 40, Y1= 28 },
new ChartData{ X= "May", Y= 80, Y1= 29 },
new ChartData{ X= "Jun", Y= 70, Y1= 30 },
new ChartData{ X= "Jul", Y= 65, Y1= 33 },
new ChartData{ X= "Aug", Y= 55, Y1= 32 },
new ChartData{ X= "Sep", Y= 50, Y1= 34 },
new ChartData{ X= "Oct", Y= 30, Y1= 32 },
new ChartData{ X= "Nov", Y= 35, Y1= 32 },
new ChartData{ X= "Dec", Y= 35, Y1= 31 }
};
}

```

The [Span](#) property of the axis can be used to span the vertical axis across multiple rows.

ASPX-CS

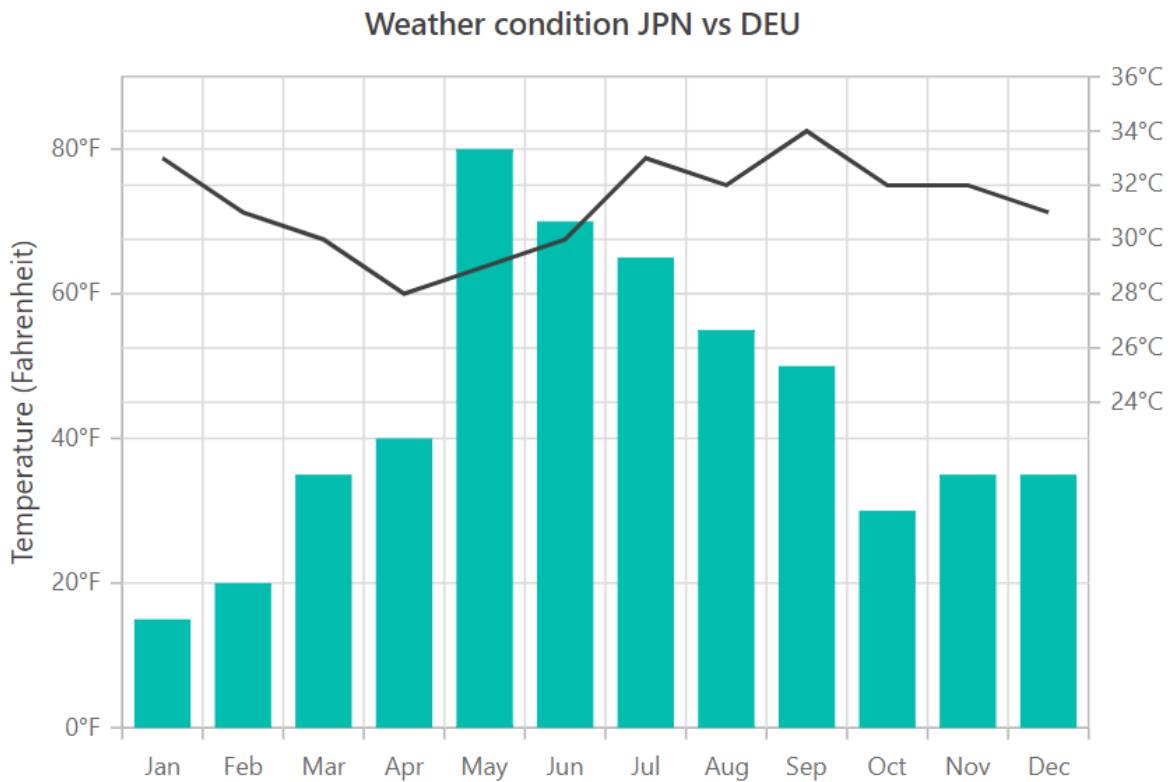
```
@using Syncfusion.Blazor.Charts
<SfChart Title="Weather condition JPN vs DEU">
  <ChartPrimaryXAxis ValueType="Syncfusion.Blazor.Charts.ValueType.Category"/>
  <ChartPrimaryYAxis Span="2" Title="Temperature (Fahrenheit)"
    LabelFormat="{value}°F" Minimum="0" Maximum="90" Interval="20"/>
  <ChartRows>
    <ChartRow Height="50%"/>
    <ChartRow Height="50%"/>
  </ChartRows>
  <ChartAxes>
    <ChartAxis Minimum="24" Maximum="36" Interval="2" OpposedPosition="true"
      RowIndex="1" Name="YAxis" LabelFormat="{value}°C"/>
  </ChartAxes>
  <ChartSeriesCollection>
    <ChartSeries DataSource="@WeatherReports" XName="X" YName="Y"
      Type="ChartSeriesType.Column"/>
    <ChartSeries DataSource="@WeatherReports" XName="X" YName="Y1"
      YAxisName="YAxis"/>
  </ChartSeriesCollection>
</SfChart>

@code{
public class ChartData
{
    public string X { get; set; }
    public double Y { get; set; }
    public double Y1 { get; set; }
}
```

```

}
public List<ChartData> WeatherReports = new List<ChartData>
{
    new ChartData{ X= "Jan", Y= 15, Y1= 33 },
    new ChartData{ X= "Feb", Y= 20, Y1= 31 },
    new ChartData{ X= "Mar", Y= 35, Y1= 30 },
    new ChartData{ X= "Apr", Y= 40, Y1= 28 },
    new ChartData{ X= "May", Y= 80, Y1= 29 },
    new ChartData{ X= "Jun", Y= 70, Y1= 30 },
    new ChartData{ X= "Jul", Y= 65, Y1= 33 },
    new ChartData{ X= "Aug", Y= 55, Y1= 32 },
    new ChartData{ X= "Sep", Y= 50, Y1= 34 },
    new ChartData{ X= "Oct", Y= 30, Y1= 32 },
    new ChartData{ X= "Nov", Y= 35, Y1= 32 },
    new ChartData{ X= "Dec", Y= 35, Y1= 31 }
};
}

```



Columns

Use the chart's [Columns](#) property to divide the chart area horizontally into any number of columns.

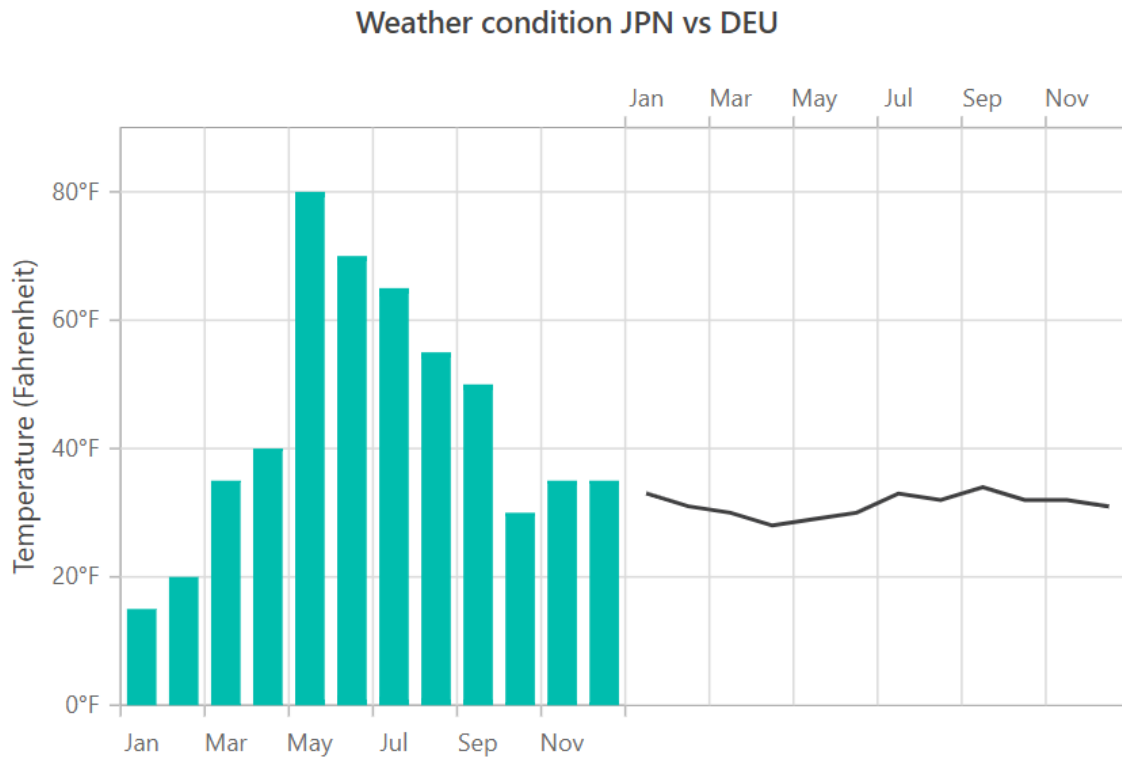
- The [Width](#) property can be used to allocate space for each column. The value can be expressed either in percentage or pixel.
- To bind a horizontal axis to a specific column, set the axis's [ColumnIndex](#) property to that column's index.
- The left line of each column can be customized by specified in [ChartBorder](#).

ASPX-CS

```

@using Syncfusion.Blazor.Charts
<SfChart Title="Weather condition JPN vs DEU">
<ChartPrimaryXAxis ValueType="Syncfusion.Blazor.Charts.ValueType.Category"/>
<ChartPrimaryYAxis Title="Temperature (Fahrenheit)" LabelFormat="{value}°F"
Minimum="0" Maximum="90" Interval="20"/>
<ChartColumns>
<ChartColumn Width="50%"/>
<ChartColumn Width="50%"/>
</ChartColumns>
<ChartAxes>
<ChartAxis Interval="2" OpposedPosition="true" ColumnIndex="1" Name="XAxis"
ValueType="Syncfusion.Blazor.Charts.ValueType.Category"/>
</ChartAxes>
<ChartSeriesCollection>
<ChartSeries DataSource="@WeatherReports" XName="X" YName="Y"
Type="ChartSeriesType.Column"/>
<ChartSeries DataSource="@WeatherReports" XName="X" YName="Y1"
XAxisName="XAxis" />
</ChartSeriesCollection>
</SfChart>
@code{
public class ChartData
{
public string X { get; set; }
public double Y { get; set; }
public double Y1 { get; set; }
}
public List<ChartData> WeatherReports = new List<ChartData>
{
new ChartData{ X= "Jan", Y= 15, Y1= 33 },
new ChartData{ X= "Feb", Y= 20, Y1= 31 },
new ChartData{ X= "Mar", Y= 35, Y1= 30 },
new ChartData{ X= "Apr", Y= 40, Y1= 28 },
new ChartData{ X= "May", Y= 80, Y1= 29 },
new ChartData{ X= "Jun", Y= 70, Y1= 30 },
new ChartData{ X= "Jul", Y= 65, Y1= 33 },
new ChartData{ X= "Aug", Y= 55, Y1= 32 },
new ChartData{ X= "Sep", Y= 50, Y1= 34 },
new ChartData{ X= "Oct", Y= 30, Y1= 32 },
new ChartData{ X= "Nov", Y= 35, Y1= 32 },
new ChartData{ X= "Dec", Y= 35, Y1= 31 }
};
}

```



The [Span](#) property of the axis can be used to span the horizontal axis across multiple column.

ASPX-CS

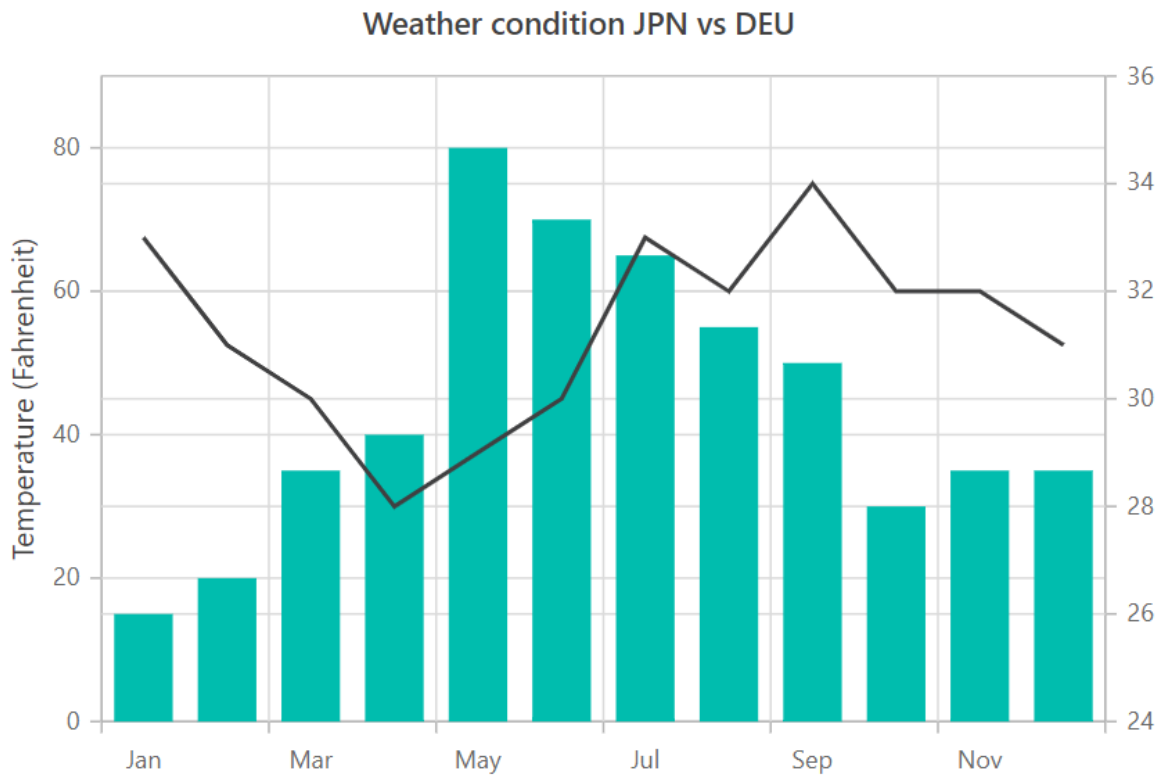
```
@using Syncfusion.Blazor.Charts
<SfChart Title="Weather condition JPN vs DEU">
  <ChartPrimaryXAxis Span="2"
  ValueType="Syncfusion.Blazor.Charts.ValueType.Category"/>
  <ChartPrimaryYAxis Title="Temperature (Fahrenheit)" LabelFormat="{value}°F"
  Minimum="0" Maximum="90" Interval="20"/>
  <ChartColumns>
    <ChartColumn Width="50%"/>
    <ChartColumn Width="50%"/>
  </ChartColumns>
  <ChartAxes>
    <ChartAxis Interval="2" OpposedPosition="true" ColumnIndex="1" Name="XAxis"
    ValueType="Syncfusion.Blazor.Charts.ValueType.Category"/>
  </ChartAxes>
  <ChartSeriesCollection>
    <ChartSeries DataSource="@WeatherReports" XName="X" YName="Y"
    Type="ChartSeriesType.Column"/>
    <ChartSeries DataSource="@WeatherReports" XName="X" YName="Y1"
    XAxisName="XAxis" />
  </ChartSeriesCollection>
</SfChart>

@code{
public class ChartData
{
  public string X { get; set; }
  public double Y { get; set; }
}
```

```

public double Y1 { get; set; }
}
public List<ChartData> WeatherReports = new List<ChartData>
{
    new ChartData{ X= "Jan", Y= 15, Y1= 33 },
    new ChartData{ X= "Feb", Y= 20, Y1= 31 },
    new ChartData{ X= "Mar", Y= 35, Y1= 30 },
    new ChartData{ X= "Apr", Y= 40, Y1= 28 },
    new ChartData{ X= "May", Y= 80, Y1= 29 },
    new ChartData{ X= "Jun", Y= 70, Y1= 30 },
    new ChartData{ X= "Jul", Y= 65, Y1= 33 },
    new ChartData{ X= "Aug", Y= 55, Y1= 32 },
    new ChartData{ X= "Sep", Y= 50, Y1= 34 },
    new ChartData{ X= "Oct", Y= 30, Y1= 32 },
    new ChartData{ X= "Nov", Y= 35, Y1= 32 },
    new ChartData{ X= "Dec", Y= 35, Y1= 31 }
};
}

```



Refer to our [Blazor Charts](#) feature tour page for its groundbreaking feature representations and also explore our [Blazor Chart Example](#) to know various chart types and how to represent time-dependent data, showing trends at equal intervals.

See Also

- [Data label](#)
- [Tooltip](#)

- [Marker](#)

Chart Types

Line Chart in Blazor Charts Component

Line

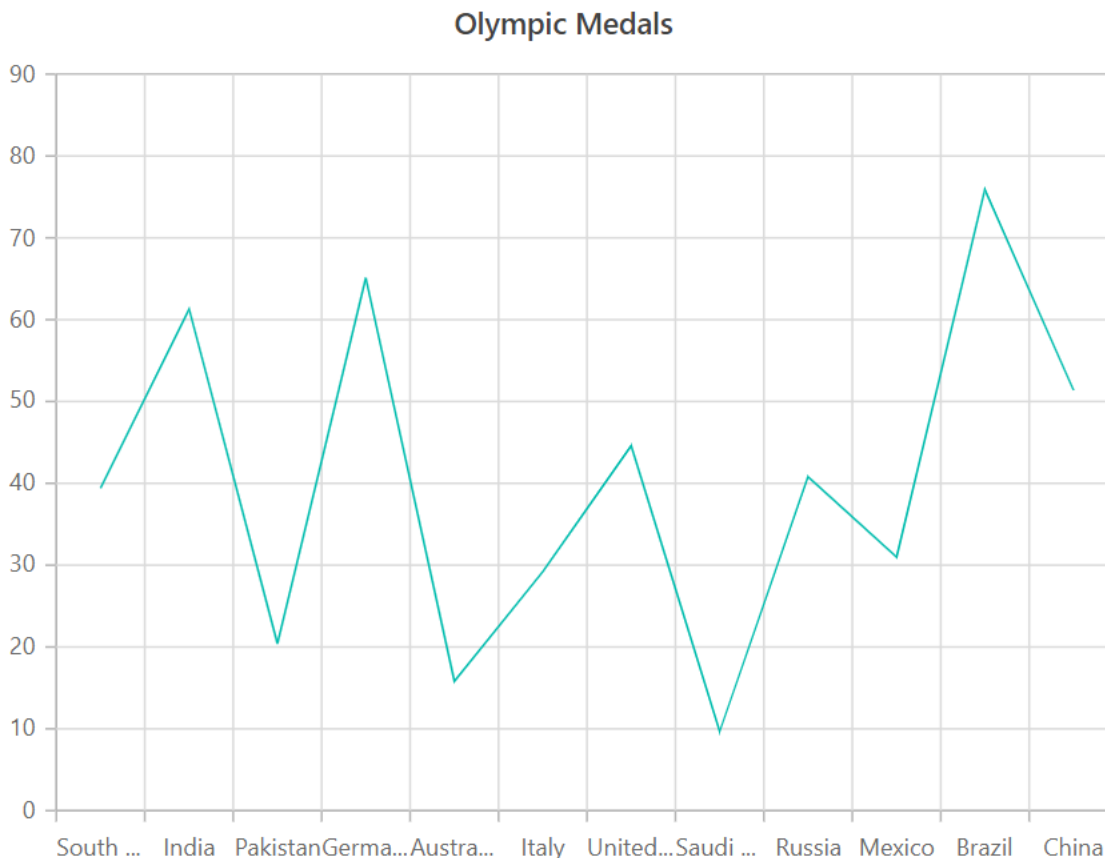
[Line Chart](#) represents and visualizes the time-dependent data to show the trends at equal intervals. It can be rendered by specifying the series [Type](#) to [Line](#).

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart Title="Olympic Medals">
  <ChartPrimaryXAxis
    ValueType="Syncfusion.Blazor.Charts.ValueType.Category"></ChartPrimaryXAxis>
  <ChartSeriesCollection>
    <ChartSeries DataSource="@MedalDetails" XName="X" YName="Y"
      Type="ChartSeriesType.Line">
    </ChartSeries>
  </ChartSeriesCollection>
</SfChart>

@code{
public class ChartData
{
    public string X { get; set; }
    public double Y { get; set; }
}

public List<ChartData> MedalDetails = new List<ChartData>
{
    new ChartData { X= "South Korea", Y= 39.4 },
    new ChartData { X= "India", Y= 61.3 },
    new ChartData { X= "Pakistan", Y= 20.4 },
    new ChartData { X= "Germany", Y= 65.1 },
    new ChartData { X= "Australia", Y= 15.8 },
    new ChartData { X= "Italy", Y= 29.2 },
    new ChartData { X= "United Kingdom", Y= 44.6 },
    new ChartData { X= "Saudi Arabia", Y= 9.7 },
    new ChartData { X= "Russia", Y= 40.8 },
    new ChartData { X= "Mexico", Y= 31 },
    new ChartData { X= "Brazil", Y= 75.9 },
    new ChartData { X= "China", Y= 51.4 }
};
}
```



Refer to our [Blazor Line Chart](#) feature tour page to know about its other groundbreaking feature representations. Explore our [Blazor Line Chart Example](#) to know how to represent time-dependent data, showing trends at equal intervals.

Multicolored Line

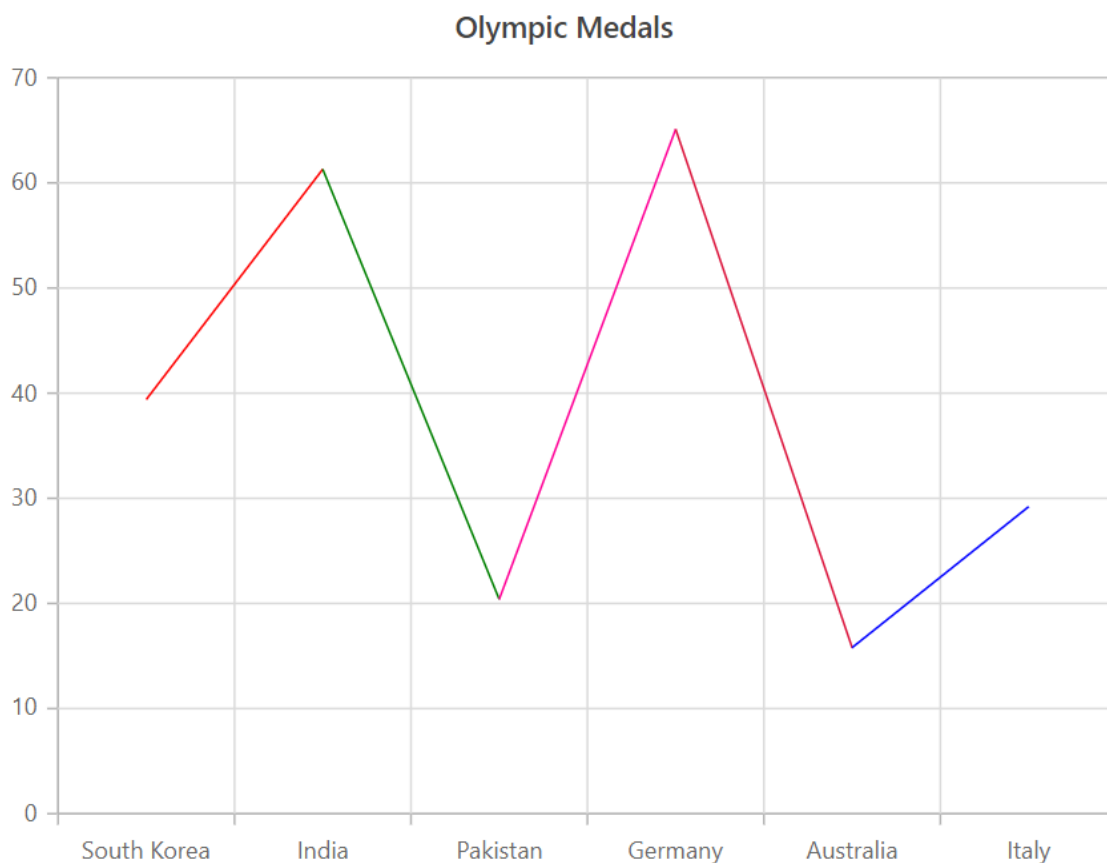
To render a multicolored line series, specify the `Type` property as `MultiColoredLine` in `ChartSeries`. Here, the individual colors of the segment can be mapped by using `PointColorMapping` property in `ChartSeries`.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart Title="Olympic Medals">
  <ChartPrimaryXAxis
    ValueType="Syncfusion.Blazor.Charts.ValueType.Category"></ChartPrimaryXAxis>
  <ChartSeriesCollection>
    <ChartSeries DataSource="@MedalDetails" XName="X" YName="Y"
      PointColorMapping="Color" Type="ChartSeriesType.MultiColoredLine">
    </ChartSeries>
  </ChartSeriesCollection>
</SfChart>

@code{
public class ChartData
{
    public string X { get; set; }
    public double Y { get; set; }
}
```

```
public string Color { get; set; }  
}  
public List<ChartData> MedalDetails = new List<ChartData>  
{  
    new ChartData { X= "South Korea", Y= 39.4, Color="red" },  
    new ChartData { X= "India", Y= 61.3, Color="green" },  
    new ChartData { X= "Pakistan", Y= 20.4, Color="#ff0097" },  
    new ChartData { X= "Germany", Y= 65.1, Color="crimson" },  
    new ChartData { X= "Australia", Y= 15.8, Color="blue" },  
    new ChartData { X= "Italy", Y= 29.2, Color="darkorange" },  
};  
}
```



Series Customization

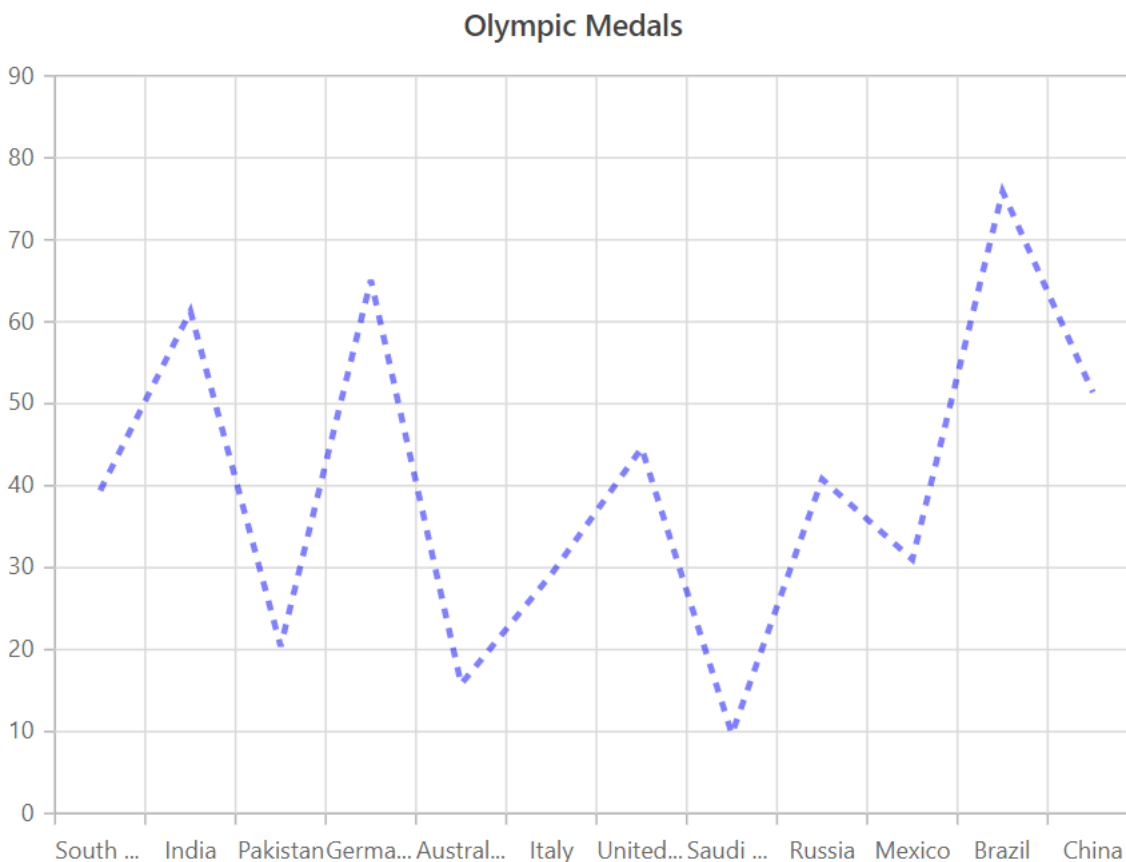
The following properties can be used to customize the [Line](#) series.

- [Fill](#) – Specifies the color of the series.
- [Opacity](#) – Specifies the opacity of [Fill](#).
- [DashArray](#) – Specifies the dashes for series.
- [Width](#) – Specifies the width for series.

ASPX-CS


```
@using Syncfusion.Blazor.Charts
<SfChart Title="Olympic Medals">
  <ChartPrimaryXAxis
    ValueType="Syncfusion.Blazor.Charts.ValueType.Category"></ChartPrimaryXAxis>
  <ChartSeriesCollection>
    <ChartSeries DataSource="@MedalDetails" XName="X" YName="Y" Width="3"
      Opacity="0.5"
      DashArray="5,5" Fill="blue" Type="ChartSeriesType.Line">
    </ChartSeries>
  </ChartSeriesCollection>
</SfChart>

@code{
public class ChartData
{
public string X { get; set; }
public double Y { get; set; }
}
public List<ChartData> MedalDetails = new List<ChartData>
{
new ChartData { X= "South Korea", Y= 39.4 },
new ChartData { X= "India", Y= 61.3 },
new ChartData { X= "Pakistan", Y= 20.4 },
new ChartData { X= "Germany", Y= 65.1 },
new ChartData { X= "Australia", Y= 15.8 },
new ChartData { X= "Italy", Y= 29.2 },
new ChartData { X= "United Kingdom", Y= 44.6 },
new ChartData { X= "Saudi Arabia", Y= 9.7 },
new ChartData { X= "Russia", Y= 40.8 },
new ChartData { X= "Mexico", Y= 31 },
new ChartData { X= "Brazil", Y= 75.9 },
new ChartData { X= "China", Y= 51.4 }
};
}
```



Refer to our [Blazor Charts](#) feature tour page for its groundbreaking feature representations and also explore our [Blazor Chart Example](#) to know various chart types and how to represent time-dependent data, showing trends at equal intervals.

See Also

- [Data Label](#)
- [Tooltip](#)

Step Line Chart in Blazor Charts Component

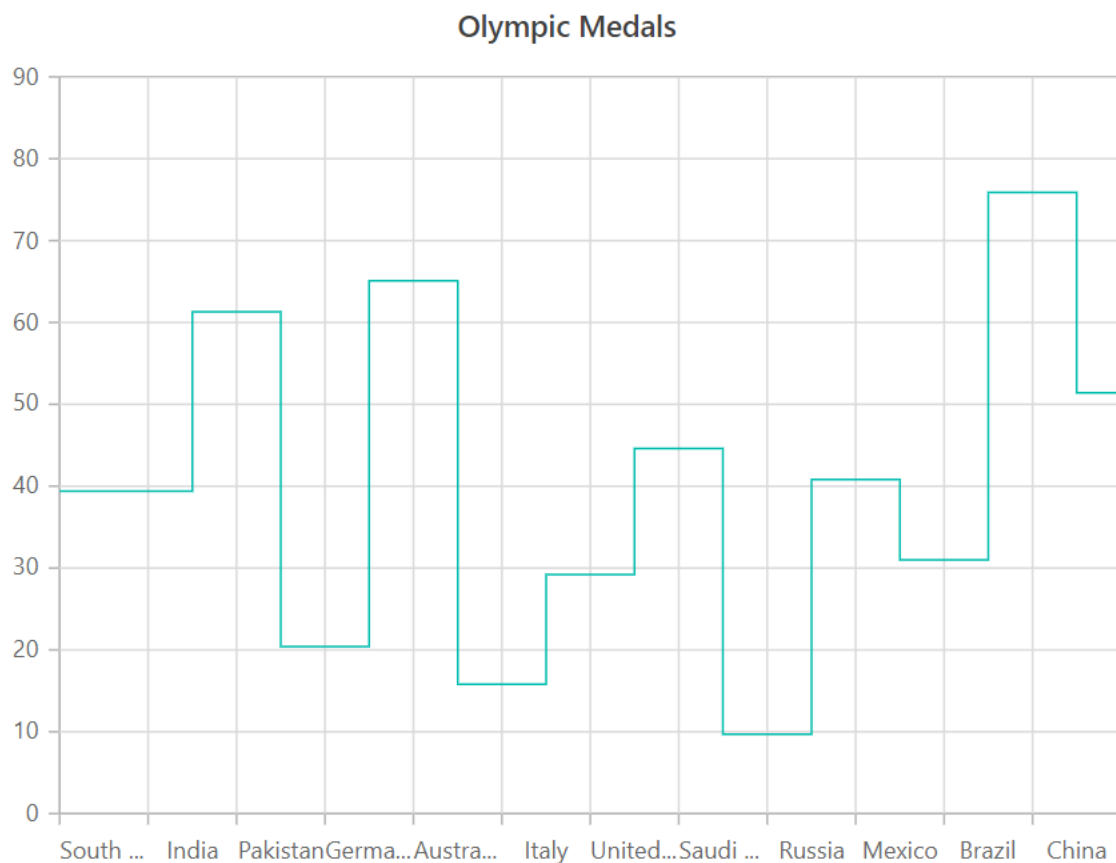
Step Line

[Step Line Chart](#) represents a set of points connected by horizontal and vertical lines. To render a step line series, set the series [Type](#) as [StepLine](#).

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart Title="Olympic Medals">
  <ChartPrimaryXAxis
    ValueType="Syncfusion.Blazor.Charts.ValueType.Category"></ChartPrimaryXAxis>
  <ChartSeriesCollection>
    <ChartSeries DataSource="@MedalDetails" XName="X" YName="Y"
      Type="ChartSeriesType.StepLine">
    </ChartSeries>
  </ChartSeriesCollection>
```

```
</SfChart>
@code{
public class ChartData
{
public string X { get; set; }
public double Y { get; set; }
}
public List<ChartData> MedalDetails = new List<ChartData>
{
new ChartData { X= "South Korea", Y= 39.4 },
new ChartData { X= "India", Y= 61.3 },
new ChartData { X= "Pakistan", Y= 20.4 },
new ChartData { X= "Germany", Y= 65.1 },
new ChartData { X= "Australia", Y= 15.8 },
new ChartData { X= "Italy", Y= 29.2 },
new ChartData { X= "United Kingdom", Y= 44.6 },
new ChartData { X= "Saudi Arabia", Y= 9.7 },
new ChartData { X= "Russia", Y= 40.8 },
new ChartData { X= "Mexico", Y= 31 },
new ChartData { X= "Brazil", Y= 75.9 },
new ChartData { X= "China", Y= 51.4 }
};
}
```



Refer to our [Blazor Step Line Chart](#) feature tour page to know about its other groundbreaking feature representations. Explore our [Blazor Step Line Chart Example](#) to know how to render and configure the Step Line type chart.

Series Customization

The following properties can be used to customize the [Step Line](#) series.

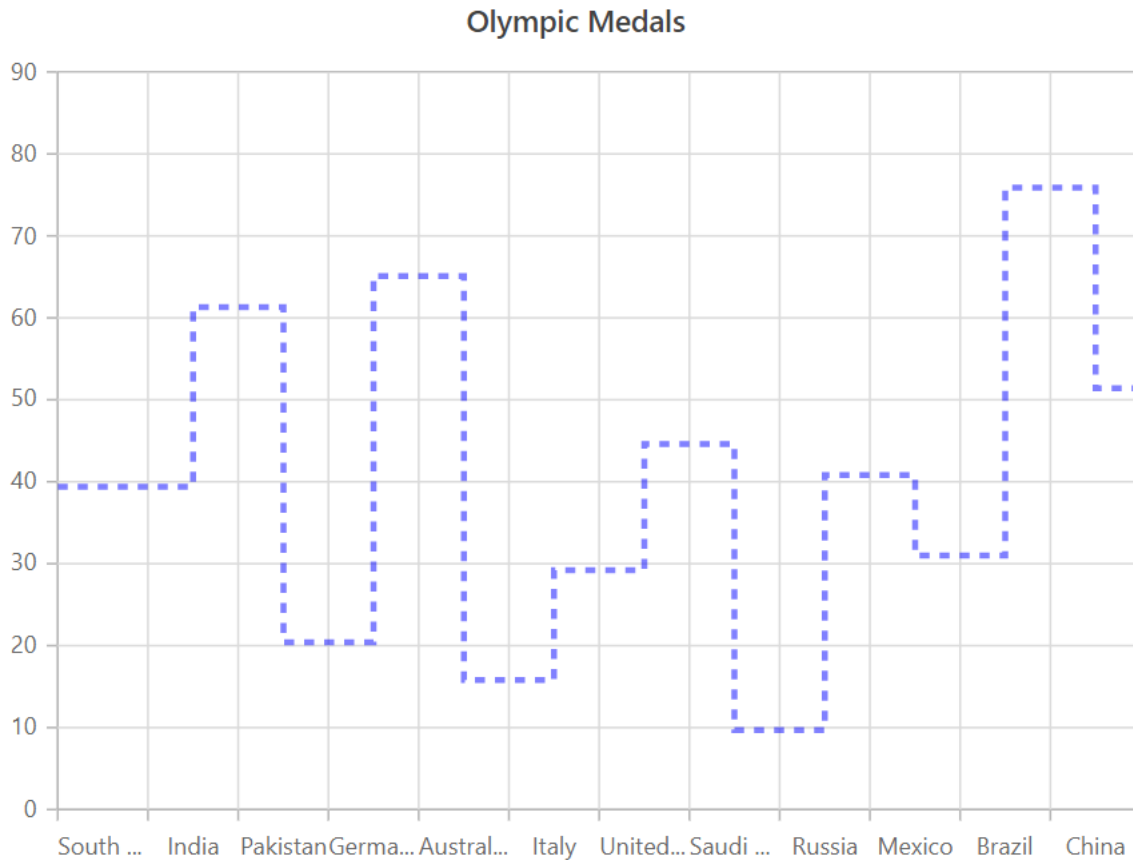
- [Fill](#) – Specifies the color of the series.
- [Opacity](#) – Specifies the opacity of [Fill](#).
- [Width](#) – Specifies the width of the line stroke.
- [DashArray](#) – Specifies the dashes of line stroke.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart Title="Olympic Medals" Width="60%">
  <ChartPrimaryXAxis
    ValueType="Syncfusion.Blazor.Charts.ValueType.Category"></ChartPrimaryXAxis>
  <ChartSeriesCollection>
    <ChartSeries DataSource="@MedalDetails" XName="X" YName="Y" Width="3"
      Opacity="0.5"
      DashArray="5,5" Fill="blue" Type="ChartSeriesType.StepLine">
    </ChartSeries>
  </ChartSeriesCollection>
</SfChart>

@code{
public class ChartData
{
    public string X;
    public double Y;
}

public List<ChartData> MedalDetails = new List<ChartData>
{
    new ChartData { X= "South Korea", Y= 39.4 },
    new ChartData { X= "India", Y= 61.3 },
    new ChartData { X= "Pakistan", Y= 20.4 },
    new ChartData { X= "Germany", Y= 65.1 },
    new ChartData { X= "Australia", Y= 15.8 },
    new ChartData { X= "Italy", Y= 29.2 },
    new ChartData { X= "United Kingdom", Y= 44.6 },
    new ChartData { X= "Saudi Arabia", Y= 9.7 },
    new ChartData { X= "Russia", Y= 40.8 },
    new ChartData { X= "Mexico", Y= 31 },
    new ChartData { X= "Brazil", Y= 75.9 },
    new ChartData { X= "China", Y= 51.4 }
};
}
```



Refer to our [Blazor Charts](#) feature tour page for its groundbreaking feature representations and also explore our [Blazor Chart Example](#) to know various chart types and how to represent time-dependent data, showing trends at equal intervals.

See Also

- [Data Label](#)
- [Tooltip](#)

Stacked Line Chart in Blazor Charts Component

Stacked Line

[Stacked Line Chart](#) is a chart with Y values stacked over one another in the series order. It shows the relation between individual values to the total sum of the points. To render a [Stacked Line](#) series, set the series [Type](#) as [StackingLine](#).

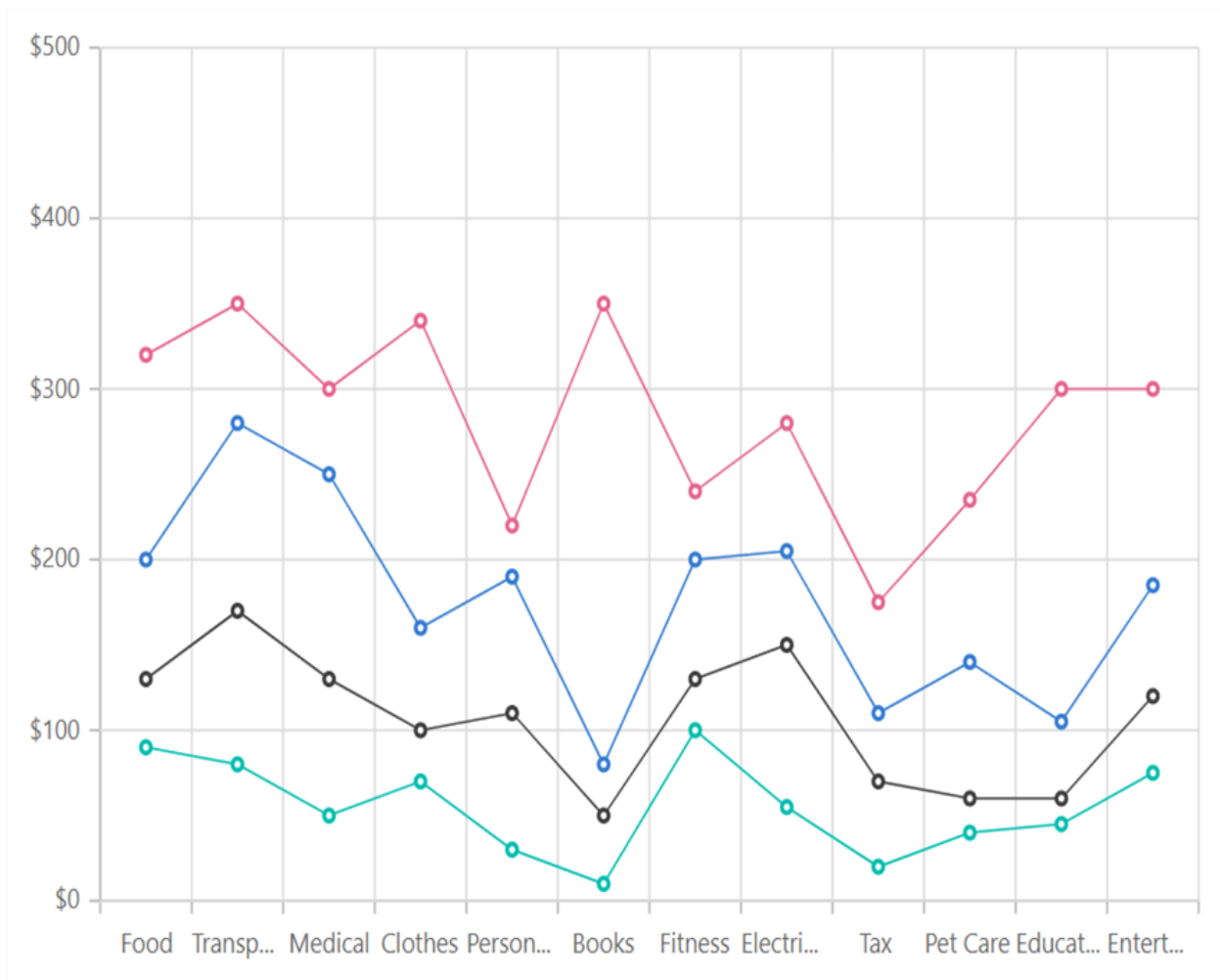
ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart Title="Family Expense for Month">
  <ChartPrimaryXAxis ValueType="Syncfusion.Blazor.Charts.ValueType.Category"
    Interval="1">
  </ChartPrimaryXAxis>
  <ChartPrimaryYAxis Title="Expense" Interval="100" LabelFormat="{value}">
  </ChartPrimaryYAxis>
```

```

<ChartArea>
<ChartAreaBorder Width="0"></ChartAreaBorder>
</ChartArea>
<ChartSeriesCollection>
<ChartSeries XName="X" DataSource="@ExpenseReports"
YName="Y" Type="ChartSeriesType.StackingLine">
<ChartMarker Visible="true">
</ChartMarker>
</ChartSeries>
<ChartSeries XName="X" DataSource="@ExpenseReports"
YName="Y1" Type="ChartSeriesType.StackingLine">
<ChartMarker Visible="true">
</ChartMarker>
</ChartSeries>
<ChartSeries XName="X" DataSource="@ExpenseReports"
YName="Y2" Type="ChartSeriesType.StackingLine">
<ChartMarker Visible="true">
</ChartMarker>
</ChartSeries>
<ChartSeries XName="X" DataSource="@ExpenseReports"
YName="Y3" Type="ChartSeriesType.StackingLine">
<ChartMarker Visible="true">
</ChartMarker>
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code{
public class ChartData
{
public string X { get; set; }
public double Y { get; set; }
public double Y1 { get; set; }
public double Y2 { get; set; }
public double Y3 { get; set; }
}
public List<ChartData> ExpenseReports = new List<ChartData>
{
new ChartData { X = "Food" , Y = 90, Y1 = 40 , Y2= 70, Y3= 120},
new ChartData { X = "Transport", Y = 80, Y1 = 90, Y2= 110, Y3= 70 },
new ChartData { X = "Medical",Y = 50, Y1 = 80, Y2= 120, Y3= 50 },
new ChartData { X = "Clothes",Y = 70, Y1 = 30, Y2= 60, Y3= 180 },
new ChartData { X = "Personal Care", Y = 30, Y1 = 80, Y2= 80, Y3= 30 },
new ChartData { X = "Books", Y = 10, Y1 = 40, Y2= 30, Y3= 270},
new ChartData { X = "Fitness",Y = 100, Y1 = 30, Y2= 70, Y3= 40 },
new ChartData { X = "Electricity", Y = 55, Y1 = 95, Y2= 55, Y3= 75},
new ChartData { X = "Tax", Y = 20, Y1 = 50, Y2= 40, Y3= 65 },
new ChartData { X = "Pet Care", Y = 40, Y1 = 20, Y2= 80, Y3= 95 },
new ChartData { X = "Education", Y = 45, Y1 = 15, Y2= 45, Y3= 195 },
new ChartData { X = "Entertainment", Y = 75, Y1 = 45, Y2= 65, Y3= 115 }
};
}

```



Refer to our [Blazor Stacked Line Chart](#) feature tour page to know about its other groundbreaking feature representations. Explore our [Blazor Stacked Line Chart Example](#) to know how to render and configure the Stacked Line type chart.

Series Customization

The following properties can be used to customize the [Stacked Line](#) series.

- [Fill](#) – Specifies the color of the series.
- [Opacity](#) – Specifies the opacity of [Fill](#).
- [Width](#) – Specifies the width of the line stroke.
- [DashArray](#) – Specifies the dashes of line stroke.

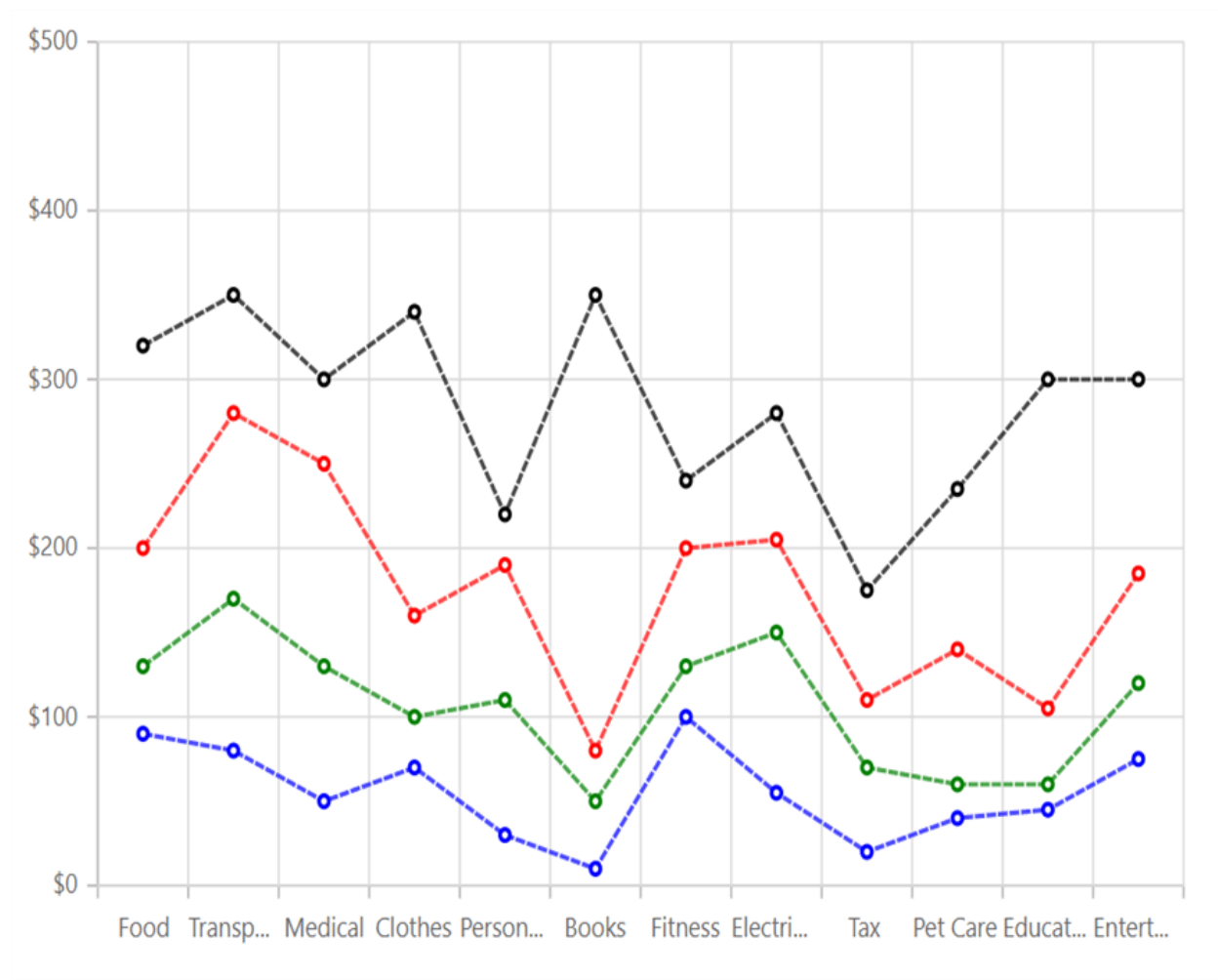
ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart Title="Family Expense for Month">
  <ChartPrimaryXAxis ValueTypes="Syncfusion.Blazor.Charts.ValueType.Category"
    Interval="1">
  </ChartPrimaryXAxis>
  <ChartPrimaryYAxis Title="Expense" Interval="100" LabelFormat="{value}">
  </ChartPrimaryYAxis>
  <ChartArea>
```

```

<ChartAreaBorder Width="0"></ChartAreaBorder>
</ChartArea>
<ChartSeriesCollection>
<ChartSeries XName="X" Width="2" DashArray="5,1"
DataSource="@ExpenseReports"
YName="Y" Fill="blue" Opacity="0.7" Type="ChartSeriesType.StackingLine">
<ChartMarker Visible="true">
</ChartMarker>
</ChartSeries>
<ChartSeries XName="X" Width="2" DashArray="5,1"
DataSource="@ExpenseReports"
YName="Y1" Fill="green" Opacity="0.7" Type="ChartSeriesType.StackingLine">
<ChartMarker Visible="true">
</ChartMarker>
</ChartSeries>
<ChartSeries XName="X" Width="2" DashArray="5,1"
DataSource="@ExpenseReports"
YName="Y2" Fill="red" Opacity="0.7" Type="ChartSeriesType.StackingLine">
<ChartMarker Visible="true">
</ChartMarker>
</ChartSeries>
<ChartSeries XName="X" Width="2" DashArray="5,1"
DataSource="@ExpenseReports"
YName="Y3" Fill="black" Opacity="0.7" Type="ChartSeriesType.StackingLine">
<ChartMarker Visible="true">
</ChartMarker>
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code{
public class ChartData
{
public string X { get; set; }
public double Y { get; set; }
public double Y1 { get; set; }
public double Y2 { get; set; }
public double Y3 { get; set; }
}
public List<ChartData> ExpenseReports = new List<ChartData>
{
new ChartData { X = "Food" , Y = 90, Y1 = 40 , Y2= 70, Y3= 120},
new ChartData { X = "Transport", Y = 80, Y1 = 90, Y2= 110, Y3= 70 },
new ChartData { X = "Medical",Y = 50, Y1 = 80, Y2= 120, Y3= 50 },
new ChartData { X = "Clothes",Y = 70, Y1 = 30, Y2= 60, Y3= 180 },
new ChartData { X = "Personal Care", Y = 30, Y1 = 80, Y2= 80, Y3= 30 },
new ChartData { X = "Books", Y = 10, Y1 = 40, Y2= 30, Y3= 270},
new ChartData { X = "Fitness",Y = 100, Y1 = 30, Y2= 70, Y3= 40 },
new ChartData { X = "Electricity", Y = 55, Y1 = 95, Y2= 55, Y3= 75},
new ChartData { X = "Tax", Y = 20, Y1 = 50, Y2= 40, Y3= 65 },
new ChartData { X = "Pet Care", Y = 40, Y1 = 20, Y2= 80, Y3= 95 },
new ChartData { X = "Education", Y = 45, Y1 = 15, Y2= 45, Y3= 195 },
new ChartData { X = "Entertainment", Y = 75, Y1 = 45, Y2= 65, Y3= 115 }
};
}

```

Refer to our [Blazor Charts](#) feature tour page for its groundbreaking feature representations and also explore our [Blazor Chart Example](#) to know various chart types and how to represent time-dependent data, showing trends at equal intervals.

See Also

- [Data Label](#)
- [Tooltip](#)

100% Stacked Line in Blazor Charts Component

100% Stacked Line

[100% Stacked Line Chart](#) displays multiple series of data as stacked lines, ensuring that the cumulative proportion of each stacked element always totals 100%. Hence, the y-axis will always be rendered with the range 0–100%. To render a [100% Stacked Line](#) series, set the series [Type](#) as [StackingLine100](#).

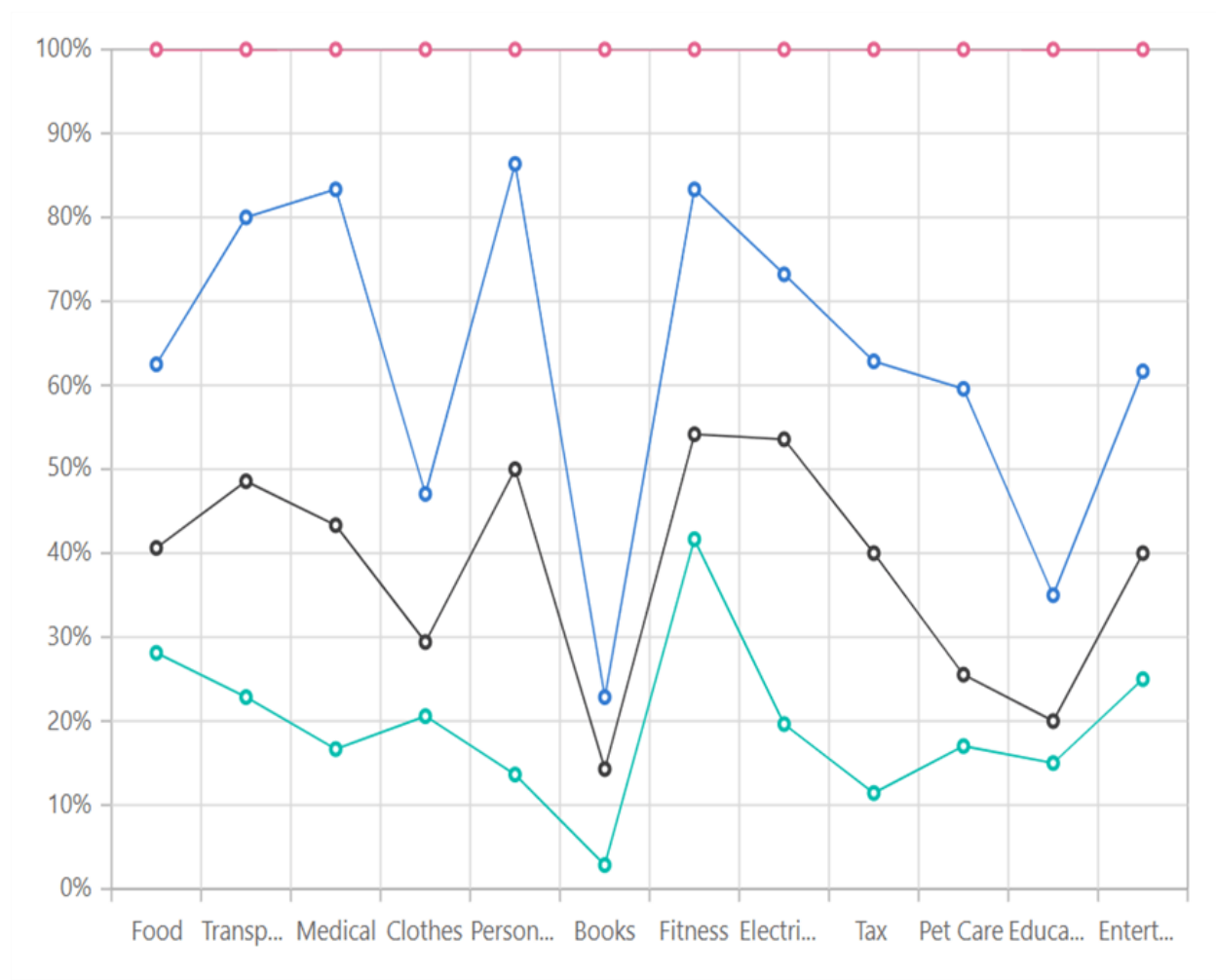
ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart Title="Family Expense for Month">
  <ChartPrimaryXAxis ValueType="Syncfusion.Blazor.Charts.ValueType.Category">
  </ChartPrimaryXAxis>
  <ChartSeriesCollection>
```

```

<ChartSeries XName="X" Width="2" DashArray="5,1"
DataSource="@ExpenseReports"
YName="Y" Type="ChartSeriesType.StackingLine100">
<ChartMarker Visible="true">
</ChartMarker>
</ChartSeries>
<ChartSeries XName="X" Width="2" DashArray="5,1"
DataSource="@ExpenseReports"
YName="Y1" Type="ChartSeriesType.StackingLine100">
<ChartMarker Visible="true">
</ChartMarker>
</ChartSeries>
<ChartSeries XName="X" Width="2" DashArray="5,1"
DataSource="@ExpenseReports"
YName="Y2" Type="ChartSeriesType.StackingLine100">
<ChartMarker Visible="true">
</ChartMarker>
</ChartSeries>
<ChartSeries XName="X" Width="2" DashArray="5,1"
DataSource="@ExpenseReports"
YName="Y3" Type="ChartSeriesType.StackingLine100">
<ChartMarker Visible="true">
</ChartMarker>
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code{
public class ChartData
{
public string X { get; set; }
public double Y { get; set; }
public double Y1 { get; set; }
public double Y2 { get; set; }
public double Y3 { get; set; }
}
public List<ChartData> ExpenseReports = new List<ChartData>
{
new ChartData { X = "Food" , Y = 90, Y1 = 40 , Y2= 70, Y3= 120},
new ChartData { X = "Transport", Y = 80, Y1 = 90, Y2= 110, Y3= 70 },
new ChartData { X = "Medical",Y = 50, Y1 = 80, Y2= 120, Y3= 50 },
new ChartData { X = "Clothes",Y = 70, Y1 = 30, Y2= 60, Y3= 180 },
new ChartData { X = "Personal Care", Y = 30, Y1 = 80, Y2= 80, Y3= 30 },
new ChartData { X = "Books", Y = 10, Y1 = 40, Y2= 30, Y3= 270},
new ChartData { X = "Fitness",Y = 100, Y1 = 30, Y2= 70, Y3= 40 },
new ChartData { X = "Electricity", Y = 55, Y1 = 95, Y2= 55, Y3= 75},
new ChartData { X = "Tax", Y = 20, Y1 = 50, Y2= 40, Y3= 65 },
new ChartData { X = "Pet Care", Y = 40, Y1 = 20, Y2= 80, Y3= 95 },
new ChartData { X = "Education", Y = 45, Y1 = 15, Y2= 45, Y3= 195 },
new ChartData { X = "Entertainment", Y = 75, Y1 = 45, Y2= 65, Y3= 115 }
};
}

```



Refer to our [Blazor 100% Stacked Line Chart](#) feature tour page to know about its other groundbreaking feature representations. Explore our [Blazor 100% Stacked Line Chart Example](#) to know how to render and configure the 100% Stacked Line type chart.

Series Customization

The following properties can be used to customize the [100% Stacked Line](#) series.

- [Fill](#) – Specifies the color of the series.
- [Opacity](#) – Specifies the opacity of [Fill](#).
- [Width](#) – Specifies the width of the line stroke.
- [DashArray](#) – Specifies the dashes of line stroke.

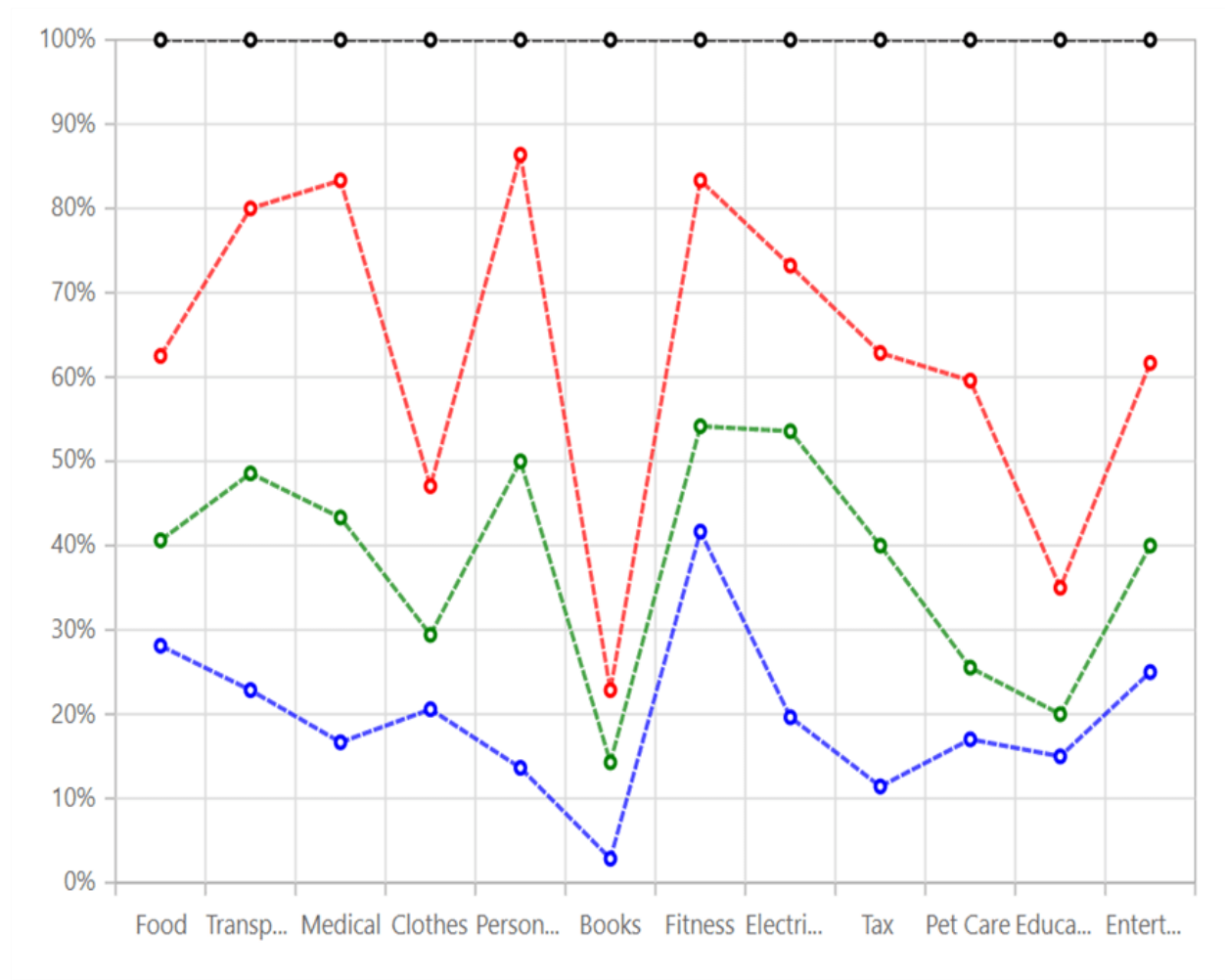
ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart Title="Family Expense for Month">
  <ChartPrimaryXAxis ValueType="Syncfusion.Blazor.Charts.ValueType.Category">
  </ChartPrimaryXAxis>
  <ChartSeriesCollection>
    <ChartSeries XName="X" Width="2" DashArray="5,1"
      DataSource="@ExpenseReports"
      YName="Y" Fill="blue" Opacity="0.7" Type="ChartSeriesType.StackingLine100">
```

```

<ChartMarker Visible="true">
</ChartMarker>
</ChartSeries>
<ChartSeries XName="X" Width="2" DashArray="5,1"
DataSource="@ExpenseReports"
YName="Y1" Fill="green" Opacity="0.7"
Type="ChartSeriesType.StackingLine100">
<ChartMarker Visible="true">
</ChartMarker>
</ChartSeries>
<ChartSeries XName="X" Width="2" DashArray="5,1"
DataSource="@ExpenseReports"
YName="Y2" Fill="red" Opacity="0.7" Type="ChartSeriesType.StackingLine100">
<ChartMarker Visible="true">
</ChartMarker>
</ChartSeries>
<ChartSeries XName="X" Width="2" DashArray="5,1"
DataSource="@ExpenseReports"
YName="Y3" Fill="black" Opacity="0.7"
Type="ChartSeriesType.StackingLine100">
<ChartMarker Visible="true">
</ChartMarker>
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code{
public class ChartData
{
    public string X { get; set; }
    public double Y { get; set; }
    public double Y1 { get; set; }
    public double Y2 { get; set; }
    public double Y3 { get; set; }
}
public List<ChartData> ExpenseReports = new List<ChartData>
{
    new ChartData { X = "Food" , Y = 90, Y1 = 40 , Y2= 70, Y3= 120},
    new ChartData { X = "Transport", Y = 80, Y1 = 90, Y2= 110, Y3= 70 },
    new ChartData { X = "Medical",Y = 50, Y1 = 80, Y2= 120, Y3= 50 },
    new ChartData { X = "Clothes",Y = 70, Y1 = 30, Y2= 60, Y3= 180 },
    new ChartData { X = "Personal Care", Y = 30, Y1 = 80, Y2= 80, Y3= 30 },
    new ChartData { X = "Books",Y = 10, Y1 = 40, Y2= 30, Y3= 270},
    new ChartData { X = "Fitness",Y = 100, Y1 = 30, Y2= 70, Y3= 40 },
    new ChartData { X = "Electricity", Y = 55, Y1 = 95, Y2= 55, Y3= 75},
    new ChartData { X = "Tax", Y = 20, Y1 = 50, Y2= 40, Y3= 65 },
    new ChartData { X = "Pet Care", Y = 40, Y1 = 20, Y2= 80, Y3= 95 },
    new ChartData { X = "Education", Y = 45, Y1 = 15, Y2= 45, Y3= 195 },
    new ChartData { X = "Entertainment", Y = 75, Y1 = 45, Y2= 65, Y3= 115 }
};
}

```



Refer to our [Blazor Charts](#) feature tour page for its groundbreaking feature representations and also explore our [Blazor Chart Example](#) to know various chart types and how to represent time-dependent data, showing trends at equal intervals.

See Also

- [Data Label](#)
- [Tooltip](#)

Spline in Blazor Charts Component

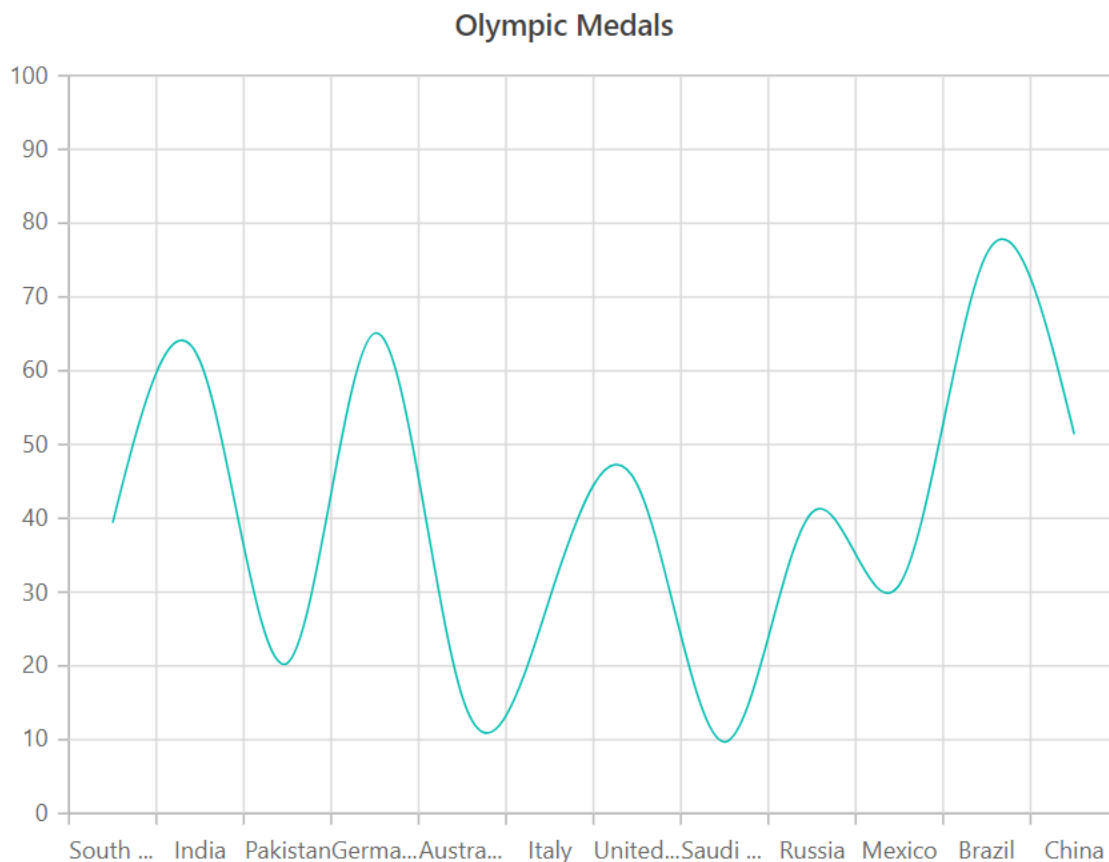
Spline

[Spline Chart](#) connects the data points with smooth curves. To render a [Spline Chart](#), set the series [Type](#) as [Spline](#).

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart Title="Olympic Medals">
  <ChartPrimaryXAxis
    ValueType="Syncfusion.Blazor.Charts.ValueType.Category"></ChartPrimaryXAxis>
  <ChartSeriesCollection>
```

```
<ChartSeries DataSource="@MedalDetails" XName="X" YName="Y"
Type="ChartSeriesType.Spline">
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code{
public class ChartData
{
public string X { get; set; }
public double Y { get; set; }
}
public List<ChartData> MedalDetails = new List<ChartData>
{
new ChartData { X= "South Korea", Y= 39.4 },
new ChartData { X= "India", Y= 61.3 },
new ChartData { X= "Pakistan", Y= 20.4 },
new ChartData { X= "Germany", Y= 65.1 },
new ChartData { X= "Australia", Y= 15.8 },
new ChartData { X= "Italy", Y= 29.2 },
new ChartData { X= "United Kingdom", Y= 44.6 },
new ChartData { X= "Saudi Arabia", Y= 9.7 },
new ChartData { X= "Russia", Y= 40.8 },
new ChartData { X= "Mexico", Y= 31 },
new ChartData { X= "Brazil", Y= 75.9 },
new ChartData { X= "China", Y= 51.4 }
};
}
```



Refer to our [Blazor Spline Chart](#) feature tour page to know about its other groundbreaking feature representations. Explore our [Blazor Spline Chart Example](#) to know how to connect the data points with smooth curves.

Type of spline

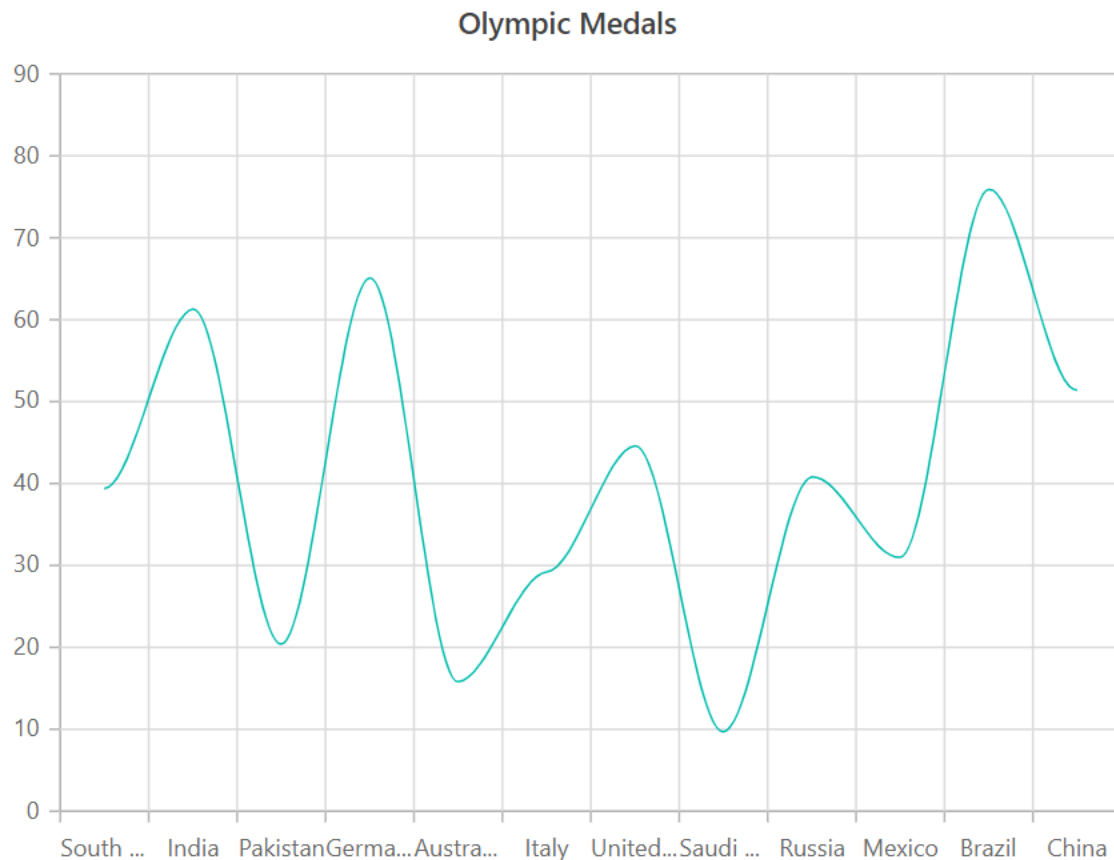
To specify the type of [Spline Chart](#) use [SplineType](#) property. The spline types are **Clamped**, **Cardinal**, **Monotonic** and **Natural**.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart Title="Olympic Medals" Width="60%">
  <ChartPrimaryXAxis
    ValueType="Syncfusion.Blazor.Charts.ValueType.Category"></ChartPrimaryXAxis>
  <ChartSeriesCollection>
    <ChartSeries DataSource="@MedalDetails" XName="X" YName="Y"
      Type="ChartSeriesType.Spline" SplineType="SplineType.Cardinal">
    </ChartSeries>
  </ChartSeriesCollection>
</SfChart>

@code{
public class ChartData
{
  public string X;
  public double Y;
}
```

```
public List<ChartData> MedalDetails = new List<ChartData>
{
    new ChartData { X= "South Korea", Y= 39.4 },
    new ChartData { X= "India", Y= 61.3 },
    new ChartData { X= "Pakistan", Y= 20.4 },
    new ChartData { X= "Germany", Y= 65.1 },
    new ChartData { X= "Australia", Y= 15.8 },
    new ChartData { X= "Italy", Y= 29.2 },
    new ChartData { X= "United Kingdom", Y= 44.6 },
    new ChartData { X= "Saudi Arabia", Y= 9.7 },
    new ChartData { X= "Russia", Y= 40.8 },
    new ChartData { X= "Mexico", Y= 31 },
    new ChartData { X= "Brazil", Y= 75.9 },
    new ChartData { X= "China", Y= 51.4 }
};
}
```



Series Customization

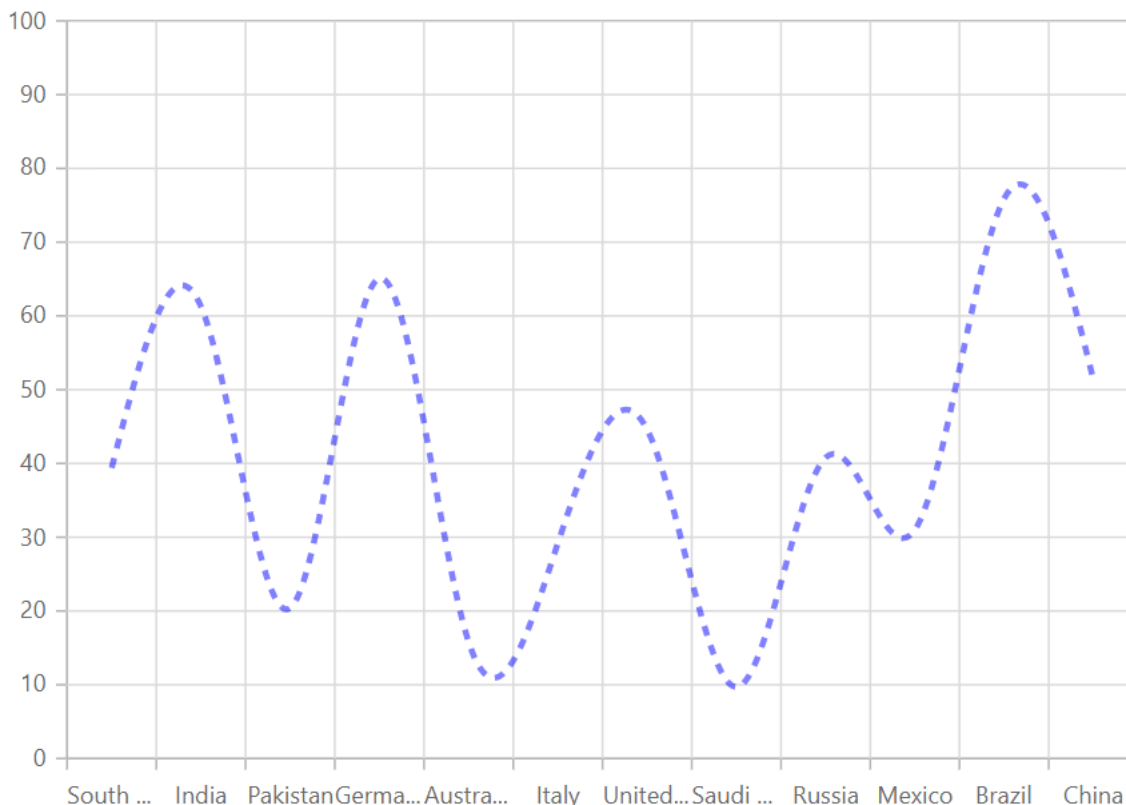
The following properties can be used to customize the [Spline](#) series.

- [Fill](#) – Specifies the color of the series.
- [Opacity](#) – Specifies the opacity of [Fill](#).
- [Width](#) – Specifies the width of the line stroke.
- [DashArray](#) – Specifies the dashes of line stroke.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart Title="Olympic Medals" Width="60%">
  <ChartPrimaryXAxis
    ValueType="Syncfusion.Blazor.Charts.ValueType.Category"></ChartPrimaryXAxis>
  <ChartSeriesCollection>
    <ChartSeries DataSource="@MedalDetails" XName="X" YName="Y" Width="3"
      Opacity="0.5"
      DashArray="5,5" Fill="blue" Type="ChartSeriesType.Spline">
    </ChartSeries>
  </ChartSeriesCollection>
</SfChart>

@code{
public class ChartData
{
public string X;
public double Y;
}
public List<ChartData> MedalDetails = new List<ChartData>
{
new ChartData { X= "South Korea", Y= 39.4 },
new ChartData { X= "India", Y= 61.3 },
new ChartData { X= "Pakistan", Y= 20.4 },
new ChartData { X= "Germany", Y= 65.1 },
new ChartData { X= "Australia", Y= 15.8 },
new ChartData { X= "Italy", Y= 29.2 },
new ChartData { X= "United Kingdom", Y= 44.6 },
new ChartData { X= "Saudi Arabia", Y= 9.7 },
new ChartData { X= "Russia", Y= 40.8 },
new ChartData { X= "Mexico", Y= 31 },
new ChartData { X= "Brazil", Y= 75.9 },
new ChartData { X= "China", Y= 51.4 }
};
}
```



Refer to our [Blazor Charts](#) feature tour page for its groundbreaking feature representations and also explore our [Blazor Chart Example](#) to know various chart types and how to represent time-dependent data, showing trends at equal intervals.

See Also

- [Data Label](#)
- [Tooltip](#)

Area in Blazor Charts Component

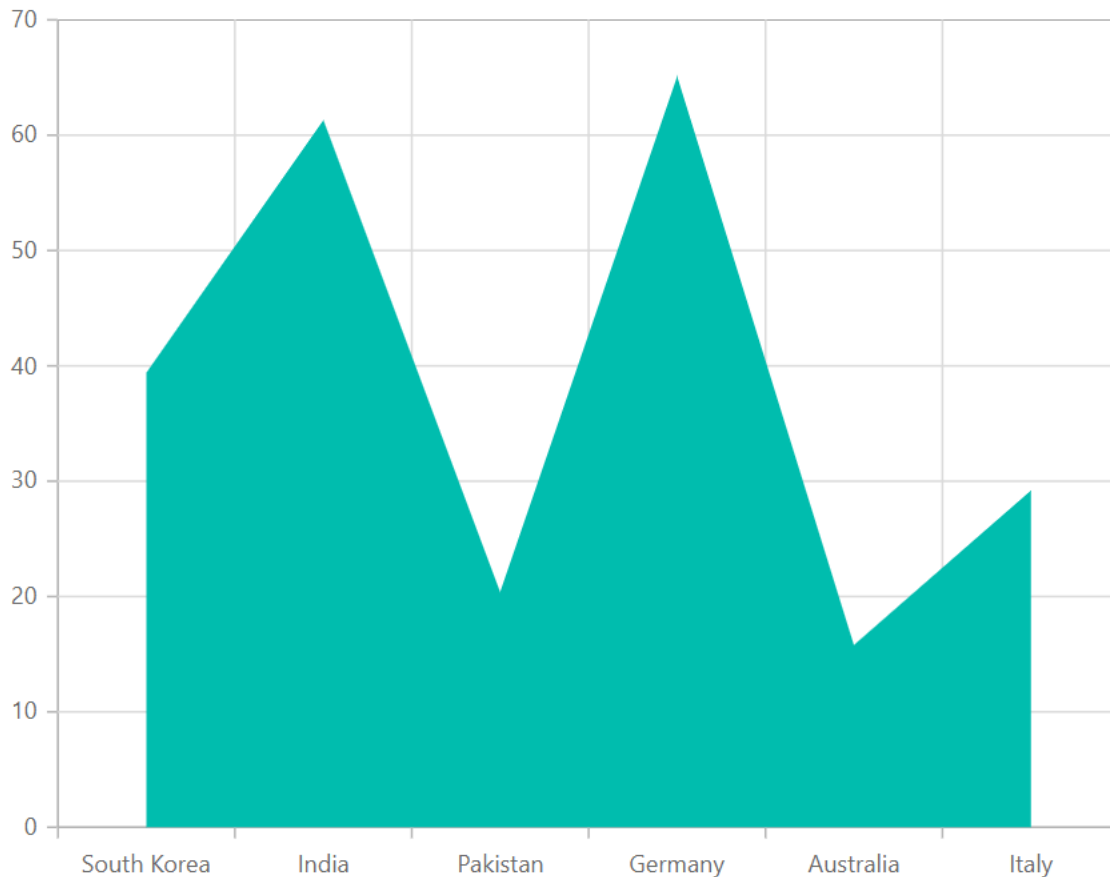
Area

[Area Chart](#) is like Line Chart, that represents time-dependent data and shows the trends at equal intervals, but it fills the area below the line. To render an area series, specify the [Type](#) property as [Area](#) and the [ValueType](#) of the plot data can be [Category](#), [DateTime](#), [DateTimeCategory](#), [Double](#) or [Logarithmic](#).

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart>
<ChartPrimaryXAxis ValueType="Syncfusion.Blazor.Charts.ValueType.Category"/>
<ChartSeriesCollection>
<ChartSeries DataSource="@MedalDetails" XName="X" YName="Y"
Type="ChartSeriesType.Area">
</ChartSeries>
</ChartSeriesCollection>
```

```
</SfChart>
@code{
public class ChartData
{
    public string X { get; set; }
    public double Y { get; set; }
}
public List<ChartData> MedalDetails = new List<ChartData>
{
    new ChartData { X= "South Korea", Y= 39.4 },
    new ChartData { X= "India", Y= 61.3 },
    new ChartData { X= "Pakistan", Y= 20.4 },
    new ChartData { X= "Germany", Y= 65.1 },
    new ChartData { X= "Australia", Y= 15.8 }
};
}
```



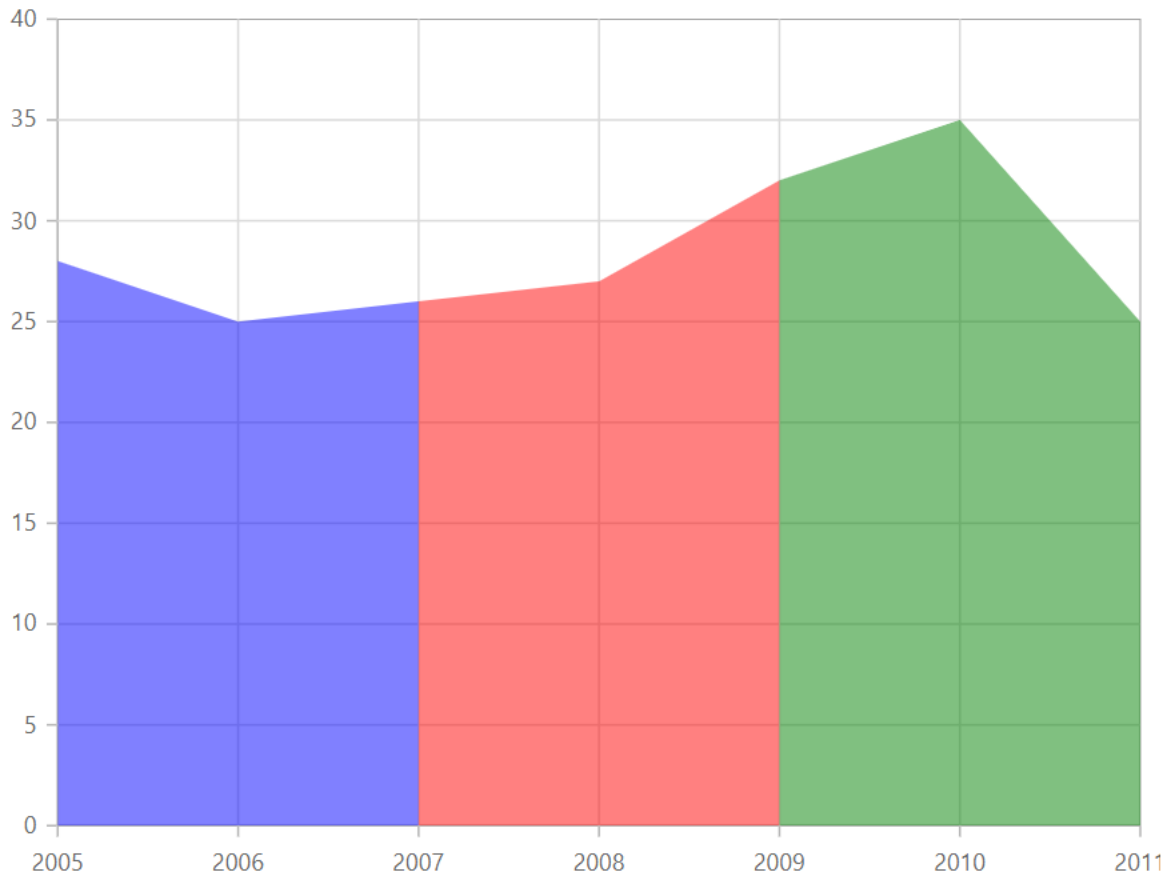
Refer to our [Blazor Area Charts](#) feature tour page to know about its other groundbreaking feature representations. Explore our [Blazor Area Chart Example](#) to know how to represent time-dependent data, showing trends at equal intervals.

Multicolored Area

To render a multicolored area series, specify the [Type](#) property as [MultiColoredArea](#) in [ChartSeries](#). Here, the individual color of the segment can be mapped by using the [Color](#) property in [ChartSegment](#).

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart Width="60%">
<ChartSeriesCollection>
<ChartSeries DataSource="@WeatherReports" XName="X" YName="Y"
Type="ChartSeriesType.MultiColoredArea">
<ChartSegments>
<ChartSegment Value="2007" Color="blue"/>
<ChartSegment Value="2009" Color="red"/>
<ChartSegment Color="green"></ChartSegment>
</ChartSegments>
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code{
public class ChartData
{
public double X { get; set; }
public double Y { get; set; }
}
public List<ChartData> WeatherReports = new List<ChartData>
{
new ChartData{ X= 2005, Y= 28 },
new ChartData{ X= 2006, Y= 25},
new ChartData{ X= 2007, Y= 26 },
new ChartData{ X= 2008, Y= 27 },
new ChartData{ X= 2009, Y= 32},
new ChartData{ X= 2010, Y= 35 },
new ChartData{ X= 2011, Y= 25 }
};
}
```



Series Customization

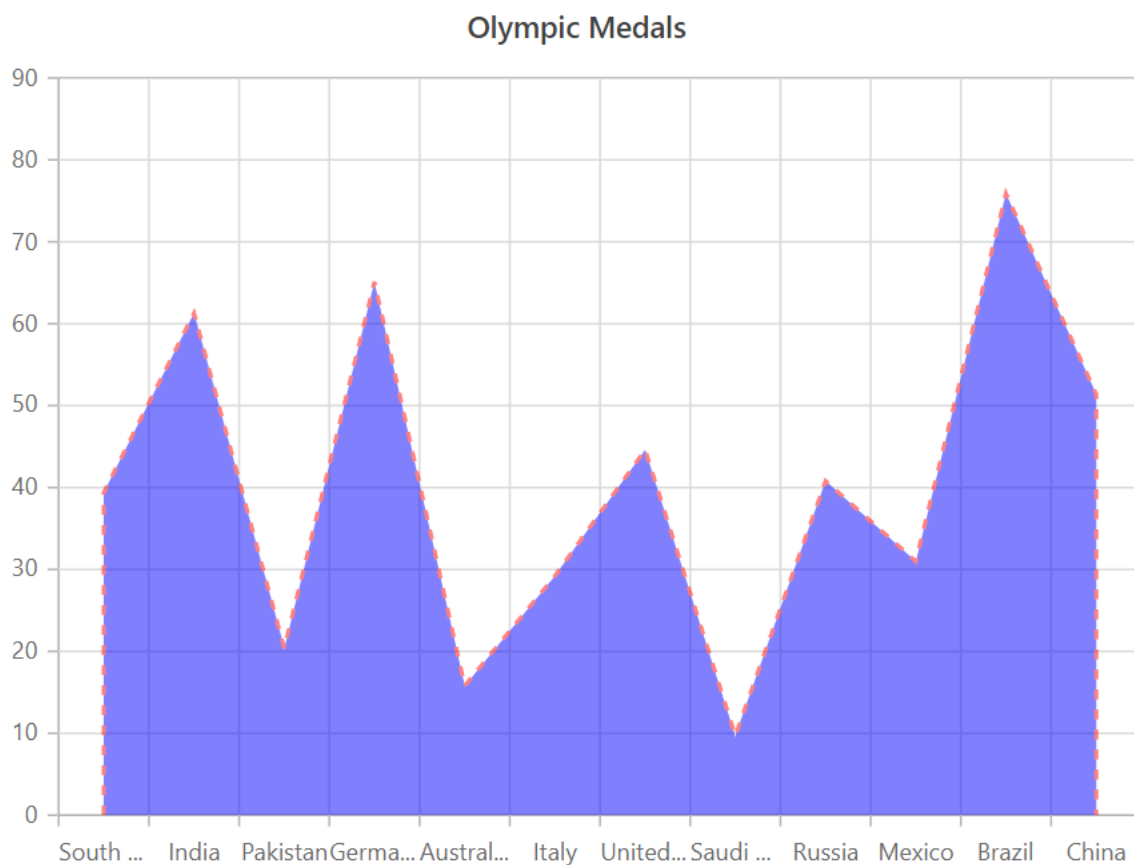
The following properties can be used to customize the [Area](#) series.

- [Fill](#) – Specifies the color of the area series.
- [Opacity](#) – Specifies the opacity of [Fill](#).
- [DashArray](#) – Specifies the dashes of series.
- [ChartSeriesBorder](#) – Specifies the [Color](#) and [Width](#) of series border.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart>
<ChartPrimaryXAxis ValueType="Syncfusion.Blazor.Charts.ValueType.Category"
/>
<ChartSeriesCollection>
<ChartSeries DataSource="@MedalDetails" XName="X" YName="Y" Opacity="0.5"
DashArray="5,5" Fill="blue" Type="ChartSeriesType.Area">
<ChartSeriesBorder Width="2" Color="red"></ChartSeriesBorder>
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code{
public class ChartData
{
public string X { get; set; }
```

```
public double Y { get; set; }  
}  
public List<ChartData> MedalDetails = new List<ChartData>  
{  
    new ChartData { X= "South Korea", Y= 39.4 },  
    new ChartData { X= "India", Y= 61.3 },  
    new ChartData { X= "Pakistan", Y= 20.4 },  
    new ChartData { X= "Germany", Y= 65.1 },  
    new ChartData { X= "Australia", Y= 15.8 }  
};  
}
```



Refer to our [Blazor Charts](#) feature tour page for its groundbreaking feature representations and also explore our [Blazor Chart Example](#) to know various chart types and how to represent time-dependent data, showing trends at equal intervals.

See Also

- [Data Label](#)
- [Tooltip](#)

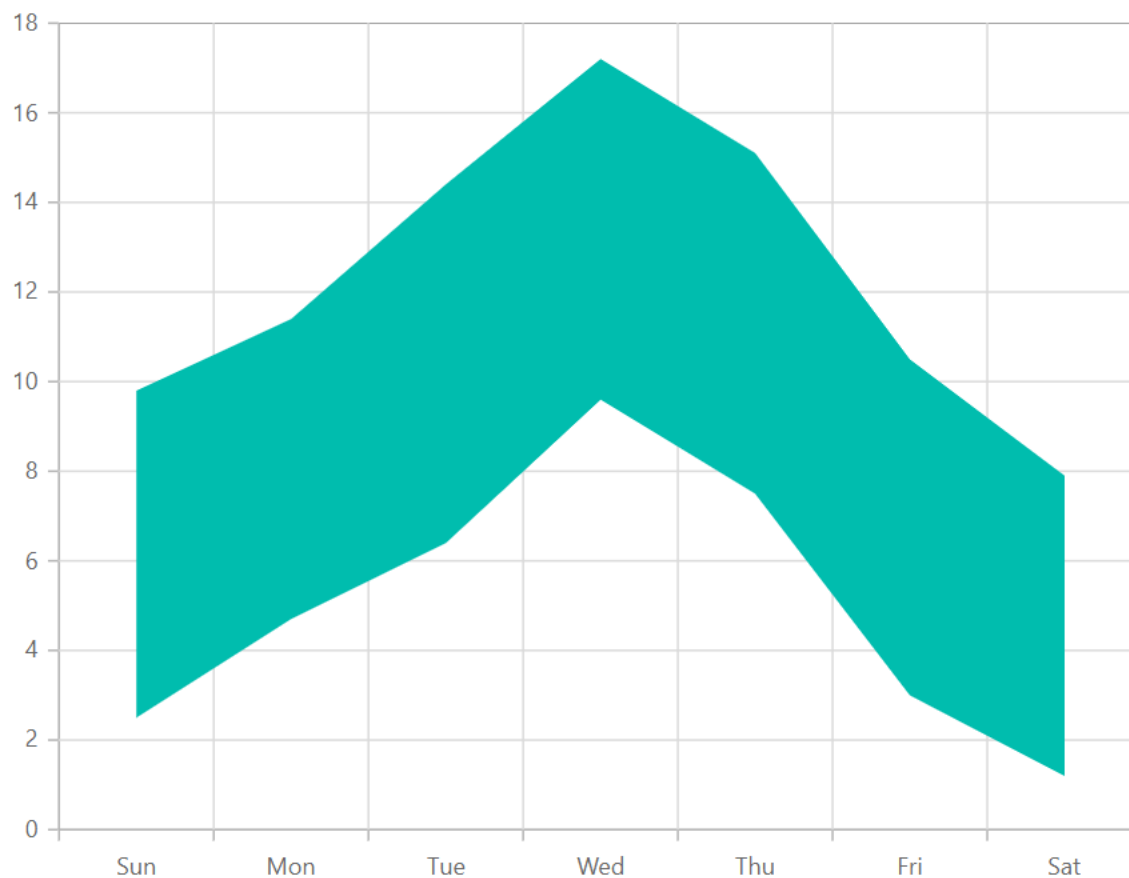
Range Area in Blazor Charts Component

Range Area

[Range Area Chart](#) shows variation in the data values for a given time. The area between the high and low range is filled. To render a [Range Area Chart](#), set the series [Type](#) as [RangeArea](#).

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart>
<ChartPrimaryXAxis ValueType="Syncfusion.Blazor.Charts.ValueType.Category"/>
<ChartSeriesCollection>
<ChartSeries DataSource="@WeatherReports" XName="X" High="High" Low="Low"
Type="ChartSeriesType.RangeArea">
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code{
public class ChartData
{
public string X { get; set; }
public double Low { get; set; }
public double High { get; set; }
}
public List<ChartData> WeatherReports = new List<ChartData>
{
new ChartData { X= "Sun", Low= 2.5, High= 9.8 },
new ChartData { X= "Mon", Low= 4.7, High= 11.4 },
new ChartData { X= "Tue", Low= 6.4, High= 14.4 },
new ChartData { X= "Wed", Low= 9.6, High= 17.2 },
new ChartData { X= "Thu", Low= 7.5, High= 15.1 },
new ChartData { X= "Fri", Low= 3.0, High= 10.5 },
new ChartData { X= "Sat", Low= 1.2, High= 7.9 }
};
}
```



Refer to our [Blazor Range Area Chart](#) feature tour page to know about its other groundbreaking feature representations. Explore our [Blazor Range Area Chart Example](#) to know how to show variations in the data values for a given time.

Series Customization

The following properties can be used to customize the [Range Area](#) series.

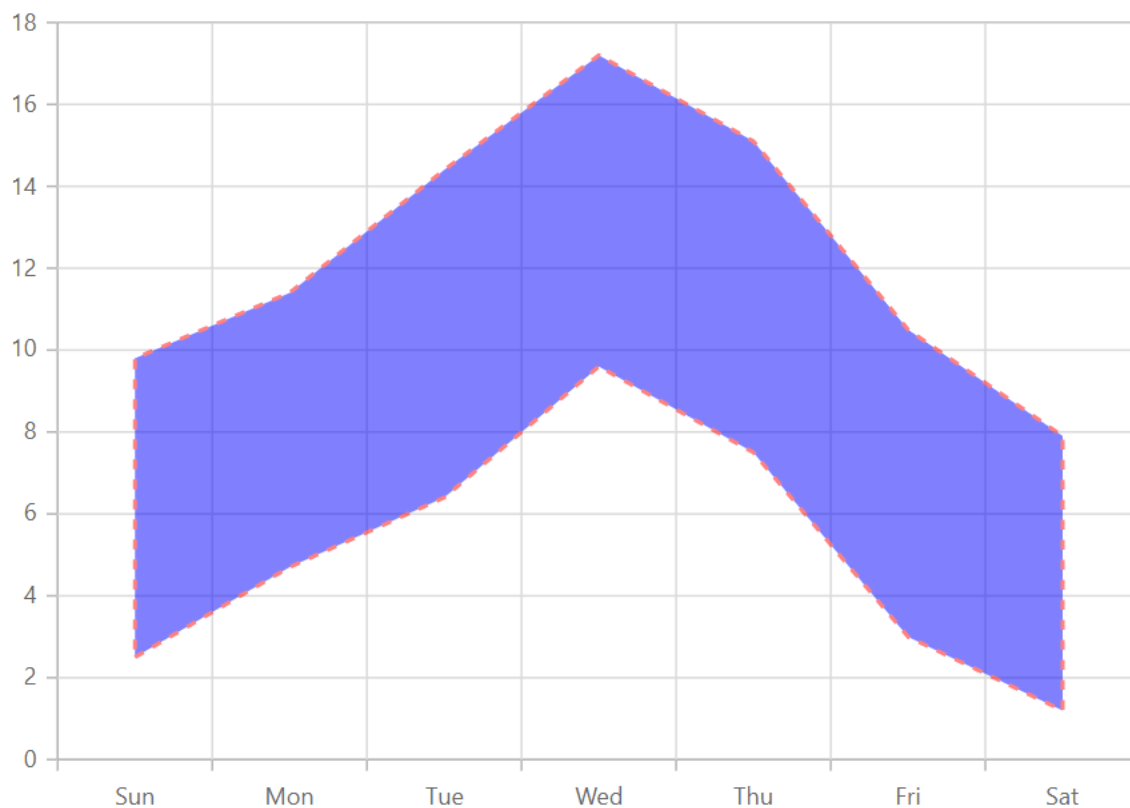
- [Fill](#) – Specifies the color of the series.
- [Opacity](#) – Specifies the opacity of [Fill](#).
- [DashArray](#) – Specifies the dashes of series border.
- [ChartSeriesBorder](#) – Specifies the [Color](#) and [Width](#) of series border.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart>
<ChartPrimaryXAxis ValueType="Syncfusion.Blazor.Charts.ValueType.Category"
/>
<ChartSeriesCollection>
<ChartSeries DataSource="@WeatherReports" XName="X" High="High" Low="Low"
Opacity="0.5"
DashArray="5,5" Fill="blue" Type="ChartSeriesType.RangeArea">
<ChartSeriesBorder Width="2" Color="red"></ChartSeriesBorder>
</ChartSeries>
```



```
</ChartSeriesCollection>
</SfChart>
@code{
public class ChartData
{
    public string X { get; set; }
    public double Low { get; set; }
    public double High { get; set; }
}
public List<ChartData> WeatherReports = new List<ChartData>
{
    new ChartData { X= "Sun", Low= 2.5, High= 9.8 },
    new ChartData { X= "Mon", Low= 4.7, High= 11.4 },
    new ChartData { X= "Tue", Low= 6.4, High= 14.4 },
    new ChartData { X= "Wed", Low= 9.6, High= 17.2 },
    new ChartData { X= "Thu", Low= 7.5, High= 15.1 },
    new ChartData { X= "Fri", Low= 3.0, High= 10.5 },
    new ChartData { X= "Sat", Low= 1.2, High= 7.9 }
};
}
```



Refer to our [Blazor Charts](#) feature tour page for its groundbreaking feature representations and also explore our [Blazor Chart Example](#) to know various chart types and how to represent time-dependent data, showing trends at equal intervals.

See Also

- [Data Label](#)
- [Tooltip](#)

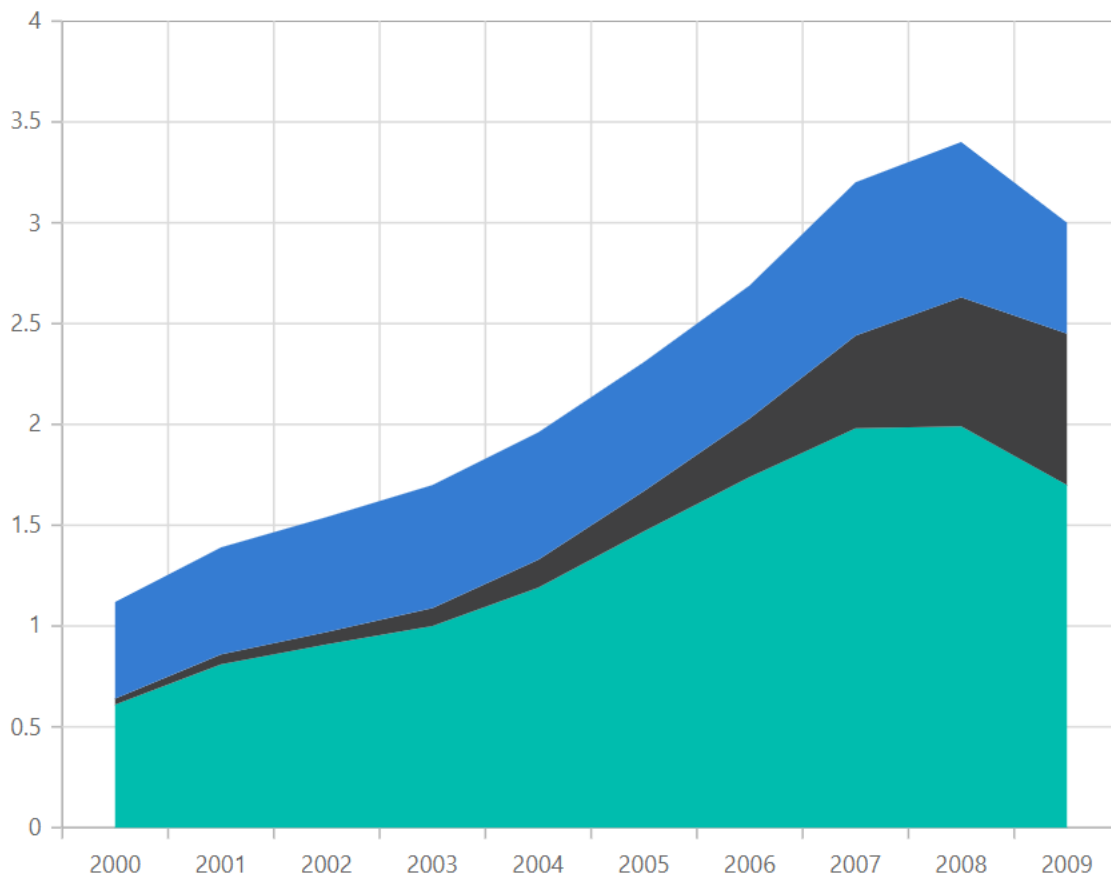
Stacked Area in Blazor Charts Component

Stacked Area

[Blazor Stacked Area Chart](#) is a chart with Y values stacked over one another in the series order. It shows the relation between individual values to the total sum of the points. To render a [Stacked Area](#) series, set the series [Type](#) as [StackingArea](#).

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart>
<ChartPrimaryXAxis ValueType="Syncfusion.Blazor.Charts.ValueType.Category"/>
<ChartSeriesCollection>
<ChartSeries DataSource="@MedalDetails" XName="X" YName="Y"
Type="ChartSeriesType.StackingArea"/>
<ChartSeries DataSource="@MedalDetails" XName="X" YName="Y1"
Type="ChartSeriesType.StackingArea"/>
<ChartSeries DataSource="@MedalDetails" XName="X" YName="Y2"
Type="ChartSeriesType.StackingArea"/>
</ChartSeriesCollection>
</SfChart>
@code{
public class ChartData
{
public double X { get; set; }
public double Y { get; set; }
public double Y1 { get; set; }
public double Y2 { get; set; }
}
public List<ChartData> MedalDetails = new List<ChartData>
{
new ChartData{ X=2000, Y= 0.61, Y1= 0.03, Y2= 0.48},
new ChartData{ X=2001, Y= 0.81, Y1= 0.05, Y2= 0.53 },
new ChartData{ X=2002, Y= 0.91, Y1= 0.06, Y2= 0.57 },
new ChartData{ X=2003, Y= 1, Y1= 0.09, Y2= 0.61 },
new ChartData{ X=2004, Y= 1.19, Y1= 0.14, Y2= 0.63 },
new ChartData{ X=2005, Y= 1.47, Y1= 0.20, Y2= 0.64 },
new ChartData{ X=2006, Y= 1.74, Y1= 0.29, Y2= 0.66 },
new ChartData{ X=2007, Y= 1.98, Y1= 0.46, Y2= 0.76 },
new ChartData{ X=2008, Y= 1.99, Y1= 0.64, Y2= 0.77 },
new ChartData{ X=2009, Y= 1.70, Y1= 0.75, Y2= 0.55 }
};
}
```



Refer to our [Blazor Stacked Area Chart](#) feature tour page to know about its other groundbreaking feature representations. Explore our [Blazor Stacked Area Chart Example](#) to know how to render and configure the Stacked Area type chart.

Series Customization

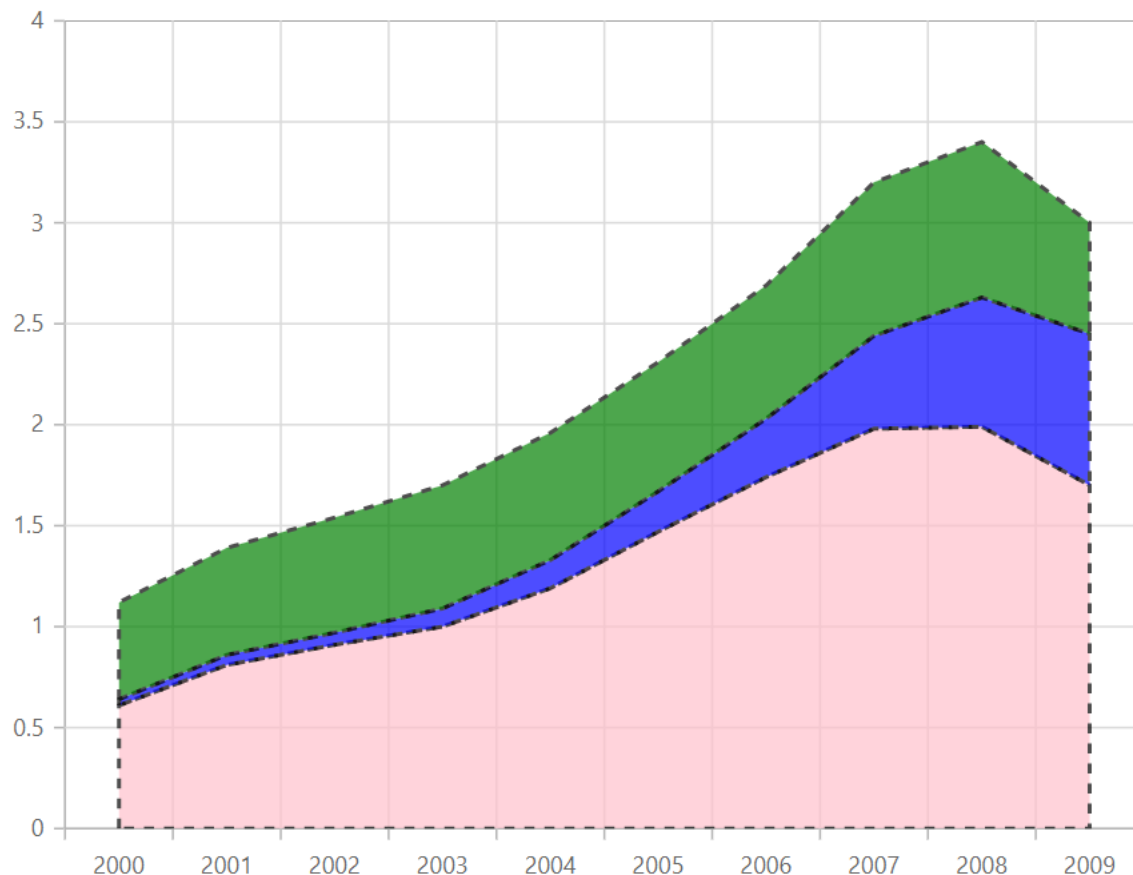
The following properties can be used to customize the [Stacked Area](#) series.

- [Fill](#) – Specifies the color of the series.
- [Opacity](#) – Specifies the opacity of [Fill](#).
- [DashArray](#) – Specifies the dashes of series border.
- [ChartSeriesBorder](#) – Specifies the [Color](#) and [Width](#) of series border.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart>
<ChartPrimaryXAxis ValueType="Syncfusion.Blazor.Charts.ValueType.Category"
/>
<ChartSeriesCollection>
<ChartSeries DataSource="@MedalDetails" XName="X" YName="Y" Fill="pink"
Opacity="0.7" DashArray="5,5" Type="ChartSeriesType.StackingArea">
<ChartSeriesBorder Width="2" Color="black"></ChartSeriesBorder>
</ChartSeries>
```

```
<ChartSeries DataSource="@MedalDetails" XName="X" YName="Y1" Fill="blue"
Opacity="0.7" DashArray="5,5" Type="ChartSeriesType.StackingArea">
<ChartSeriesBorder Width="2" Color="black"></ChartSeriesBorder>
</ChartSeries>
<ChartSeries DataSource="@MedalDetails" XName="X" YName="Y2" Fill="green"
Opacity="0.7" DashArray="5,5" Type="ChartSeriesType.StackingArea">
<ChartSeriesBorder Width="2" Color="black"></ChartSeriesBorder>
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code{
public class ChartData
{
public double X { get; set; }
public double Y { get; set; }
public double Y1 { get; set; }
public double Y2 { get; set; }
}
public List<ChartData> MedalDetails = new List<ChartData>
{
new ChartData{ X=2000, Y= 0.61, Y1= 0.03, Y2= 0.48},
new ChartData{ X=2001, Y= 0.81, Y1= 0.05, Y2= 0.53 },
new ChartData{ X=2002, Y= 0.91, Y1= 0.06, Y2= 0.57 },
new ChartData{ X=2003, Y= 1, Y1= 0.09, Y2= 0.61 },
new ChartData{ X=2004, Y= 1.19, Y1= 0.14, Y2= 0.63 },
new ChartData{ X=2005, Y= 1.47, Y1= 0.20, Y2= 0.64 },
new ChartData{ X=2006, Y= 1.74, Y1= 0.29, Y2= 0.66 },
new ChartData{ X=2007, Y= 1.98, Y1= 0.46, Y2= 0.76 },
new ChartData{ X=2008, Y= 1.99, Y1= 0.64, Y2= 0.77 },
new ChartData{ X=2009, Y= 1.70, Y1= 0.75, Y2= 0.55 }
};
}
```



Refer to our [Blazor Charts](#) feature tour page for its groundbreaking feature representations and also explore our [Blazor Chart Example](#) to know various chart types and how to represent time-dependent data, showing trends at equal intervals.

See Also

- [Data Label](#)
- [Tooltip](#)

100% Stacked Area in Blazor Charts Component

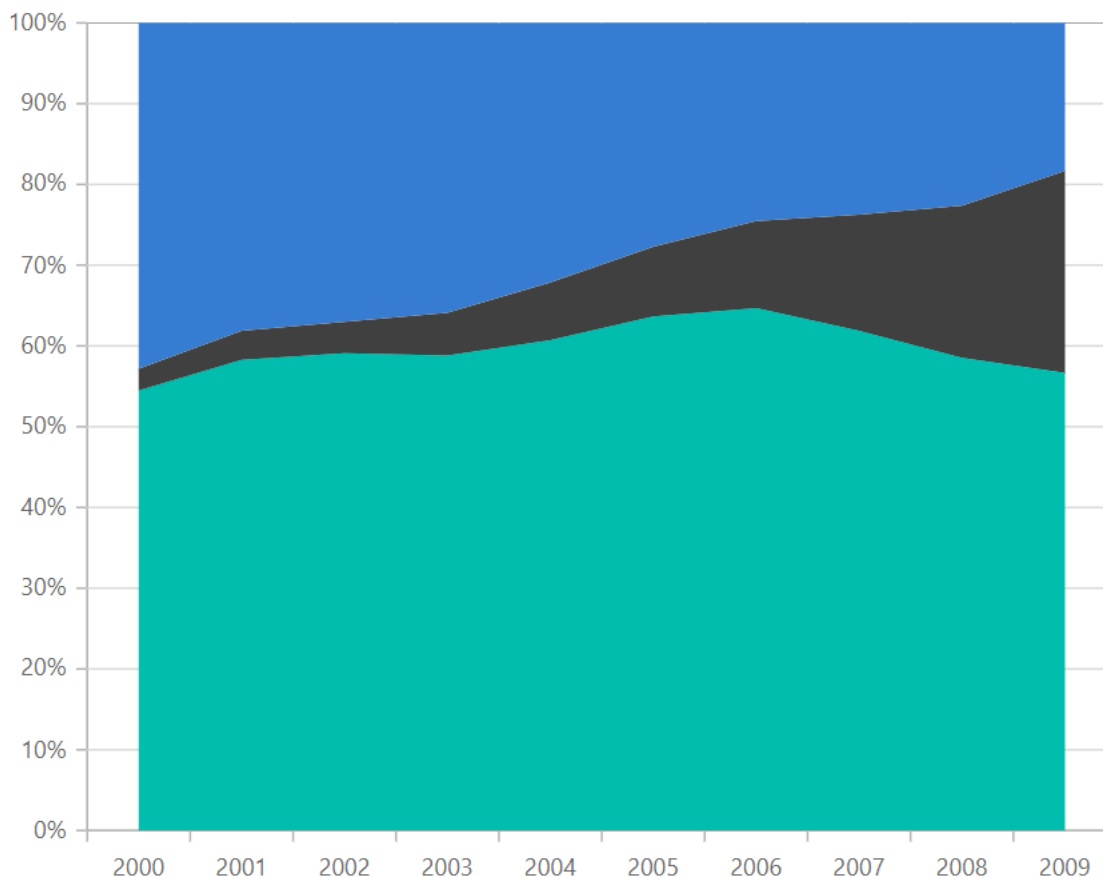
100% Stacked Area Chart

[100% Stacked Area Chart](#) displays multiple series of data as stacked areas, ensuring that the cumulative proportion of each stacked element always totals 100%. Hence, the y-axis will always be rendered with the range 0–100%. To render a [100% Stacked Area](#) series, set the series [Type](#) as [StackingArea100](#).

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart>
<ChartPrimaryXAxis ValueTypes="Syncfusion.Blazor.Charts.ValueType.Category"/>
<ChartSeriesCollection>
<ChartSeries DataSource="@MedalDetails" XName="X" YName="Y"
Type="ChartSeriesType.StackingArea100"/>
<ChartSeries DataSource="@MedalDetails" XName="X" YName="Y1"
Type="ChartSeriesType.StackingArea100"/>
</ChartSeriesCollection>
</SfChart>
```

```
<ChartSeries DataSource="@MedalDetails" XName="X" YName="Y2"
Type="ChartSeriesType.StackingArea100"/>
</ChartSeriesCollection>
</SfChart>
@code{
public class ChartData
{
public double X { get; set; }
public double Y { get; set; }
public double Y1 { get; set; }
public double Y2 { get; set; }
}
public List<ChartData> MedalDetails = new List<ChartData>
{
new ChartData{ X=2000, Y= 0.61, Y1= 0.03, Y2= 0.48},
new ChartData{ X=2001, Y= 0.81, Y1= 0.05, Y2= 0.53 },
new ChartData{ X=2002, Y= 0.91, Y1= 0.06, Y2= 0.57 },
new ChartData{ X=2003, Y= 1, Y1= 0.09, Y2= 0.61 },
new ChartData{ X=2004, Y= 1.19, Y1= 0.14, Y2= 0.63 },
new ChartData{ X=2005, Y= 1.47, Y1= 0.20, Y2= 0.64 },
new ChartData{ X=2006, Y= 1.74, Y1= 0.29, Y2= 0.66 },
new ChartData{ X=2007, Y= 1.98, Y1= 0.46, Y2= 0.76 },
new ChartData{ X=2008, Y= 1.99, Y1= 0.64, Y2= 0.77 },
new ChartData{ X=2009, Y= 1.70, Y1= 0.75, Y2= 0.55 }
};
}
```



Refer to our [Blazor 100% Stacked Area Chart](#) feature tour page to know about its other groundbreaking feature representations. Explore our [Blazor 100% Stacked Area Chart Example](#) to know how to render and configure the 100% Stacked Area type chart.

Series Customization

The following properties can be used to customize the [100% Stacked Area](#) series.

- [Fill](#) – Specifies the color of the series.
- [Opacity](#) – Specifies the opacity of [Fill](#).
- [DashArray](#) – Specifies the dashes of series border.
- [ChartSeriesBorder](#) – Specifies the [Color](#) and [Width](#) of series border.

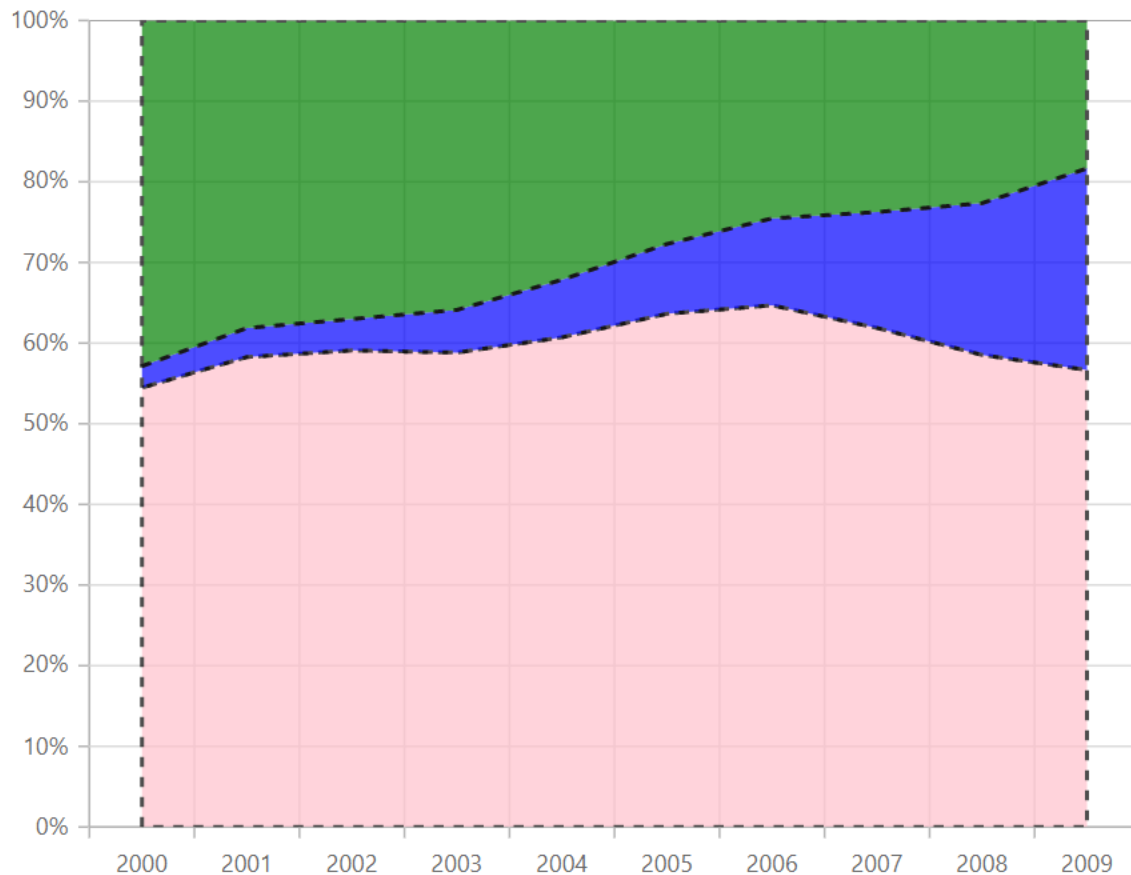
ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart>
<ChartPrimaryXAxis ValueType="Syncfusion.Blazor.Charts.ValueType.Category"
/>
<ChartSeriesCollection>
<ChartSeries DataSource="@MedalDetails" XName="X" YName="Y" Fill="pink"
Opacity="0.7" DashArray="5,5" Type="ChartSeriesType.StackingArea100">
<ChartSeriesBorder Width="2" Color="black"></ChartSeriesBorder>
</ChartSeries>
```

```

<ChartSeries DataSource="@MedalDetails" XName="X" YName="Y1" Fill="blue"
Opacity="0.7" DashArray="5,5" Type="ChartSeriesType.StackingArea100">
<ChartSeriesBorder Width="2" Color="black"></ChartSeriesBorder>
</ChartSeries>
<ChartSeries DataSource="@MedalDetails" XName="X" YName="Y2" Fill="green"
Opacity="0.7" DashArray="5,5" Type="ChartSeriesType.StackingArea100">
<ChartSeriesBorder Width="2" Color="black"></ChartSeriesBorder>
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code{
public class ChartData
{
public double X { get; set; }
public double Y { get; set; }
public double Y1 { get; set; }
public double Y2 { get; set; }
}
public List<ChartData> MedalDetails = new List<ChartData>
{
new ChartData{ X=2000, Y= 0.61, Y1= 0.03, Y2= 0.48},
new ChartData{ X=2001, Y= 0.81, Y1= 0.05, Y2= 0.53 },
new ChartData{ X=2002, Y= 0.91, Y1= 0.06, Y2= 0.57 },
new ChartData{ X=2003, Y= 1, Y1= 0.09, Y2= 0.61 },
new ChartData{ X=2004, Y= 1.19, Y1= 0.14, Y2= 0.63 },
new ChartData{ X=2005, Y= 1.47, Y1= 0.20, Y2= 0.64 },
new ChartData{ X=2006, Y= 1.74, Y1= 0.29, Y2= 0.66 },
new ChartData{ X=2007, Y= 1.98, Y1= 0.46, Y2= 0.76 },
new ChartData{ X=2008, Y= 1.99, Y1= 0.64, Y2= 0.77 },
new ChartData{ X=2009, Y= 1.70, Y1= 0.75, Y2= 0.55 }
};
}

```

Refer to our [Blazor Charts](#) feature tour page for its groundbreaking feature representations and also explore our [Blazor Chart Example](#) to know various chart types and how to represent time-dependent data, showing trends at equal intervals.

See Also

- [Data Label](#)
- [Tooltip](#)

Step Area in Blazor Charts Component

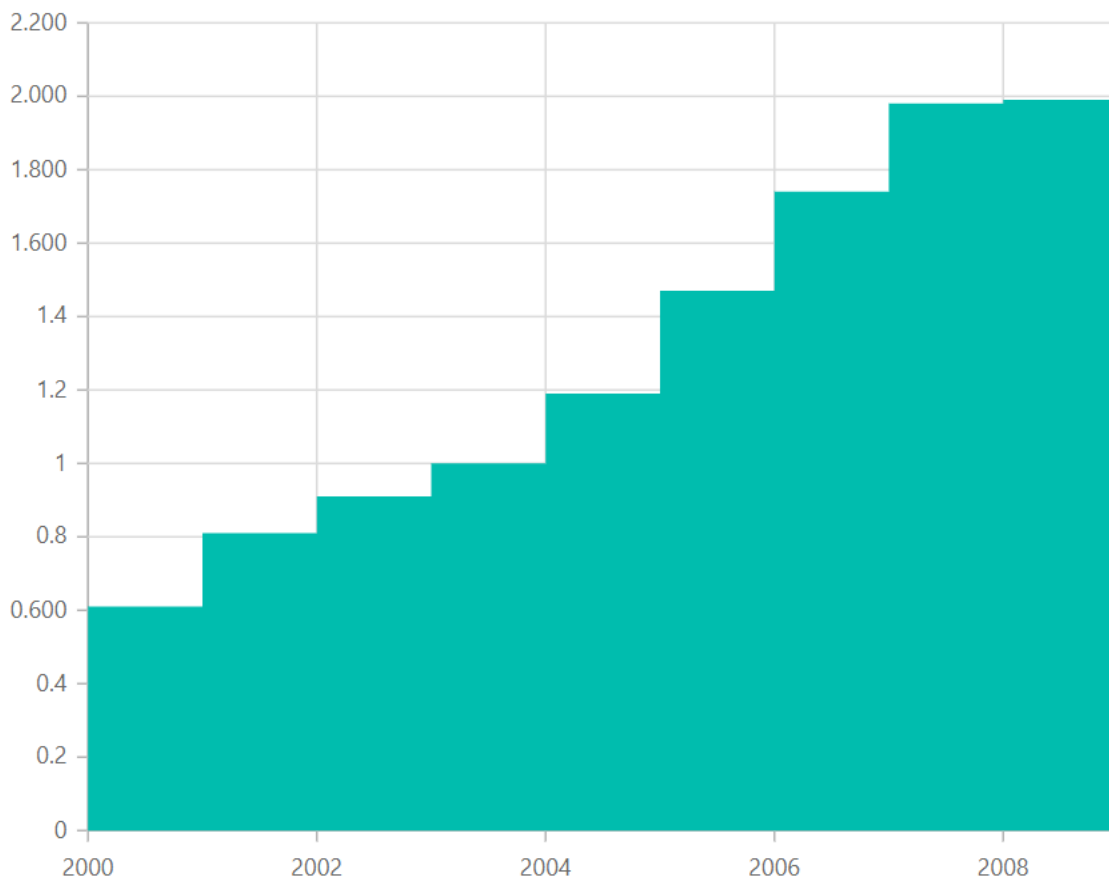
Step Area

[Step Area Chart](#) is similar to a step line chart, but with the areas connected with lines shaded. To render a step area series, set the series [Type](#) as [StepArea](#).

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart>
<ChartPrimaryXAxis ValueType="Syncfusion.Blazor.Charts.ValueType.Category"/>
<ChartSeriesCollection>
<ChartSeries DataSource="@WeatherReports" XName="X" YName="Low"
Type="ChartSeriesType.StepArea">
</ChartSeries>
</ChartSeriesCollection>
```

```
</SfChart>
@code{
public class ChartData
{
    public string X { get; set; }
    public double Low { get; set; }
    public double High { get; set; }
}
public List<ChartData> WeatherReports = new List<ChartData>
{
    new ChartData { X= "Sun", Low= 2.5, High= 9.8 },
    new ChartData { X= "Mon", Low= 4.7, High= 11.4 },
    new ChartData { X= "Tue", Low= 6.4, High= 14.4 },
    new ChartData { X= "Wed", Low= 9.6, High= 17.2 },
    new ChartData { X= "Thu", Low= 7.5, High= 15.1 },
    new ChartData { X= "Fri", Low= 3.0, High= 10.5 },
    new ChartData { X= "Sat", Low= 1.2, High= 7.9 }
};
}
```



Refer to our [Blazor Step Area Chart](#) feature tour page to know about its other groundbreaking feature representations. Explore our [Blazor Step Area Chart Example](#) to know how to render and configure the Step Area type chart.

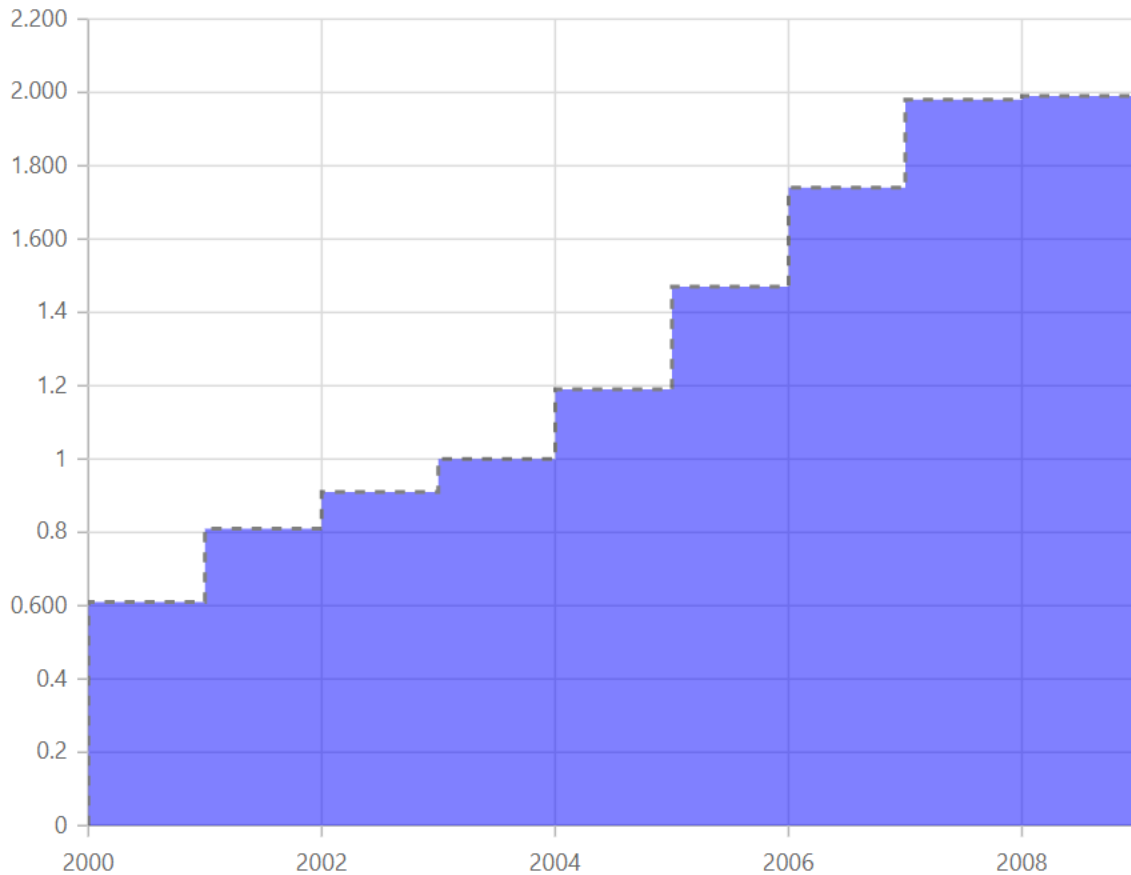
Series Customization

The following properties can be used to customize the [Step Area](#) series.

- [Fill](#) – Specifies the color of the series.
- [Opacity](#) – Specifies the opacity of [Fill](#).
- [DashArray](#) – Specifies the dashes of series border.
- [ChartSeriesBorder](#) – Specifies the [Color](#) and [Width](#) of series border.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart>
<ChartPrimaryXAxis ValueType="Syncfusion.Blazor.Charts.ValueType.Category"
/>
<ChartSeriesCollection>
<ChartSeries DataSource="@WeatherReports" XName="X" Fill="blue" YName="Low"
DashArray="5,5" Opacity="0.5" Type="ChartSeriesType.StepArea">
<ChartSeriesBorder Width="2" Color="black"></ChartSeriesBorder>
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code{
public class ChartData
{
public string X { get; set; }
public double Low { get; set; }
public double High { get; set; }
}
public List<ChartData> WeatherReports = new List<ChartData>
{
new ChartData { X= "Sun", Low= 2.5, High= 9.8 },
new ChartData { X= "Mon", Low= 4.7, High= 11.4 },
new ChartData { X= "Tue", Low= 6.4, High= 14.4 },
new ChartData { X= "Wed", Low= 9.6, High= 17.2 },
new ChartData { X= "Thu", Low= 7.5, High= 15.1 },
new ChartData { X= "Fri", Low= 3.0, High= 10.5 },
new ChartData { X= "Sat", Low= 1.2, High= 7.9 }
};
}
```



Refer to our [Blazor Charts](#) feature tour page for its groundbreaking feature representations and also explore our [Blazor Chart Example](#) to know various chart types and how to represent time-dependent data, showing trends at equal intervals.

See Also

- [Data Label](#)
- [Tooltip](#)

Spline Area in Blazor Charts Component

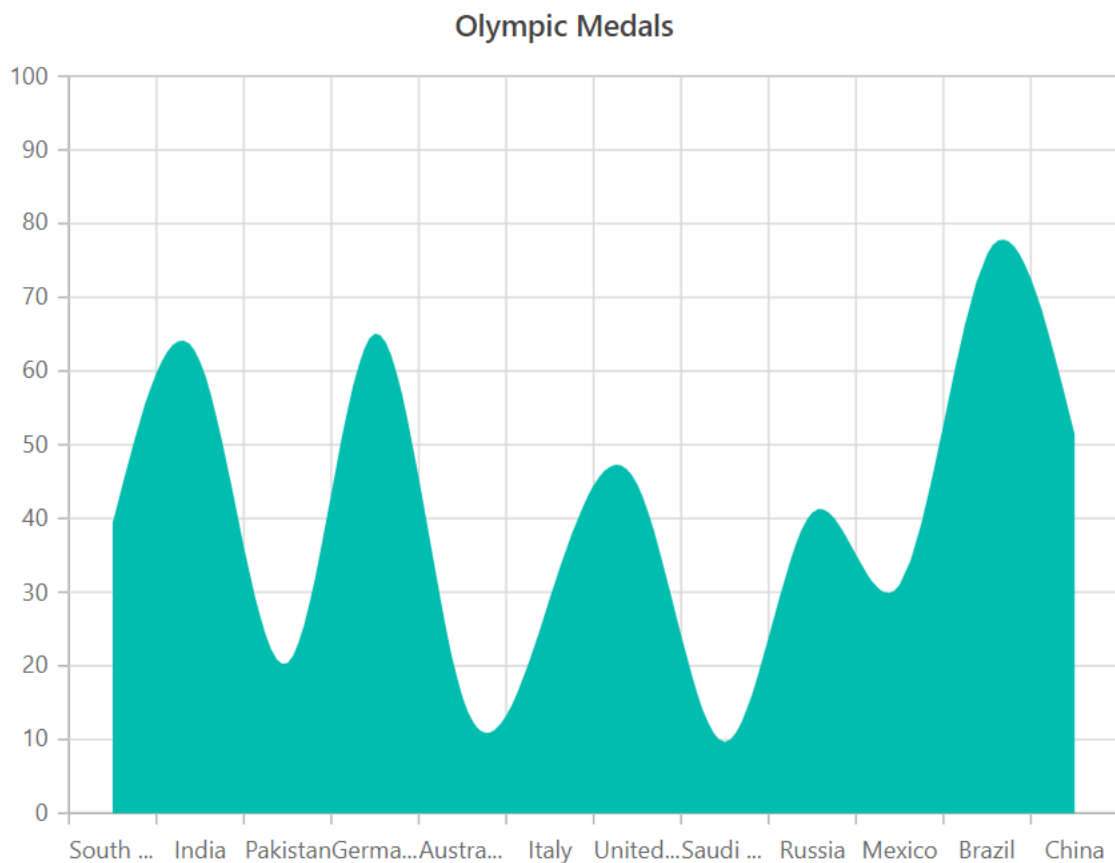
Spline Area

[Spline Area Chart](#) represents time dependent data and visualizes trends at equal intervals, but with data points connected with a smooth line. To render a [Spline Area Chart](#), set the series [Type](#) as [Spline Area](#).

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart Title="Olympic Medals">
  <ChartPrimaryXAxis
    ValueType="Syncfusion.Blazor.Charts.ValueType.Category"></ChartPrimaryXAxis>
  <ChartSeriesCollection>
    <ChartSeries DataSource="@MedalDetails" XName="X" YName="Y"
      Type="ChartSeriesType.SplineArea">
    </ChartSeries>
  </ChartSeriesCollection>
```

```
</SfChart>
@code{
public class ChartData
{
public string X { get; set;}
public double Y { get; set; }
}
public List<ChartData> MedalDetails = new List<ChartData>
{
new ChartData { X= "South Korea", Y= 39.4 },
new ChartData { X= "India", Y= 61.3 },
new ChartData { X= "Pakistan", Y= 20.4 },
new ChartData { X= "Germany", Y= 65.1 },
new ChartData { X= "Australia", Y= 15.8 },
new ChartData { X= "Italy", Y= 29.2 },
new ChartData { X= "United Kingdom", Y= 44.6 },
new ChartData { X= "Saudi Arabia", Y= 9.7 },
new ChartData { X= "Russia", Y= 40.8 },
new ChartData { X= "Mexico", Y= 31 },
new ChartData { X= "Brazil", Y= 75.9 },
new ChartData { X= "China", Y= 51.4 }
};
}
```



Refer to our [Blazor Spline Area Chart](#) feature tour page to know about its other groundbreaking feature representations. Explore our [Blazor Spline Area Chart Example](#) to know how to connect the data points with smooth curves.

Series Customization

The following properties can be used to customize the [Spline Area](#) series.

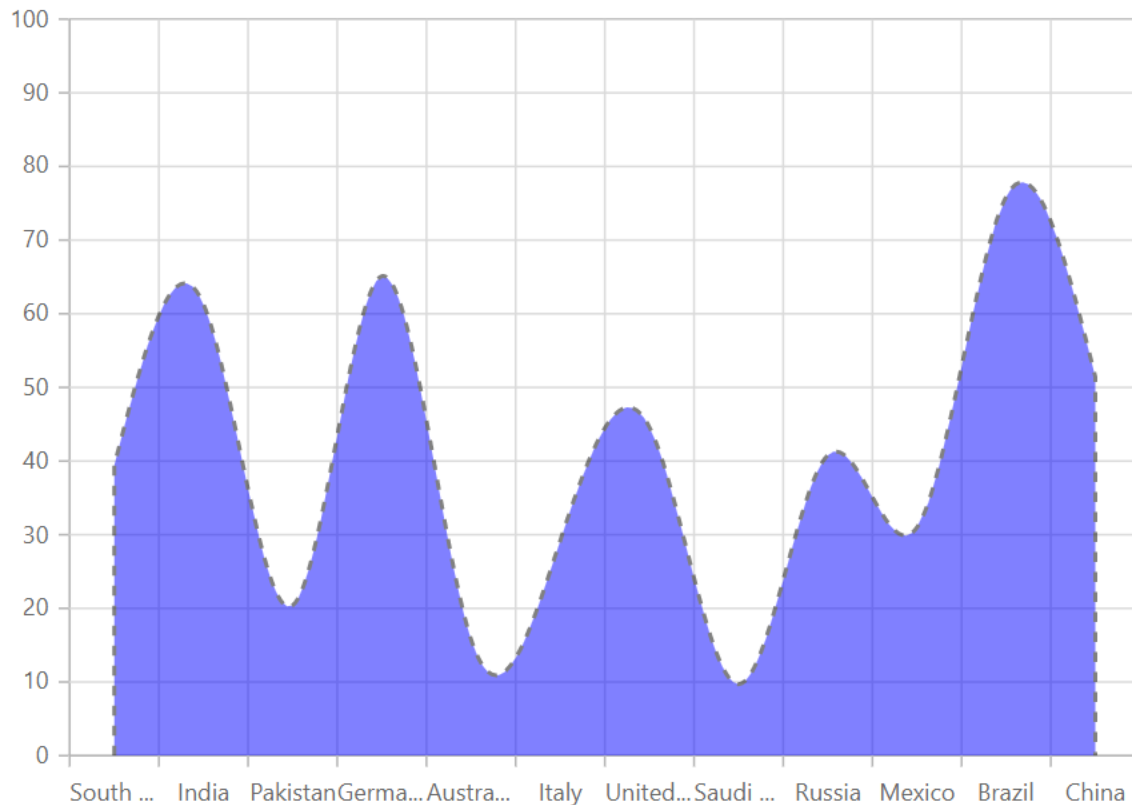
- [Fill](#) – Specifies the color of the series.
- [Opacity](#) – Specifies the opacity of [Fill](#).
- [DashArray](#) – Specifies the dashes of series border.
- [ChartSeriesBorder](#) – Specifies the [Color](#) and [Width](#) of series border.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart Title="Olympic Medals">
  <ChartPrimaryXAxis
    ValueType="Syncfusion.Blazor.Charts.ValueType.Category"></ChartPrimaryXAxis>
  <ChartSeriesCollection>
    <ChartSeries DataSource="@MedalDetails" XName="X" YName="Y" Opacity="0.5"
      DashArray="5,5" Fill="blue" Type="ChartSeriesType.SplineArea">
      <ChartSeriesBorder Width="2" Color="black"></ChartSeriesBorder>
    </ChartSeries>
  </ChartSeriesCollection>
</SfChart>

@code{
public class ChartData
{
    public string X { get; set; }
    public double Y { get; set; }
}

public List<ChartData> MedalDetails = new List<ChartData>
{
    new ChartData { X= "South Korea", Y= 39.4 },
    new ChartData { X= "India", Y= 61.3 },
    new ChartData { X= "Pakistan", Y= 20.4 },
    new ChartData { X= "Germany", Y= 65.1 },
    new ChartData { X= "Australia", Y= 15.8 },
    new ChartData { X= "Italy", Y= 29.2 },
    new ChartData { X= "United Kingdom", Y= 44.6 },
    new ChartData { X= "Saudi Arabia", Y= 9.7 },
    new ChartData { X= "Russia", Y= 40.8 },
    new ChartData { X= "Mexico", Y= 31 },
    new ChartData { X= "Brazil", Y= 75.9 },
    new ChartData { X= "China", Y= 51.4 }
};
}
```



Refer to our [Blazor Charts](#) feature tour page for its groundbreaking feature representations and also explore our [Blazor Chart Example](#) to know various chart types and how to represent time-dependent data, showing trends at equal intervals.

See Also

- [Data Label](#)
- [Tooltip](#)

Column Chart in Blazor Charts Component

Column

[Column Chart](#) is the most common chart type that is used to compare **Frequency, Count, Total**, or **Average** of data in different categories. It is ideal for showing variations in the value of an item over time. To render a column series, set series [Type](#) as [Column](#).

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart>
<ChartPrimaryXAxis ValueType="Syncfusion.Blazor.Charts.ValueType.Category"/>
<ChartSeriesCollection>
<ChartSeries DataSource="@MedalDetails" XName="X" YName="Y"
Type="ChartSeriesType.Column">
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code{
```

```

public class ChartData
{
    public string X { get; set; }
    public double Y { get; set; }
}
public List<ChartData> MedalDetails = new List<ChartData>
{
    new ChartData { X= "South Korea", Y= 39.4 },
    new ChartData { X= "India", Y= 61.3 },
    new ChartData { X= "Pakistan", Y= 20.4 },
    new ChartData { X= "Germany", Y= 65.1 },
    new ChartData { X= "Australia", Y= 15.8 },
    new ChartData { X= "Italy", Y= 29.2 },
    new ChartData { X= "United Kingdom", Y= 44.6 },
    new ChartData { X= "Saudi Arabia", Y= 9.7 },
    new ChartData { X= "Russia", Y= 40.8 },
    new ChartData { X= "Mexico", Y= 31 },
    new ChartData { X= "Brazil", Y= 75.9 },
    new ChartData { X= "China", Y= 51.4 }
};
}

```

Refer to our [Blazor Column Charts](#) feature tour page to know about its other groundbreaking feature representations. Explore our [Blazor Column Charts Example](#) to compare **Frequency**, **Count**, **Total**, or **Average** of data in different categories.

Column Space and Width

The [ColumnSpacing](#) and [ColumnWidth](#) properties are used to customize the space between columns.

ASPX-CS

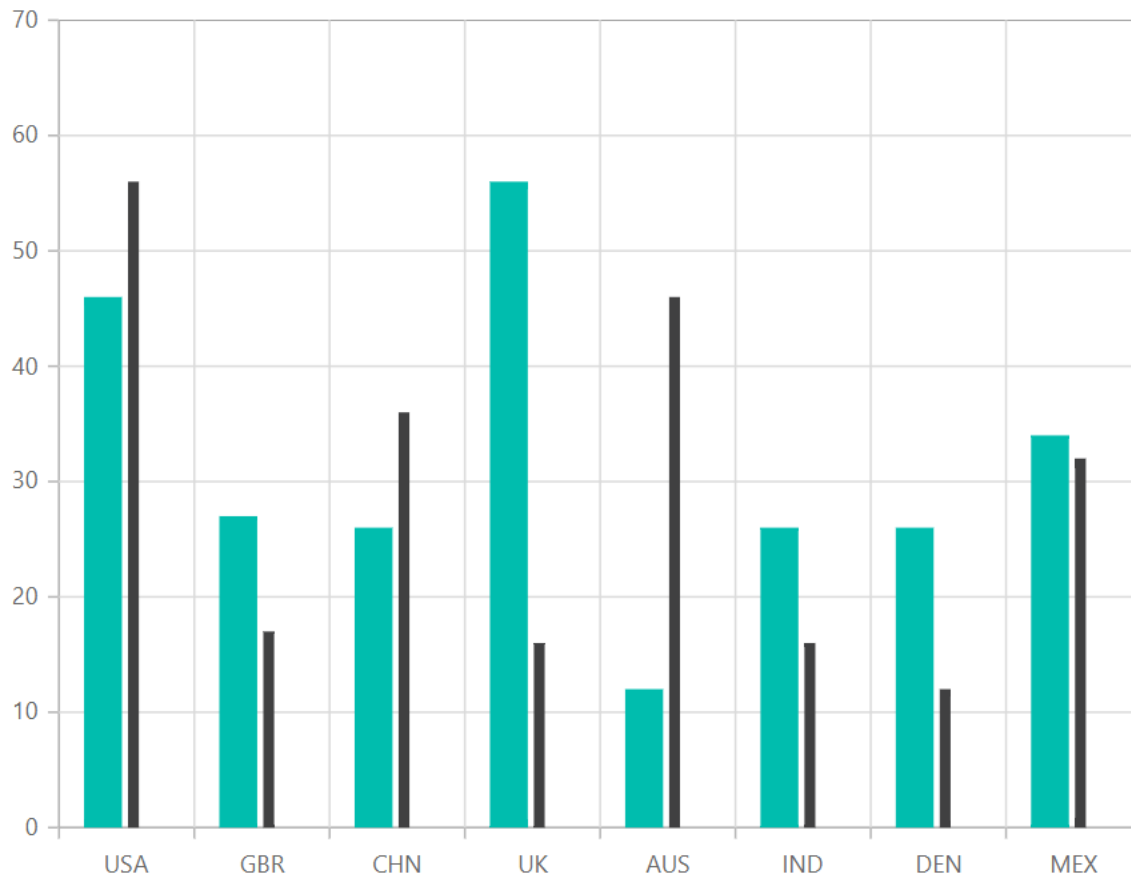
```

@using Syncfusion.Blazor.Charts
<SfChart>
<ChartPrimaryXAxis ValueType="Syncfusion.Blazor.Charts.ValueType.Category"
/>
<ChartSeriesCollection>
<ChartSeries DataSource="@MedalDetails" XName="X" YName="Y"
Type="ChartSeriesType.Column" ColumnSpacing="0.2" ColumnWidth="0.2" >
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code{
public class ChartData
{
    public string X { get; set; }
    public double Y { get; set; }
}
public List<ChartData> MedalDetails = new List<ChartData>
{
    new ChartData { X= "South Korea", Y= 39.4 },
    new ChartData { X= "India", Y= 61.3 },
    new ChartData { X= "Pakistan", Y= 20.4 },
    new ChartData { X= "Germany", Y= 65.1 },
    new ChartData { X= "Australia", Y= 15.8 },
    new ChartData { X= "Italy", Y= 29.2 },
    new ChartData { X= "United Kingdom", Y= 44.6 },
}

```



```
new ChartData { X= "Saudi Arabia", Y= 9.7 },
new ChartData { X= "Russia", Y= 40.8 },
new ChartData { X= "Mexico", Y= 31 },
new ChartData { X= "Brazil", Y= 75.9 },
new ChartData { X= "China", Y= 51.4 }
};
}
```



Series Customization

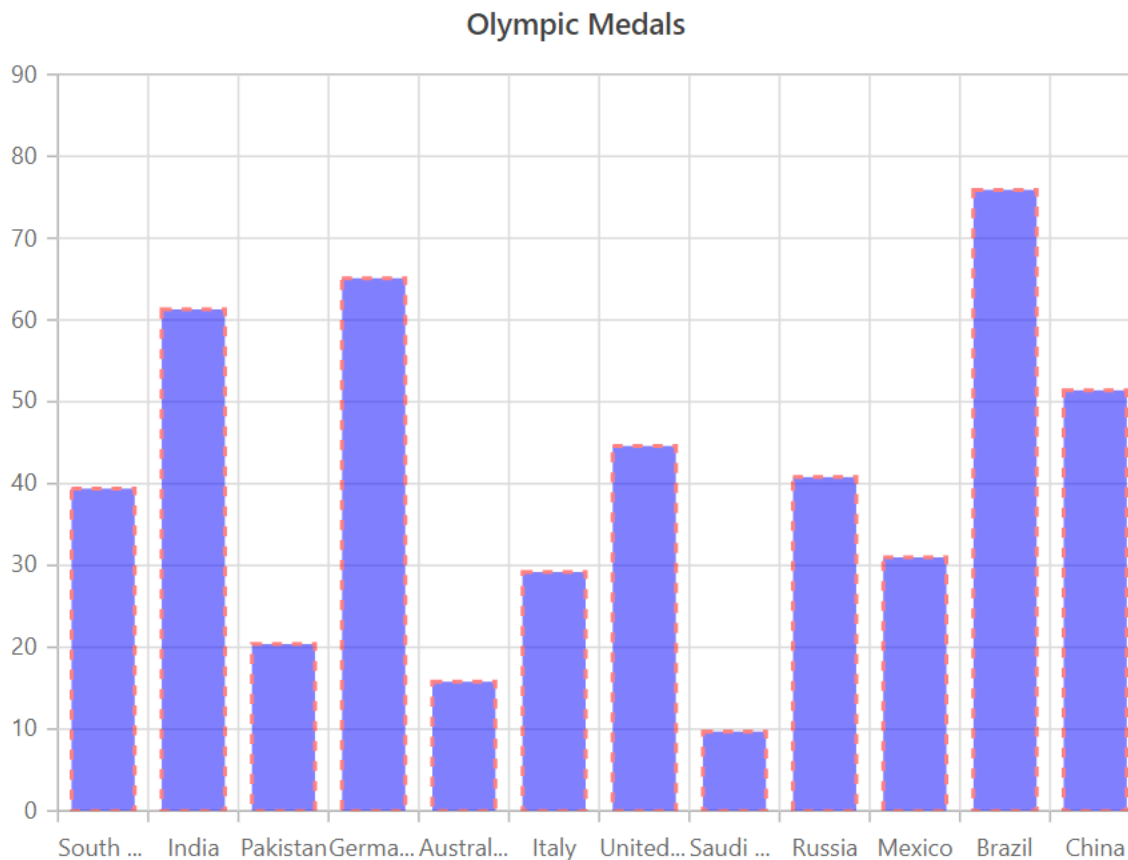
The following properties can be used to customize the [Column](#) series.

- [Fill](#) – Specifies the color of the series.
- [Opacity](#) – Specifies the opacity of [Fill](#).
- [DashArray](#) – Specifies the dashes of series.
- [ChartSeriesBorder](#) – Specifies the [Color](#) and [Width](#) of series border.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart>
<ChartPrimaryXAxis Value="Syncfusion.Blazor.Charts.ValueType.Category"
/>
<ChartSeriesCollection>
```

```
<ChartSeries DataSource="@MedalDetails" XName="X" YName="Y"
Type="ChartSeriesType.Column" Opacity="0.5"
DashArray="5,5" Fill="blue" >
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code{
public class ChartData
{
public string X { get; set; }
public double Y { get; set; }
}
public List<ChartData> MedalDetails = new List<ChartData>
{
new ChartData { X= "South Korea", Y= 39.4 },
new ChartData { X= "India", Y= 61.3 },
new ChartData { X= "Pakistan", Y= 20.4 },
new ChartData { X= "Germany", Y= 65.1 },
new ChartData { X= "Australia", Y= 15.8 },
new ChartData { X= "Italy", Y= 29.2 },
new ChartData { X= "United Kingdom", Y= 44.6 },
new ChartData { X= "Saudi Arabia", Y= 9.7 },
new ChartData { X= "Russia", Y= 40.8 },
new ChartData { X= "Mexico", Y= 31 },
new ChartData { X= "Brazil", Y= 75.9 },
new ChartData { X= "China", Y= 51.4 }
};
}
```



Refer to our [Blazor Charts](#) feature tour page for its groundbreaking feature representations and also explore our [Blazor Chart Example](#) to know various chart types and how to represent time-dependent data, showing trends at equal intervals.

See Also

- [Data Label](#)
- [Tooltip](#)

Range Column in Blazor Charts Component

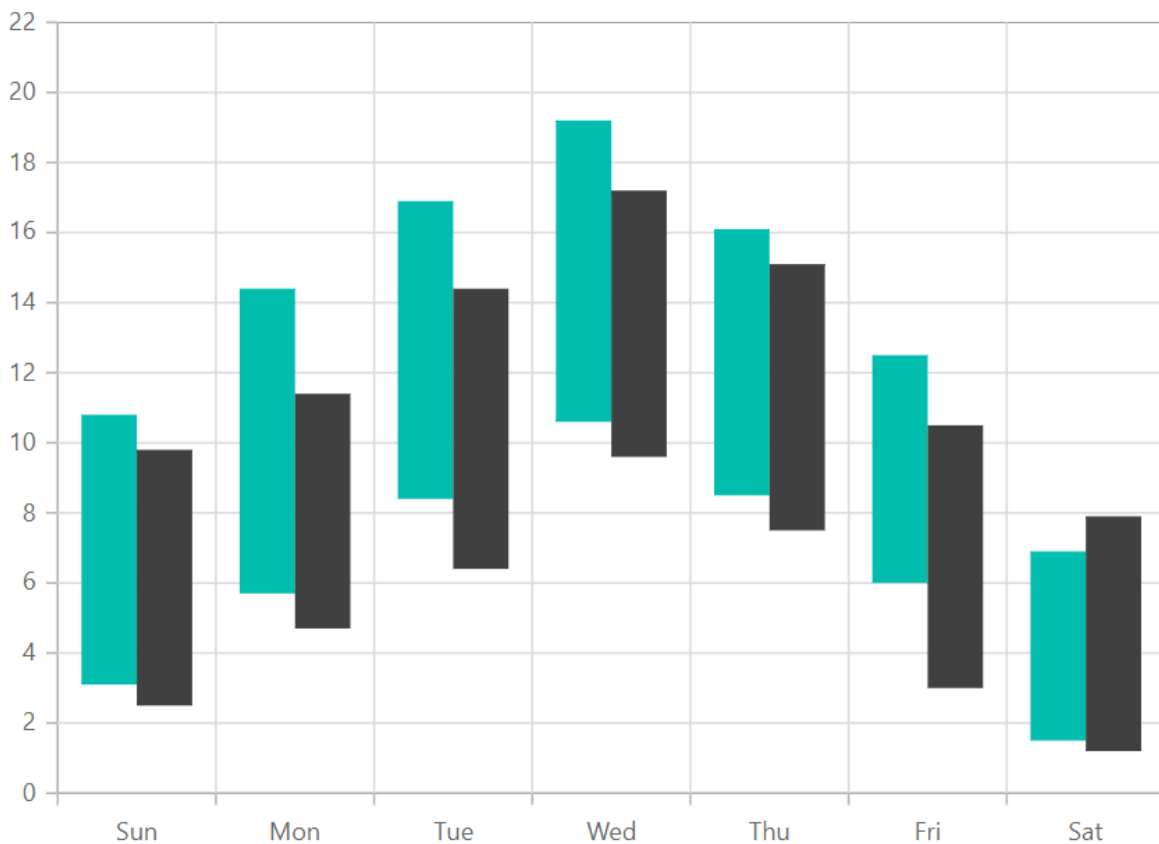
Range Column

[Range Column Chart](#) shows variation in the data values for a given time using vertical bars. To render a [Range Column Chart](#), set the series [Type](#) as [RangeColumn](#).

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart>
<ChartPrimaryXAxis
ValueTypes="Syncfusion.Blazor.Charts.ValueType.Category"></ChartPrimaryXAxis>
<ChartSeriesCollection>
<ChartSeries DataSource="@WeatherReport1" XName="X" High="High" Low="Low"
Width="2" Type="ChartSeriesType.RangeColumn">
</ChartSeries>
```

```
<ChartSeries DataSource="@WeatherReport2" XName="X" High="High" Low="Low"
Width="2" Type="ChartSeriesType.RangeColumn">
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code{
public class ChartData
{
public string X { get; set; }
public double Low { get; set; }
public double High { get; set; }
}
public List<ChartData> WeatherReport1 = new List<ChartData>
{
new ChartData { X= "Sun", Low= 3.1, High= 10.8 },
new ChartData { X= "Mon", Low= 5.7, High= 14.4 },
new ChartData { X= "Tue", Low= 8.4, High= 16.9 },
new ChartData { X= "Wed", Low= 10.6, High= 19.2 },
new ChartData { X= "Thu", Low= 8.5, High= 16.1 },
new ChartData { X= "Fri", Low= 6.0, High= 12.5 },
new ChartData { X= "Sat", Low= 1.5, High= 6.9 }
};
public List<ChartData> WeatherReport2 = new List<ChartData>
{
new ChartData { X= "Sun", Low= 2.5, High= 9.8 },
new ChartData { X= "Mon", Low= 4.7, High= 11.4 },
new ChartData { X= "Tue", Low= 6.4, High= 14.4 },
new ChartData { X= "Wed", Low= 9.6, High= 17.2 },
new ChartData { X= "Thu", Low= 7.5, High= 15.1 },
new ChartData { X= "Fri", Low= 3.0, High= 10.5 },
new ChartData { X= "Sat", Low= 1.2, High= 7.9 }
};
}
```



Refer to our [Blazor Range Column Chart](#) feature tour page to know about its other groundbreaking feature representations. Explore our [Blazor Range Column Chart Example](#) to know how to show variations in the data values for a given time.

Series Customization

The following properties can be used to customize the [Range Column](#) series.

- [Fill](#) – Specifies the color of the series.
- [Opacity](#) – Specifies the opacity of [Fill](#).
- [DashArray](#) – Specifies the dashes of series border.
- [ChartSeriesBorder](#) – Specifies the [Color](#) and [Width](#) of series border.
- [ColumnSpacing](#) – Specifies the space between the series segments.

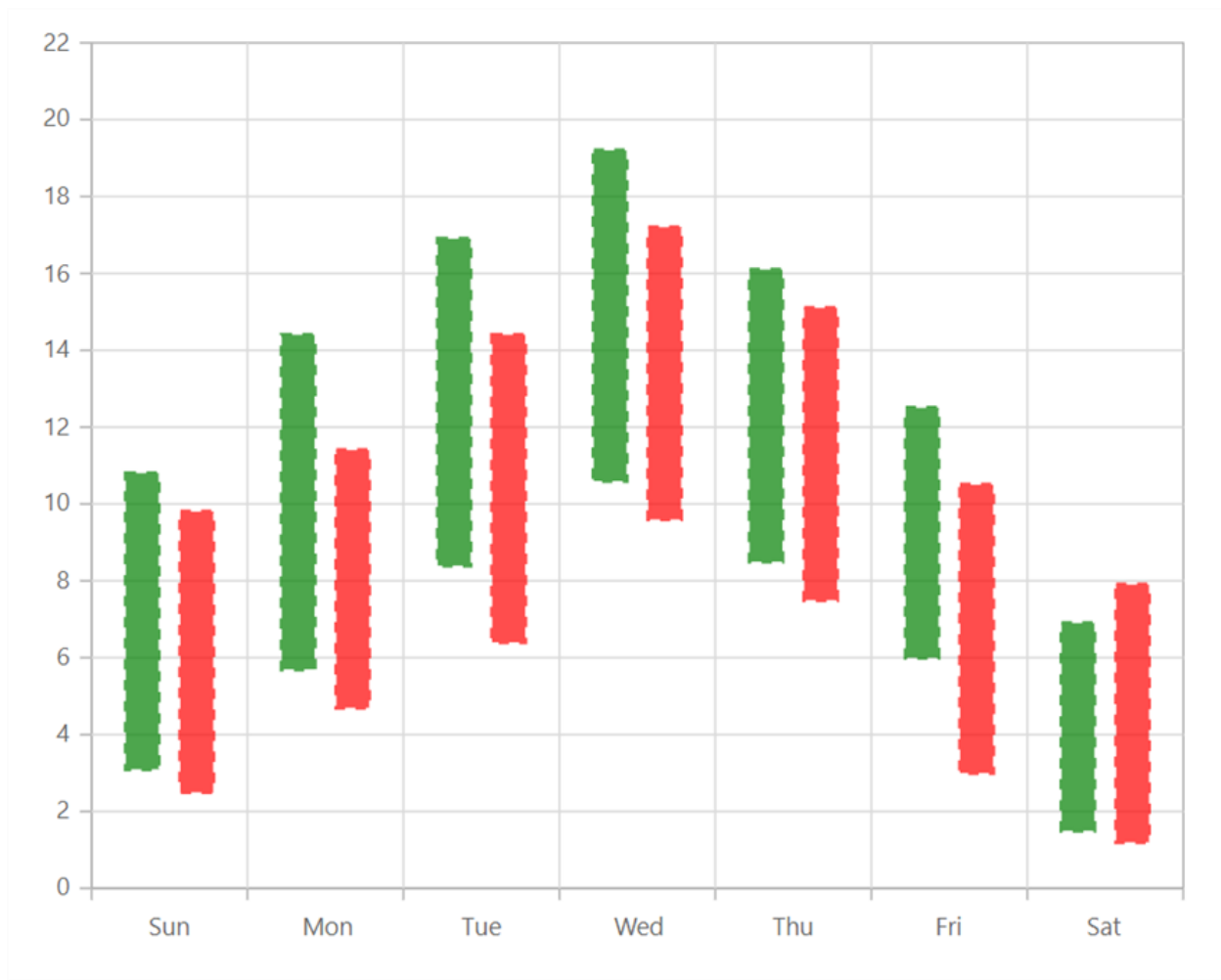
ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart>
<ChartPrimaryXAxis
Value="Syncfusion.Blazor.Charts.ValueType.Category"></ChartPrimaryXAxis>
<ChartSeriesCollection>
<ChartSeries DataSource="@WeatherReport1" XName="X" High="High" Low="Low"
Width="2" Type="ChartSeriesType.RangeColumn" Fill="green"
ColumnSpacing="0.4" DashArray="5,5" Opacity="0.7">
<ChartSeriesBorder Width="2" Color="green"></ChartSeriesBorder>
</ChartSeries>
```

```

<ChartSeries DataSource="@WeatherReport2" XName="X" High="High" Low="Low"
Width="2" Type="ChartSeriesType.RangeColumn" Fill="red" ColumnSpacing="0.4"
DashArray="5,5" Opacity="0.7">
<ChartSeriesBorder Width="2" Color="red"></ChartSeriesBorder>
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code{
public class ChartData
{
public string X { get; set; }
public double Low { get; set; }
public double High { get; set; }
}
public List<ChartData> WeatherReport1 = new List<ChartData>
{
new ChartData { X= "Sun", Low= 3.1, High= 10.8 },
new ChartData { X= "Mon", Low= 5.7, High= 14.4 },
new ChartData { X= "Tue", Low= 8.4, High= 16.9 },
new ChartData { X= "Wed", Low= 10.6, High= 19.2 },
new ChartData { X= "Thu", Low= 8.5, High= 16.1 },
new ChartData { X= "Fri", Low= 6.0, High= 12.5 },
new ChartData { X= "Sat", Low= 1.5, High= 6.9 }
};
public List<ChartData> WeatherReport2 = new List<ChartData>
{
new ChartData { X= "Sun", Low= 2.5, High= 9.8 },
new ChartData { X= "Mon", Low= 4.7, High= 11.4 },
new ChartData { X= "Tue", Low= 6.4, High= 14.4 },
new ChartData { X= "Wed", Low= 9.6, High= 17.2 },
new ChartData { X= "Thu", Low= 7.5, High= 15.1 },
new ChartData { X= "Fri", Low= 3.0, High= 10.5 },
new ChartData { X= "Sat", Low= 1.2, High= 7.9 }
};
}

```



Refer to our [Blazor Charts](#) feature tour page for its groundbreaking feature representations and also explore our [Blazor Chart Example](#) to know various chart types and how to represent time-dependent data, showing trends at equal intervals.

See Also

- [Data Label](#)
- [Tooltip](#)

Stacked Column in Blazor Charts Component

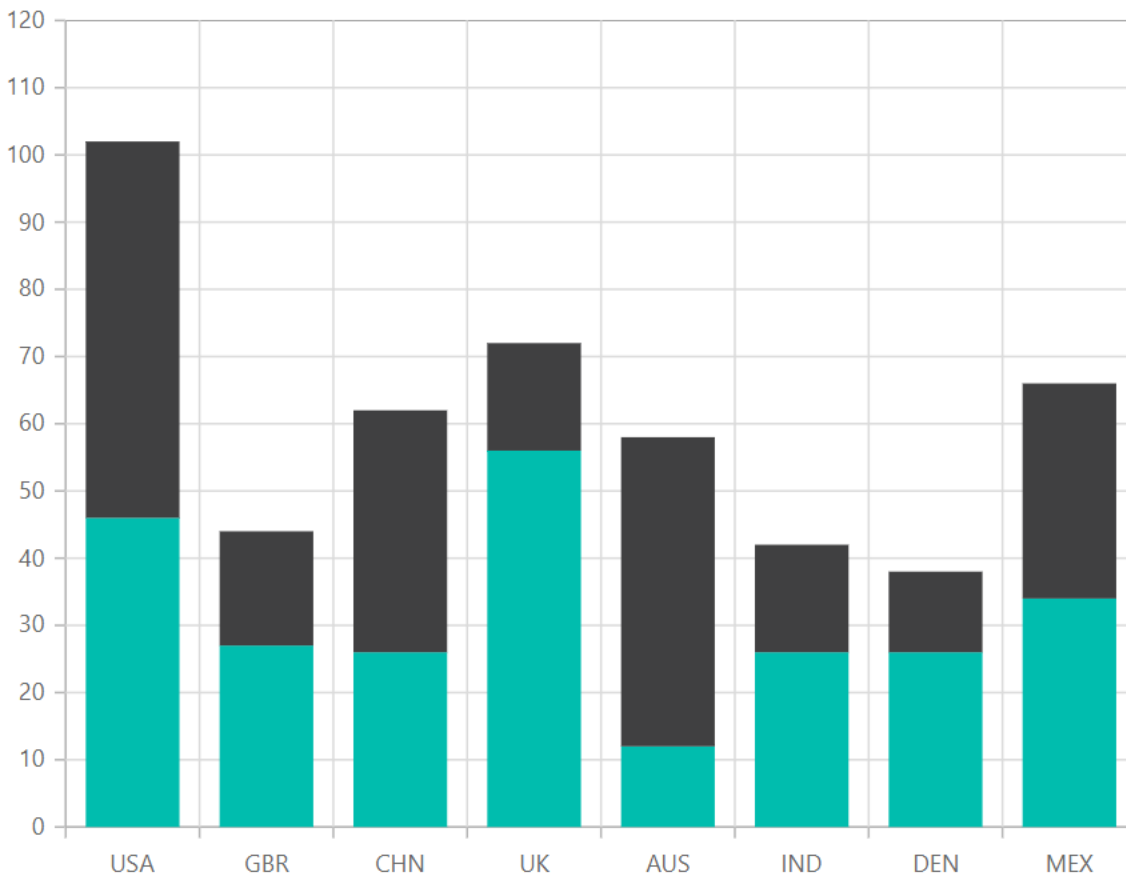
Stacked Column

[Stacked Column Chart](#) is a chart with Y values stacked over one another in the series order. It shows the relation between individual values to the total sum of the points. To render a [Stacked Column](#) series, set the series [Type](#) as [StackingColumn](#).

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart>
<ChartPrimaryXAxis
Value="Syncfusion.Blazor.Charts.ValueType.Category"></ChartPrimaryXAxis>
<ChartSeriesCollection>
```

```
<ChartSeries DataSource="@MedalDetails" XName="X" YName="YValue"
Type="ChartSeriesType.StackingColumn">
</ChartSeries>
<ChartSeries DataSource="@MedalDetails" XName="X" YName="YValue1"
Type="ChartSeriesType.StackingColumn">
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code{
public class ChartData
{
public string X { get; set; }
public double YValue { get; set; }
public double YValue1 { get; set; }
}
public List<ChartData> MedalDetails = new List<ChartData>
{
new ChartData { X= "USA", YValue= 46, YValue1=56 },
new ChartData { X= "GBR", YValue= 27, YValue1=17 },
new ChartData { X= "CHN", YValue= 26, YValue1=36 },
new ChartData { X= "UK", YValue= 56, YValue1=16 },
new ChartData { X= "AUS", YValue= 12, YValue1=46 },
new ChartData { X= "IND", YValue= 26, YValue1=16 },
new ChartData { X= "DEN", YValue= 26, YValue1=12 },
new ChartData { X= "MEX", YValue= 34, YValue1=32},
};
}
```

Refer to our [Blazor Stacked Column Chart](#) feature tour page to know about its other groundbreaking feature representations. Explore our [Blazor Stacked Column Chart Example](#) to know how to render and configure the Stacked Column type chart.

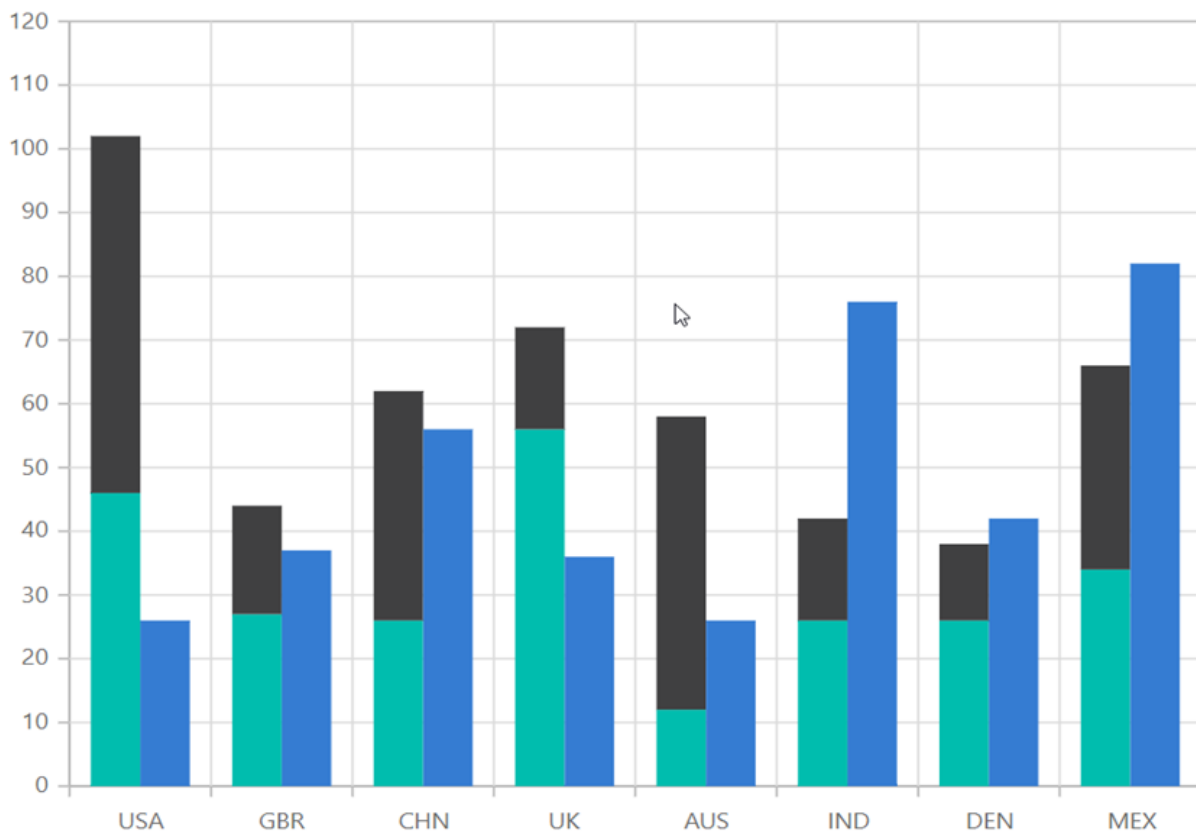
Stacking Group

The [StackingGroup](#) property is used to group stacked columns and 100% stacked columns. Columns with same group name are stacked on top of each other.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart>
<ChartPrimaryXAxis
ValueTypes="Syncfusion.Blazor.Charts.ValueType.Category"></ChartPrimaryXAxis>
<ChartSeriesCollection>
<ChartSeries DataSource="@DataSource" StackingGroup="Group1" XName="X"
YName="YValue" Type="ChartSeriesType.StackingColumn">
</ChartSeries>
<ChartSeries DataSource="@DataSource" StackingGroup="Group1" XName="X"
YName="YValue1" Type="ChartSeriesType.StackingColumn">
</ChartSeries>
<ChartSeries DataSource="@DataSource" StackingGroup="Group2" XName="X"
YName="YValue2" Type="ChartSeriesType.StackingColumn">
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
```

```
@code{
public class ChartData
{
    public string X { get; set; }
    public double YValue { get; set; }
    public double YValue1 { get; set; }
    public double YValue2 { get; set; }
}
public List<ChartData> DataSource = new List<ChartData>
{
    new ChartData { X= "USA", YValue= 46, YValue1=56, YValue2=26},
    new ChartData { X= "GBR", YValue= 27, YValue1=17, YValue2=37},
    new ChartData { X= "CHN", YValue= 26, YValue1=36, YValue2=56},
    new ChartData { X= "UK", YValue= 56, YValue1=16, YValue2=36},
    new ChartData { X= "AUS", YValue= 12, YValue1=46, YValue2=26},
    new ChartData { X= "IND", YValue= 26, YValue1=16, YValue2=76},
    new ChartData { X= "DEN", YValue= 26, YValue1=12, YValue2=42},
    new ChartData { X= "MEX", YValue= 34, YValue1=32, YValue2=82 },
};
}
```



Series Customization

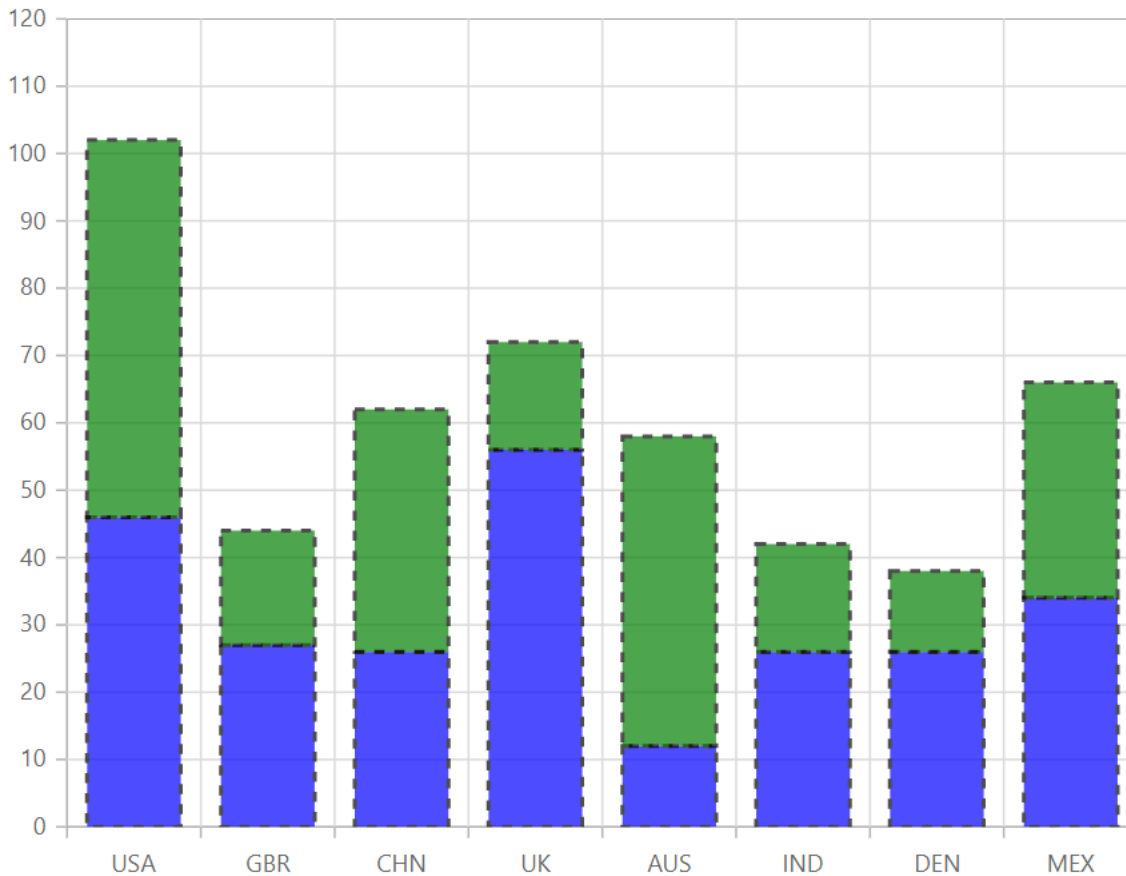
The following properties can be used to customize the [Stacked Column](#) series.

- [Fill](#) – Specifies the color of the series.
- [Opacity](#) – Specifies the opacity of [Fill](#).

- [DashArray](#) – Specifies the dashes of series border.
- [ChartSeriesBorder](#) – Specifies the [Color](#) and [Width](#) of series border.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart>
<ChartPrimaryXAxis
ValueType="Syncfusion.Blazor.Charts.ValueType.Category"></ChartPrimaryXAxis>
<ChartSeriesCollection>
<ChartSeries DataSource="@DataSource" XName="X" DashArray="5,5" Fill="blue"
Opacity="0.7" YName="YValue" Type="ChartSeriesType.StackingColumn">
<ChartSeriesBorder Width="2" Color="black"></ChartSeriesBorder>
</ChartSeries>
<ChartSeries DataSource="@DataSource" XName="X" DashArray="5,5" Fill="green"
Opacity="0.7" YName="YValue" Type="ChartSeriesType.StackingColumn">
<ChartSeriesBorder Width="2" Color="black"></ChartSeriesBorder>
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code{
public class ChartData
{
public string X { get; set; }
public double YValue { get; set; }
public double YValue1 { get; set; }
}
public List<ChartData> DataSource = new List<ChartData>
{
new ChartData { X= "USA", YValue= 46, YValue1=56 },
new ChartData { X= "GBR", YValue= 27, YValue1=17 },
new ChartData { X= "CHN", YValue= 26, YValue1=36 },
new ChartData { X= "UK", YValue= 56, YValue1=16 },
new ChartData { X= "AUS", YValue= 12, YValue1=46 },
new ChartData { X= "IND", YValue= 26, YValue1=16 },
new ChartData { X= "DEN", YValue= 26, YValue1=12 },
new ChartData { X= "MEX", YValue= 34, YValue1=32},
};
}
```



Refer to our [Blazor Charts](#) feature tour page for its groundbreaking feature representations and also explore our [Blazor Chart Example](#) to know various chart types and how to represent time-dependent data, showing trends at equal intervals.

See Also

- [Data Label](#)
- [Tooltip](#)

100% Stacked Column in Blazor Charts Component

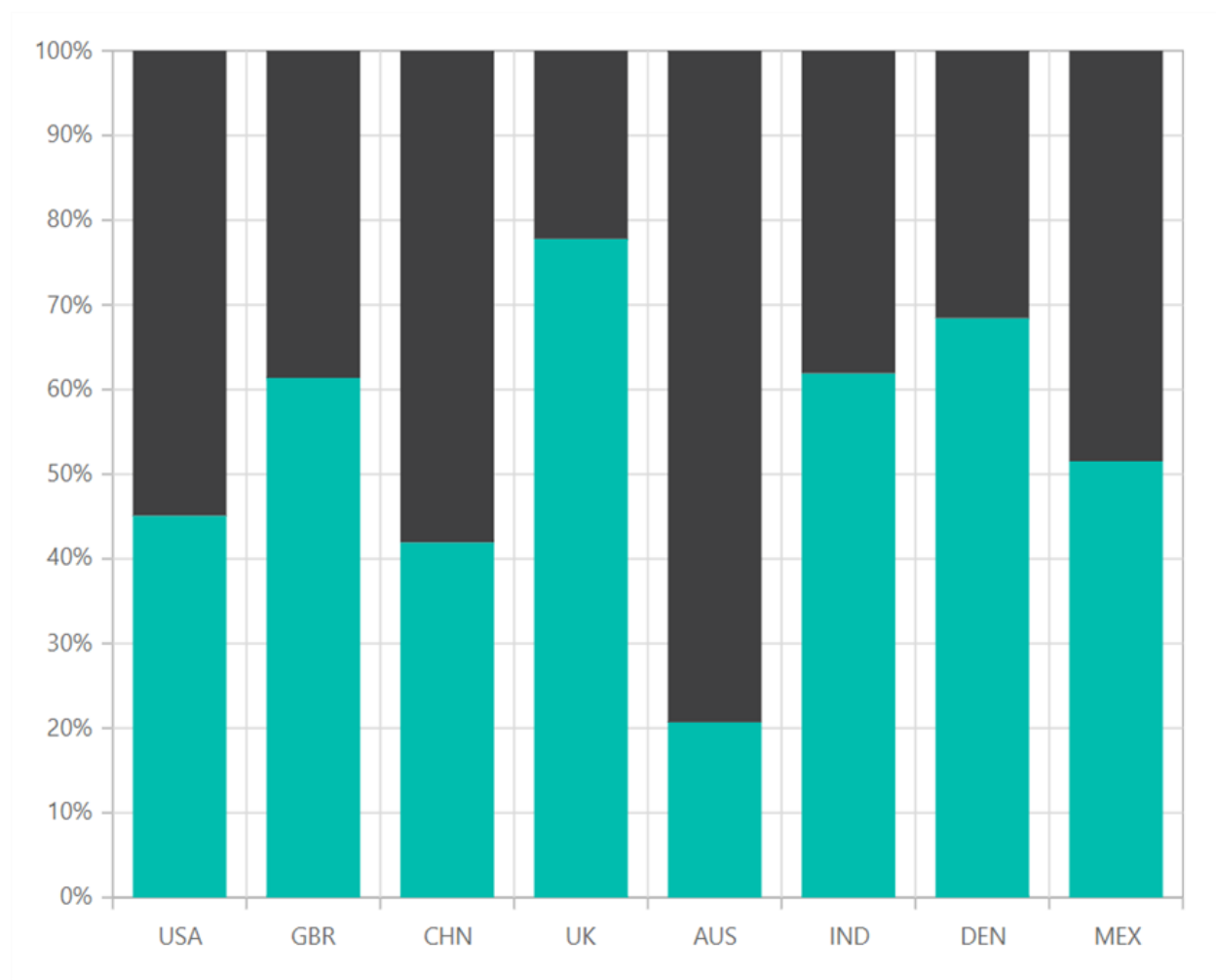
100% Stacked Column

[100% Stacked Column Chart](#) displays multiple series of data as stacked columns, ensuring that the cumulative proportion of each stacked element always totals 100%. Hence, the y-axis will always be rendered with the range 0–100%. To render a [100% Stacked Column](#) series, set the series [Type](#) as [StackingColumn100](#).

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart>
<ChartPrimaryXAxis ValueTypes="Syncfusion.Blazor.Charts.ValueType.Category"/>
<ChartSeriesCollection>
<ChartSeries DataSource="@MedalDetails" XName="X" YName="YValue"
Type="ChartSeriesType.StackingColumn100">
```

```
</ChartSeries>
<ChartSeries DataSource="@MedalDetails" XName="X" YName="YValue1"
Type="ChartSeriesType.StackingColumn100">
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code{
public class ChartData
{
public string X { get; set; }
public double YValue { get; set; }
public double YValue1 { get; set; }
}
public List<ChartData> MedalDetails = new List<ChartData>
{
new ChartData { X= "USA", YValue= 46, YValue1=56 },
new ChartData { X= "GBR", YValue= 27, YValue1=17 },
new ChartData { X= "CHN", YValue= 26, YValue1=36 },
new ChartData { X= "UK", YValue= 56, YValue1=16 },
new ChartData { X= "AUS", YValue= 12, YValue1=46 },
new ChartData { X= "IND", YValue= 26, YValue1=16 },
new ChartData { X= "DEN", YValue= 26, YValue1=12 },
new ChartData { X= "MEX", YValue= 34, YValue1=32},
};
}
```



Refer to our [Blazor 100% Stacked Column Chart](#) feature tour page to know about its other groundbreaking feature representations. Explore our [Blazor 100% Stacked Column Chart Example](#) to know how to render and configure the 100% Stacked Column type chart.

Series Customization

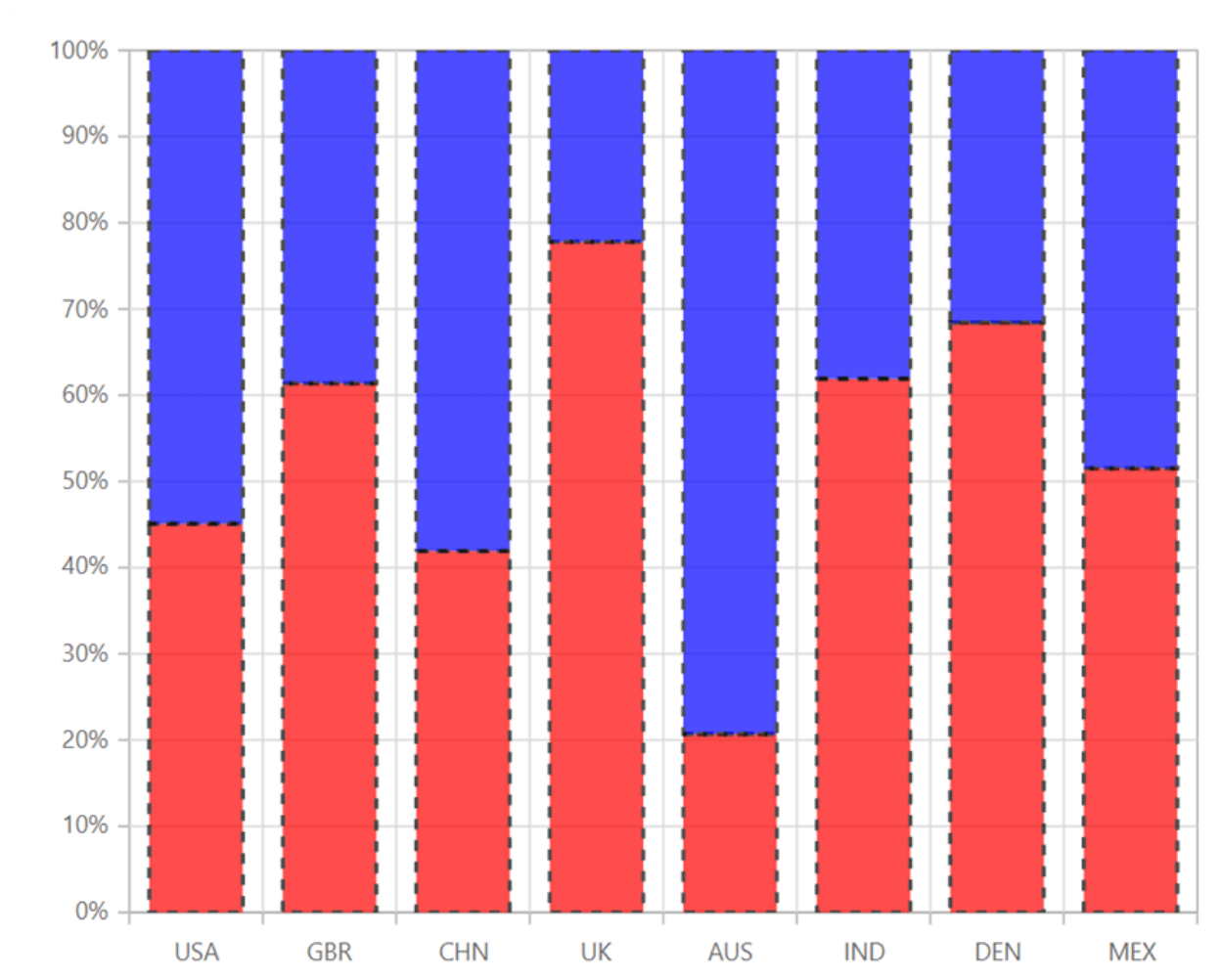
The following properties can be used to customize the [100% Stacked Column](#) series.

- [Fill](#) – Specifies the color of the series.
- [Opacity](#) – Specifies the opacity of [Fill](#).
- [DashArray](#) – Specifies the dashes of series border.
- [ChartSeriesBorder](#) – Specifies the [Color](#) and [Width](#) of series border.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart>
<ChartPrimaryXAxis
Value="Syncfusion.Blazor.Charts.ValueType.Category"></ChartPrimaryXAxis>
<ChartSeriesCollection>
<ChartSeries DataSource="@DataSource" StackingGroup="Asia" XName="X"
DashArray="5,5" Fill="red" Opacity="0.7" YName="YValue"
Type="ChartSeriesType.StackingColumn100">
```

```
<ChartSeriesBorder Width="2" Color="black"></ChartSeriesBorder>
</ChartSeries>
<ChartSeries DataSource="@DataSource" StackingGroup="Asia" XName="X"
DashArray="5,5" Fill="blue" Opacity="0.7" YName="YValue"
Type="ChartSeriesType.StackingColumn100">
<ChartSeriesBorder Width="2" Color="black"></ChartSeriesBorder>
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code{
public class ChartData
{
public string X { get; set; }
public double YValue { get; set; }
public double YValue1 { get; set; }
}
public List<ChartData> DataSource = new List<ChartData>
{
new ChartData { X= "USA", YValue= 46, YValue1=56 },
new ChartData { X= "GBR", YValue= 27, YValue1=17 },
new ChartData { X= "CHN", YValue= 26, YValue1=36 },
new ChartData { X= "UK", YValue= 56, YValue1=16 },
new ChartData { X= "AUS", YValue= 12, YValue1=46 },
new ChartData { X= "IND", YValue= 26, YValue1=16 },
new ChartData { X= "DEN", YValue= 26, YValue1=12 },
new ChartData { X= "MEX", YValue= 34, YValue1=32},
};
}
```



Refer to our [Blazor Charts](#) feature tour page for its groundbreaking feature representations and also explore our [Blazor Chart Example](#) to know various chart types and how to represent time-dependent data, showing trends at equal intervals.

See Also

- [Data Label](#)
- [Tooltip](#)

Bar Charts in Blazor Charts Component

Bar

[Bar Chart](#) is the most commonly used chart type to compare different categories of data, such as **Frequency**, **Count**, **Total**, or **Average** displayed in horizontal bars. It is ideal for showing variations in the value of an item over time. To render a bar series, set series [Type](#) as [Bar](#).

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart Title="Olympic Medals">
  <ChartPrimaryXAxis ValueTypes="Syncfusion.Blazor.Charts.ValueType.Category">
  </ChartPrimaryXAxis>
  <ChartSeriesCollection>
```



```

<ChartSeries DataSource="@MedalDetails" XName="Country" YName="Gold"
Type="ChartSeriesType.Bar">
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code{
public class ChartData
{
public string Country { get; set; }
public double Gold { get; set; }
}
public List<ChartData> MedalDetails = new List<ChartData>
{
new ChartData{ Country= "USA", Gold=50 },
new ChartData{ Country="China", Gold=40 },
new ChartData{ Country= "Japan", Gold=70 },
new ChartData{ Country= "Australia", Gold=60},
new ChartData{ Country= "France", Gold=50 },
new ChartData{ Country= "Germany", Gold=40 },
new ChartData{ Country= "Italy", Gold=40 },
new ChartData{ Country= "Sweden", Gold=30 }
};
}

```

Refer to our [Blazor Bar Charts](#) feature tour page to know about its other groundbreaking feature representations. Explore our [Blazor Bar Chart Example](#) to compare values across categories by using horizontal bars.

Bar Space and Width

The [ColumnSpacing](#) and [ColumnWidth](#) properties are used to customize the space between bars.

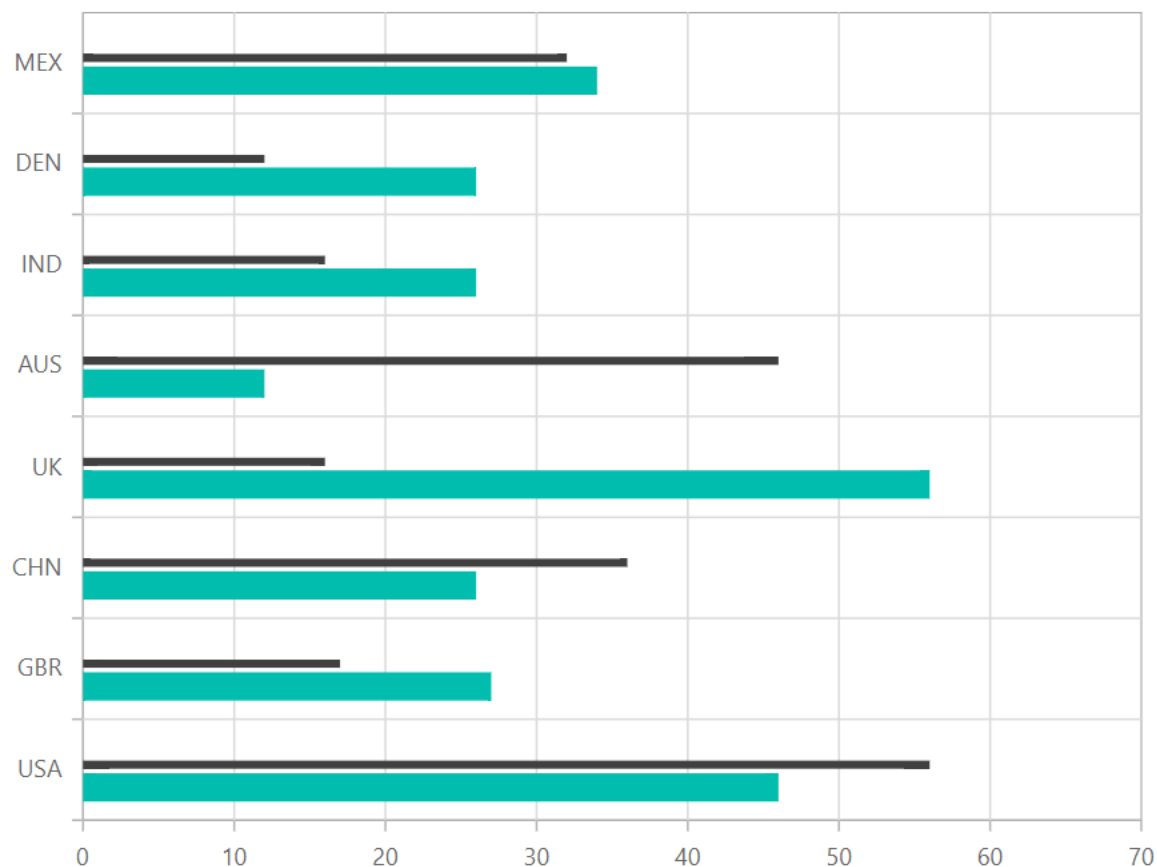
ASPX-CS

```

@using Syncfusion.Blazor.Charts
<SfChart Title="Olympic Medals">
<ChartPrimaryXAxis ValueType="Syncfusion.Blazor.Charts.ValueType.Category">
</ChartPrimaryXAxis>
<ChartSeriesCollection>
<ChartSeries DataSource="@MedalDetails" XName="Country" YName="Gold"
ColumnSpacing="0.2" ColumnWidth="0.7" Type="ChartSeriesType.Bar">
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code{
public class ChartData
{
public string Country { get; set; }
public double Gold { get; set; }
}
public List<ChartData> MedalDetails = new List<ChartData>
{
new ChartData{ Country= "USA", Gold=50 },
new ChartData{ Country="China", Gold=40 },
new ChartData{ Country= "Japan", Gold=70 },
new ChartData{ Country= "Australia", Gold=60},
new ChartData{ Country= "France", Gold=50 },

```

```
new ChartData{ Country= "Germany", Gold=40 },
new ChartData{ Country= "Italy", Gold=40 },
new ChartData{ Country= "Sweden", Gold=30 }
};
}
```



Series Customization

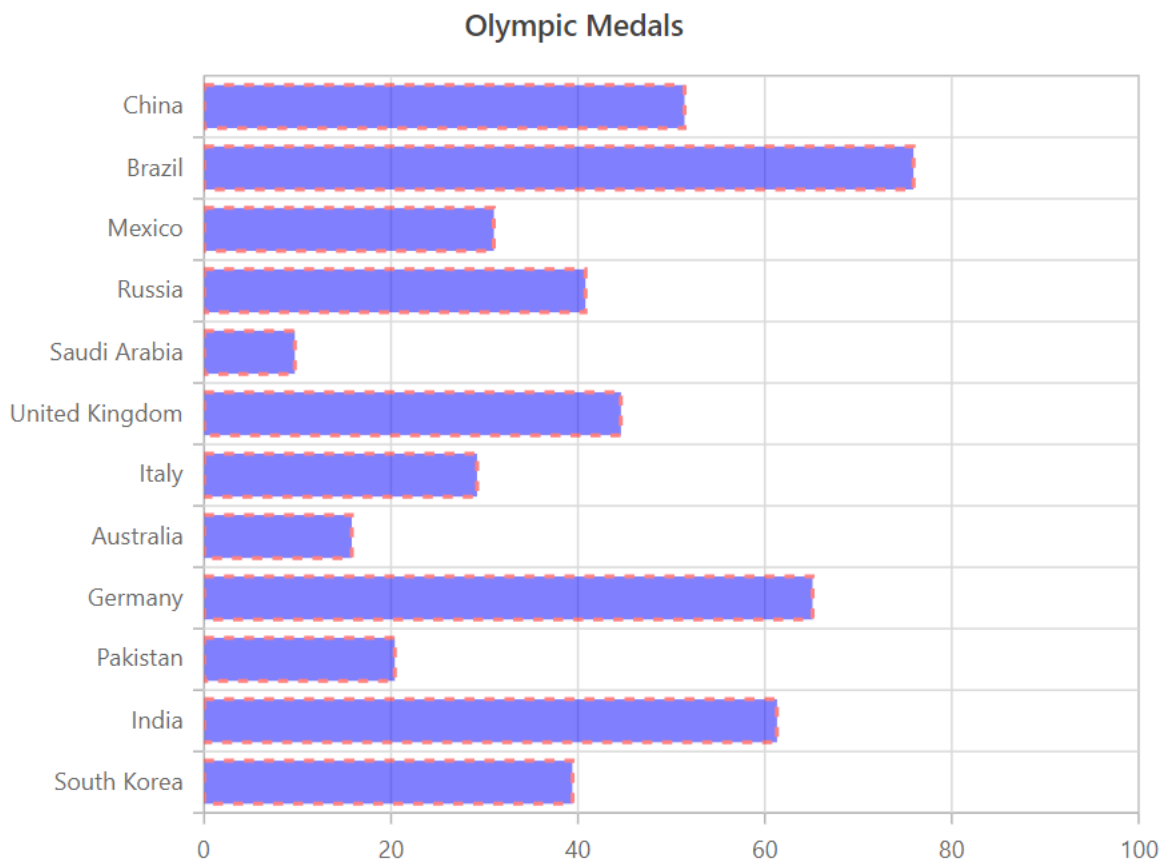
The following properties can be used to customize the [Bar](#) series.

- [Fill](#) – Specifies the color of the series.
- [Opacity](#) – Specifies the opacity of [Fill](#).
- [DashArray](#) – Specifies the dashes of series.
- [ChartSeriesBorder](#) – Specifies the [Color](#) and [Width](#) of series border.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart Title="Olympic Medals">
<ChartPrimaryXAxis ValueType="Syncfusion.Blazor.Charts.ValueType.Category">
</ChartPrimaryXAxis>
<ChartSeriesCollection>
<ChartSeries DataSource="@MedalDetails" XName="Country" YName="Gold"
Opacity="0.5" DashArray="5,5" Fill="blue" Type="ChartSeriesType.Bar">
</ChartSeries>
```

```
</ChartSeriesCollection>
</SfChart>
@code{
public class ChartData
{
public string Country { get; set; }
public double Gold { get; set; }
}
public List<ChartData> MedalDetails = new List<ChartData>
{
new ChartData{ Country= "USA", Gold=50 },
new ChartData{ Country="China", Gold=40 },
new ChartData{ Country= "Japan", Gold=70 },
new ChartData{ Country= "Australia", Gold=60},
new ChartData{ Country= "France", Gold=50 },
new ChartData{ Country= "Germany", Gold=40 },
new ChartData{ Country= "Italy", Gold=40 },
new ChartData{ Country= "Sweden", Gold=30 }
};
}
```



Refer to our [Blazor Charts](#) feature tour page for its groundbreaking feature representations and also explore our [Blazor Chart Example](#) to know various chart types and how to represent time-dependent data, showing trends at equal intervals.

See Also

- [Data Label](#)
- [Tooltip](#)

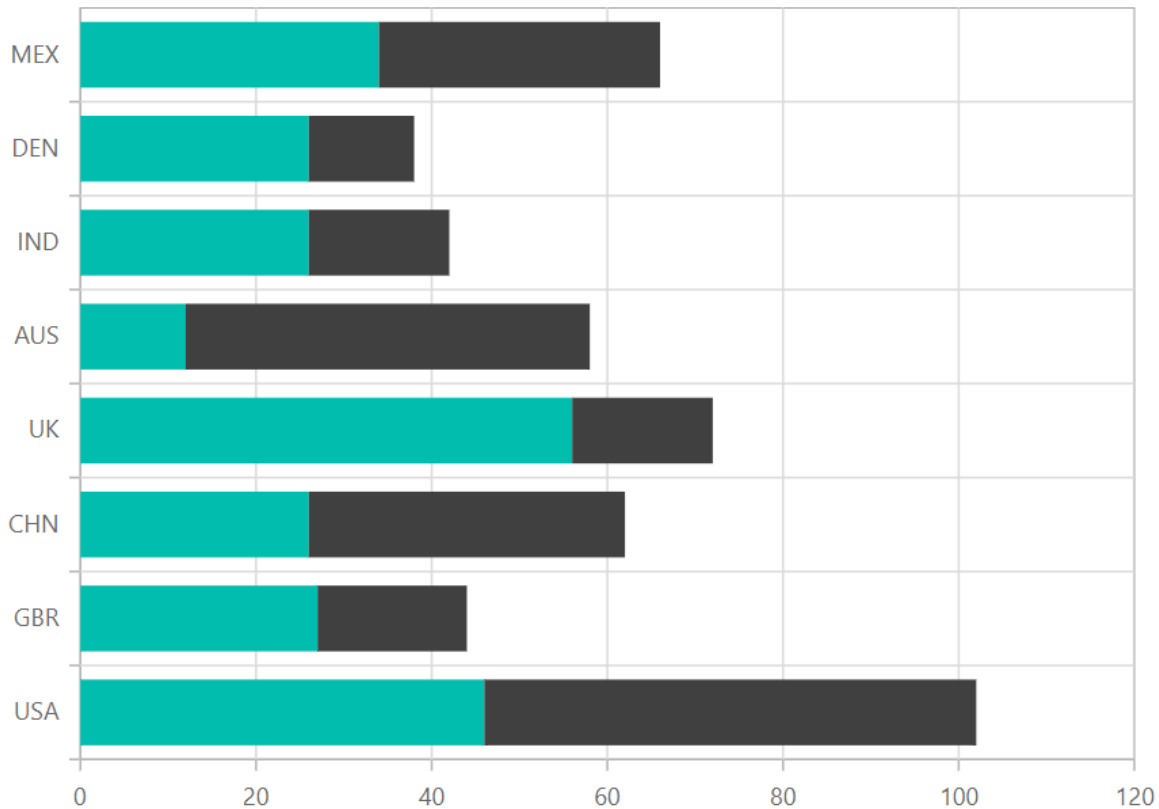
Stacked Bar in Blazor Charts Component

Stacked Bar

[Stacked Bar Chart](#) is a chart with Y values stacked over one another in the series order. It shows the relation between individual values to the total sum of the points. To render a [Stacked Bar](#) series, set the series [Type](#) as [StackingBar](#).

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart>
<ChartPrimaryXAxis
ValueType="Syncfusion.Blazor.Charts.ValueType.Category"></ChartPrimaryXAxis>
<ChartSeriesCollection>
<ChartSeries DataSource="@MedalDetails" XName="X" YName="YValue"
Type="ChartSeriesType.StackingBar">
</ChartSeries>
<ChartSeries DataSource="@MedalDetails" XName="X" YName="YValue1"
Type="ChartSeriesType.StackingBar">
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code{
public class ChartData
{
public string X { get; set; }
public double YValue { get; set; }
public double YValue1 { get; set; }
}
public List<ChartData> MedalDetails = new List<ChartData>
{
new ChartData { X= "USA", YValue= 46, YValue1=56 },
new ChartData { X= "GBR", YValue= 27, YValue1=17 },
new ChartData { X= "CHN", YValue= 26, YValue1=36 },
new ChartData { X= "UK", YValue= 56, YValue1=16 },
new ChartData { X= "AUS", YValue= 12, YValue1=46 },
new ChartData { X= "IND", YValue= 26, YValue1=16 },
new ChartData { X= "DEN", YValue= 26, YValue1=12 },
new ChartData { X= "MEX", YValue= 34, YValue1=32},
};
}
```



Refer to our [Blazor Stacked Bar Chart](#) feature tour page to know about its other groundbreaking feature representations. Explore our [Blazor Stacked Bar Chart Example](#) to know how to render and configure the Stacked Bar type chart.

Stacking Group

The [StackingGroup](#) property is used to group stacked bar and 100% stacked bar. Bars with same group name are stacked on top of each other.

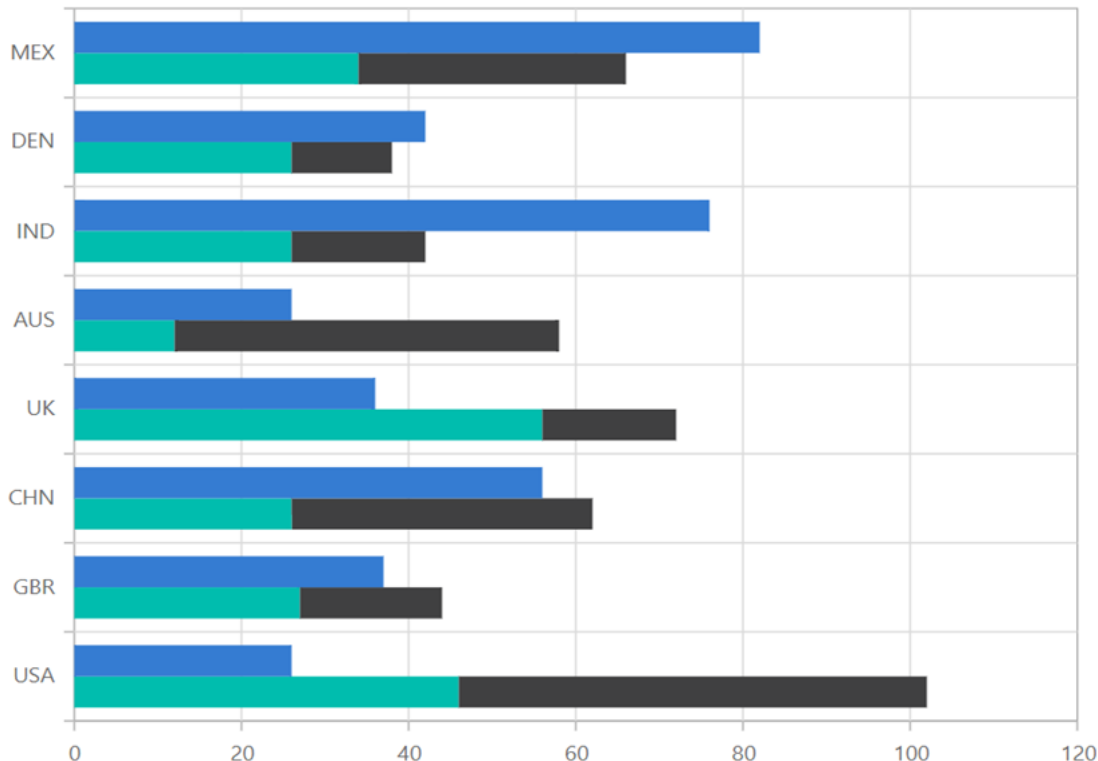
ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart>
<ChartPrimaryXAxis
ValueTypes="Syncfusion.Blazor.Charts.ValueType.Category"></ChartPrimaryXAxis>
<ChartSeriesCollection>
<ChartSeries DataSource="@DataSource" StackingGroup="Group1" XName="X"
YName="YValue" Type="ChartSeriesType.StackingBar">
</ChartSeries>
<ChartSeries DataSource="@DataSource" StackingGroup="Group1" XName="X"
YName="YValue1" Type="ChartSeriesType.StackingBar">
</ChartSeries>
<ChartSeries DataSource="@DataSource" StackingGroup="Group2" XName="X"
YName="YValue2" Type="ChartSeriesType.StackingBar">
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code{
public class ChartData
```

```

{
public string X { get; set; }
public double YValue { get; set; }
public double YValue1 { get; set; }
public double YValue2 { get; set; }
}
public List<ChartData> DataSource = new List<ChartData>
{
new ChartData { X= "USA", YValue= 46, YValue1=56, YValue2=26},
new ChartData { X= "GBR", YValue= 27, YValue1=17, YValue2=37},
new ChartData { X= "CHN", YValue= 26, YValue1=36, YValue2=56},
new ChartData { X= "UK", YValue= 56, YValue1=16, YValue2=36},
new ChartData { X= "AUS", YValue= 12, YValue1=46, YValue2=26},
new ChartData { X= "IND", YValue= 26, YValue1=16, YValue2=76},
new ChartData { X= "DEN", YValue= 26, YValue1=12, YValue2=42},
new ChartData { X= "MEX", YValue= 34, YValue1=32, YValue2=82 },
};
}

```



Series Customization

The following properties can be used to customize the [Stacked Bar](#) series.

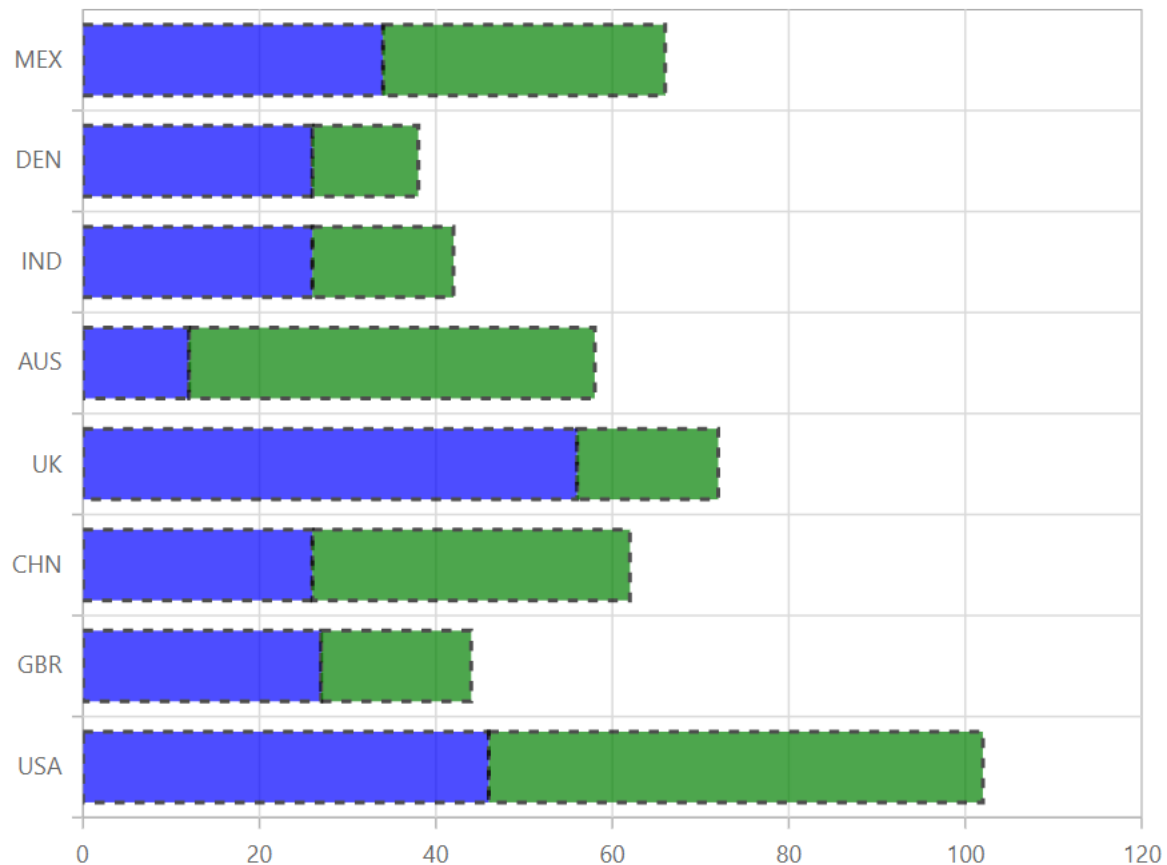
- [Fill](#) – Specifies the color of the series.
- [Opacity](#) – Specifies the opacity of [Fill](#).
- [DashArray](#) – Specifies the dashes of series border.
- [ChartSeriesBorder](#) – Specifies the [Color](#) and [Width](#) of series border.

ASPX-CS

```

@using Syncfusion.Blazor.Charts
<SfChart>
<ChartPrimaryXAxis
ValueType="Syncfusion.Blazor.Charts.ValueType.Category"></ChartPrimaryXAxis>
<ChartSeriesCollection>
<ChartSeries DataSource="@MedalDetails" XName="X" YName="YValue"
DashArray="5,5" Fill="blue" Opacity="0.7"
Type="ChartSeriesType.StackingBar">
<ChartSeriesBorder Width="2" Color="black"></ChartSeriesBorder>
</ChartSeries>
<ChartSeries DataSource="@MedalDetails" XName="X" YName="YValue1"
DashArray="5,5" Fill="green" Opacity="0.7"
Type="ChartSeriesType.StackingBar">
<ChartSeriesBorder Width="2" Color="black"></ChartSeriesBorder>
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code{
public class ChartData
{
public string X { get; set; }
public double YValue { get; set; }
public double YValue1 { get; set; }
}
public List<ChartData> MedalDetails = new List<ChartData>
{
new ChartData { X= "USA", YValue= 46, YValue1=56 },
new ChartData { X= "GBR", YValue= 27, YValue1=17 },
new ChartData { X= "CHN", YValue= 26, YValue1=36 },
new ChartData { X= "UK", YValue= 56, YValue1=16 },
new ChartData { X= "AUS", YValue= 12, YValue1=46 },
new ChartData { X= "IND", YValue= 26, YValue1=16 },
new ChartData { X= "DEN", YValue= 26, YValue1=12 },
new ChartData { X= "MEX", YValue= 34, YValue1=32},
};
}

```



Refer to our [Blazor Charts](#) feature tour page for its groundbreaking feature representations and also explore our [Blazor Chart Example](#) to know various chart types and how to represent time-dependent data, showing trends at equal intervals.

See Also

- [Data Label](#)
- [Tooltip](#)

100% Stacked Bar in Blazor Charts Component

100% Stacked Bar

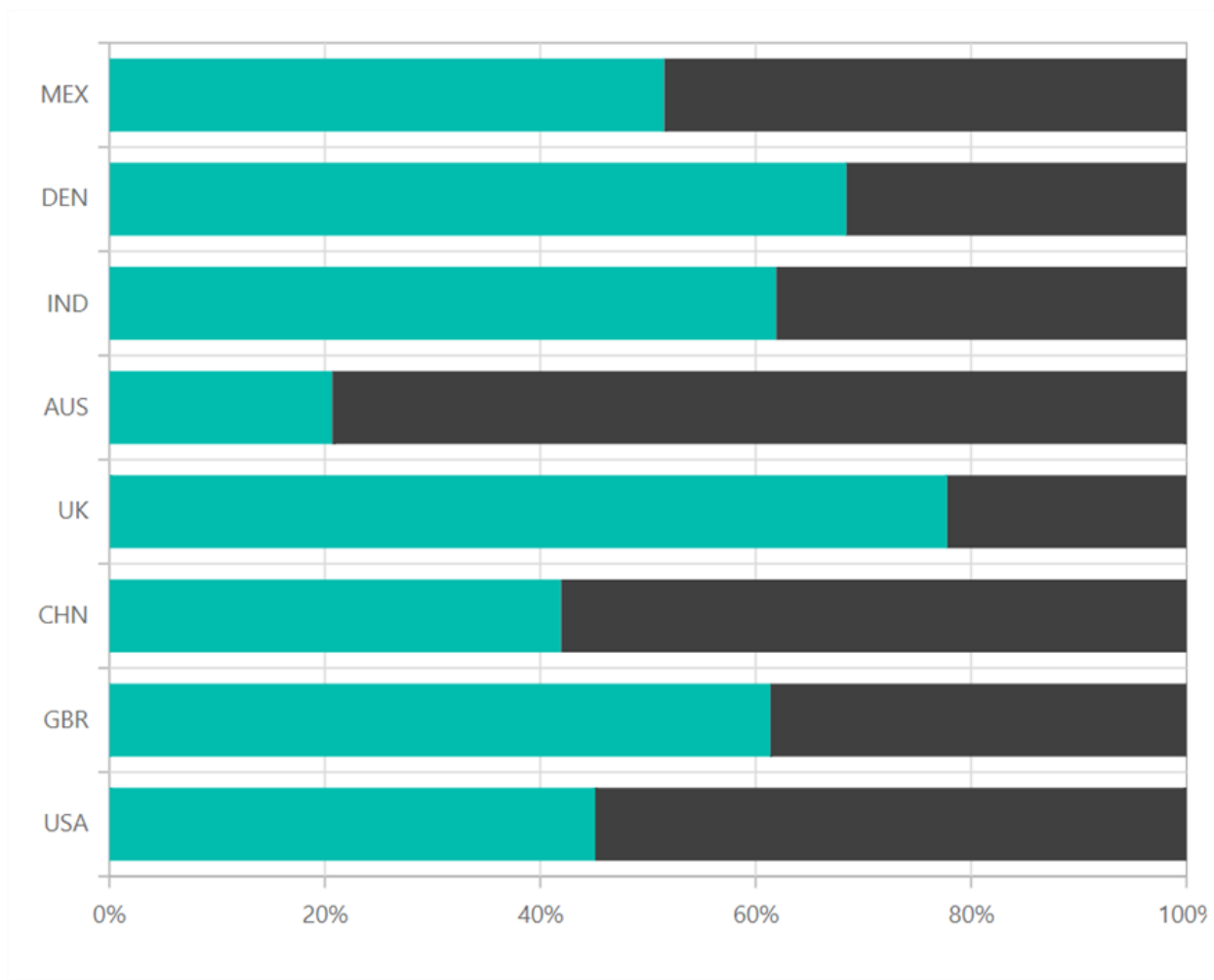
[100% Stacked Bar Chart](#) displays multiple series of data as stacked bars, ensuring that the cumulative proportion of each stacked element always totals 100%. Hence, the y-axis will always be rendered with the range 0–100%. To render a [100% Stacked Bar](#) series, set the series [Type](#) as [StackingBar100](#).

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart>
<ChartPrimaryXAxis
Value="Syncfusion.Blazor.Charts.ValueType.Category"></ChartPrimaryXAxis>
<ChartSeriesCollection>
<ChartSeries DataSource="@MedalDetails" XName="X" YName="YValue"
Type="ChartSeriesType.StackingBar100">
</ChartSeries>
```



```
<ChartSeries DataSource="@MedalDetails" XName="X" YName="YValue1"
Type="ChartSeriesType.StackingBar100">
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code{
public class ChartData
{
public string X { get; set; }
public double YValue { get; set; }
public double YValue1 { get; set; }
}
public List<ChartData> MedalDetails = new List<ChartData>
{
new ChartData { X= "USA", YValue= 46, YValue1=56 },
new ChartData { X= "GBR", YValue= 27, YValue1=17 },
new ChartData { X= "CHN", YValue= 26, YValue1=36 },
new ChartData { X= "UK", YValue= 56, YValue1=16 },
new ChartData { X= "AUS", YValue= 12, YValue1=46 },
new ChartData { X= "IND", YValue= 26, YValue1=16 },
new ChartData { X= "DEN", YValue= 26, YValue1=12 },
new ChartData { X= "MEX", YValue= 34, YValue1=32},
};
}
```



Refer to our [Blazor 100% Stacked Bar Chart](#) feature tour page to know about its other groundbreaking feature representations. You can also explore our [Blazor 100% Stacked Bar Chart Example](#) to know how to render and configure the 100% Stacked Bar type chart.

Series Customization

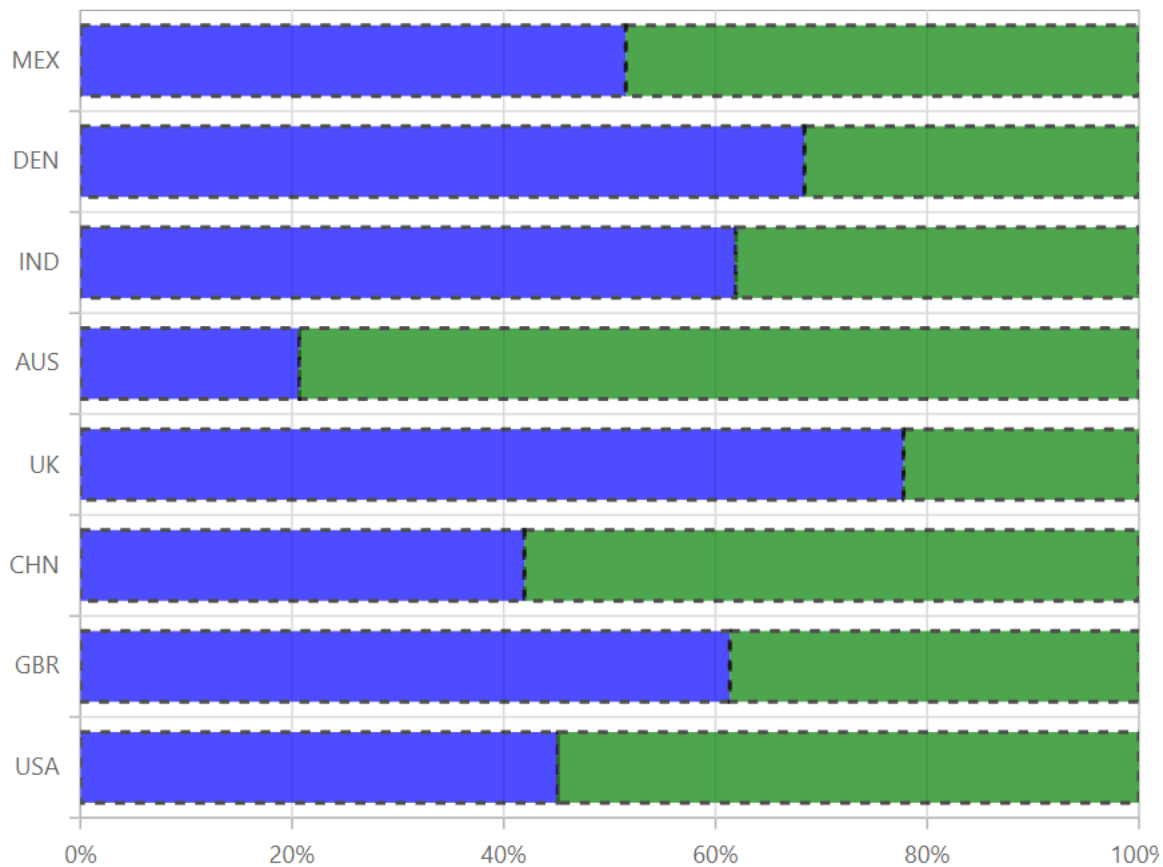
The following properties can be used to customize the [100% Stacked Bar](#) series.

- [Fill](#) – Specifies the color of the series.
- [Opacity](#) – Specifies the opacity of [Fill](#).
- [DashArray](#) – Specifies the dashes of series border.
- [ChartSeriesBorder](#) – Specifies the [Color](#) and [Width](#) of series border.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart>
<ChartPrimaryXAxis
Value="Syncfusion.Blazor.Charts.ValueType.Category"></ChartPrimaryXAxis>
<ChartSeriesCollection>
<ChartSeries DataSource="@MedalDetails" XName="X" YName="YValue"
DashArray="5,5" Fill="blue" Opacity="0.7"
Type="ChartSeriesType.StackingBar100">
```

```
<ChartSeriesBorder Width="2" Color="black"></ChartSeriesBorder>
</ChartSeries>
<ChartSeries DataSource="@MedalDetails" XName="X" YName="YValue1"
DashArray="5,5" Fill="green" Opacity="0.7"
Type="ChartSeriesType.StackingBar100">
<ChartSeriesBorder Width="2" Color="black"></ChartSeriesBorder>
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code{
public class ChartData
{
public string X { get; set; }
public double YValue { get; set; }
public double YValue1 { get; set; }
}
public List<ChartData> MedalDetails = new List<ChartData>
{
new ChartData { X= "USA", YValue= 46, YValue1=56 },
new ChartData { X= "GBR", YValue= 27, YValue1=17 },
new ChartData { X= "CHN", YValue= 26, YValue1=36 },
new ChartData { X= "UK", YValue= 56, YValue1=16 },
new ChartData { X= "AUS", YValue= 12, YValue1=46 },
new ChartData { X= "IND", YValue= 26, YValue1=16 },
new ChartData { X= "DEN", YValue= 26, YValue1=12 },
new ChartData { X= "MEX", YValue= 34, YValue1=32},
};
}
```



Refer to our [Blazor Charts](#) feature tour page for its groundbreaking feature representations and also explore our [Blazor Chart Example](#) to know various chart types and how to represent time-dependent data, showing trends at equal intervals.

See Also

- [Data Label](#)
- [Tooltip](#)

Scatter in Blazor Charts Component

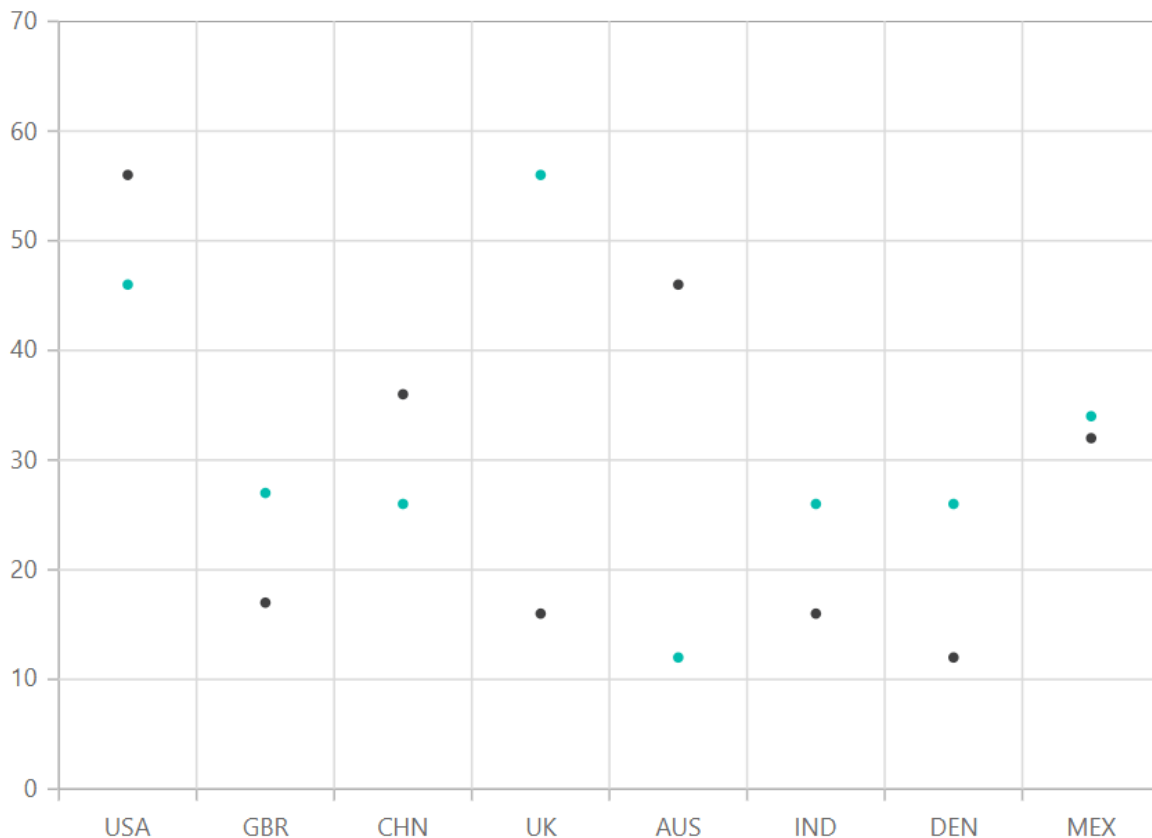
Scatter

[Scatter Chart](#) is used to visualize the relationship between two Cartesian parameters. To render a [Scatter Chart](#), set the series [Type](#) as [Scatter](#).

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart DataSource="@MedalDetails">
  <ChartPrimaryXAxis
    ValueType="Syncfusion.Blazor.Charts.ValueType.Category"></ChartPrimaryXAxis>
  <ChartSeriesCollection>
    <ChartSeries XName="X" YName="YValue" Type="ChartSeriesType.Scatter">
    </ChartSeries>
    <ChartSeries XName="X" YName="YValue1" Type="ChartSeriesType.Scatter">
    </ChartSeries>
  </ChartSeriesCollection>
</SfChart>
```

```
</ChartSeriesCollection>
</SfChart>
@code{
public class ChartData
{
    public string X { get; set; }
    public double YValue { get; set; }
    public double YValue1 { get; set; }
}
public List<ChartData> MedalDetails = new List<ChartData>
{
    new ChartData { X= "USA", YValue= 46, YValue1=56 },
    new ChartData { X= "GBR", YValue= 27, YValue1=17 },
    new ChartData { X= "CHN", YValue= 26, YValue1=36 },
    new ChartData { X= "UK", YValue= 56, YValue1=16 },
    new ChartData { X= "AUS", YValue= 12, YValue1=46 },
    new ChartData { X= "IND", YValue= 26, YValue1=16 },
    new ChartData { X= "DEN", YValue= 26, YValue1=12 },
    new ChartData { X= "MEX", YValue= 34, YValue1=32},
};
}
```



Refer to our [Blazor Scatter Chart](#) feature tour page to know about its other groundbreaking feature representations. Explore our [Blazor Scatter Chart Example](#) to know how to plot data with two numeric parameters.

Series Customization

The following properties can be used to customize the [Scatter](#) series.

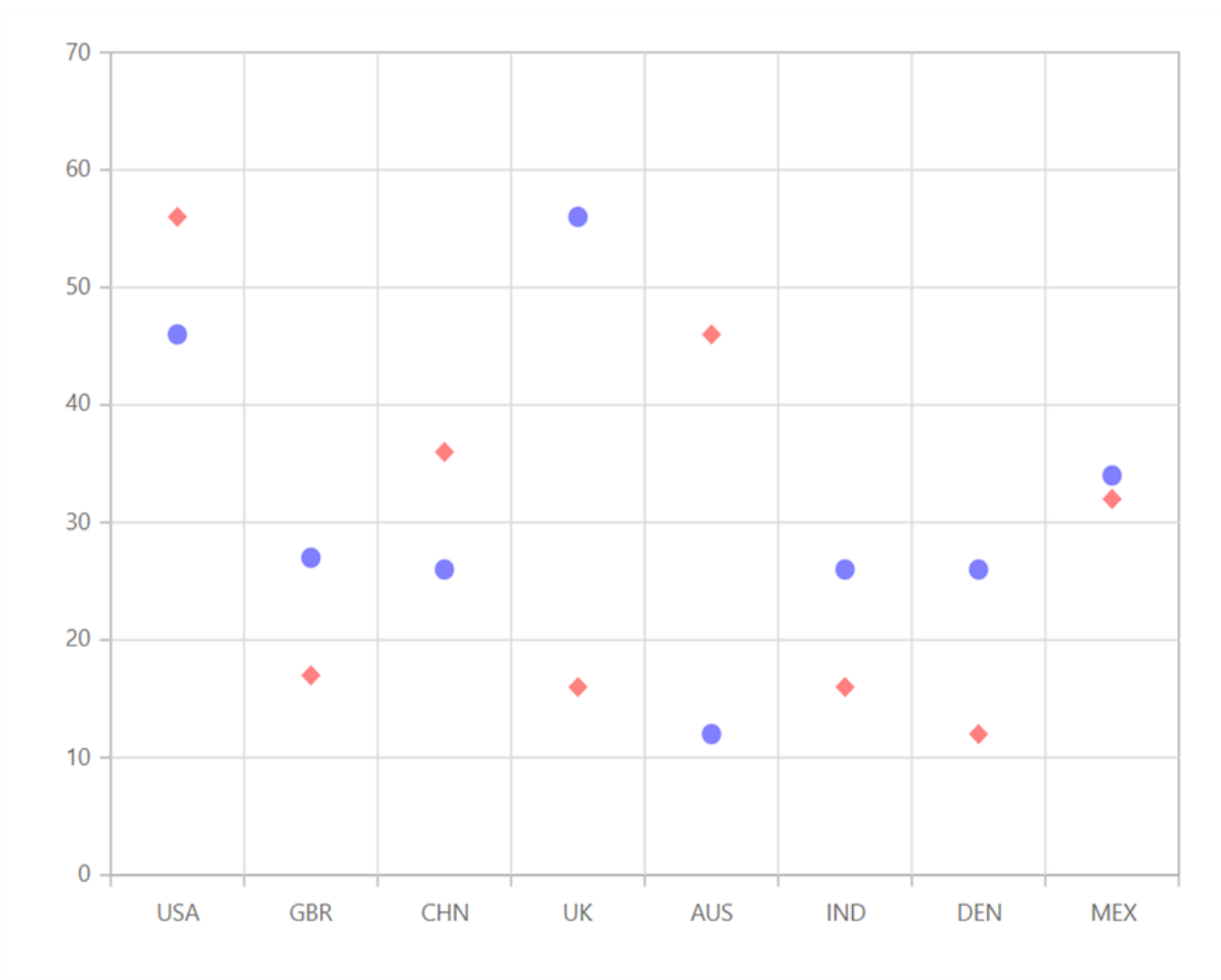
- [Fill](#) – Specifies the color of the series.
- [Opacity](#) – Specifies the opacity of [Fill](#).
- [Shape](#) - Specifies the shape of the scatter series.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart DataSource="@MedalDetails">
  <ChartPrimaryXAxis
    ValueType="Syncfusion.Blazor.Charts.ValueType.Category"></ChartPrimaryXAxis>
  <ChartSeriesCollection>
    <ChartSeries XName="X" YName="YValue" Type="ChartSeriesType.Scatter"
      Fill="blue" Opacity="0.5">
      <ChartMarker Height="10" Width="10" Shape="ChartShape.Circle">
      </ChartMarker>
    </ChartSeries>
    <ChartSeries XName="X" YName="YValue1" Type="ChartSeriesType.Scatter"
      Fill="red" Opacity="0.5">
      <ChartMarker Height="10" Width="10" Shape="ChartShape.Diamond">
      </ChartMarker>
    </ChartSeries>
  </ChartSeriesCollection>
</SfChart>

@code{
public class ChartData
{
    public string X { get; set; }
    public double YValue { get; set; }
    public double YValue1 { get; set; }
}

public List<ChartData> MedalDetails = new List<ChartData>
{
    new ChartData { X= "USA", YValue= 46, YValue1=56 },
    new ChartData { X= "GBR", YValue= 27, YValue1=17 },
    new ChartData { X= "CHN", YValue= 26, YValue1=36 },
    new ChartData { X= "UK", YValue= 56, YValue1=16 },
    new ChartData { X= "AUS", YValue= 12, YValue1=46 },
    new ChartData { X= "IND", YValue= 26, YValue1=16 },
    new ChartData { X= "DEN", YValue= 26, YValue1=12 },
    new ChartData { X= "MEX", YValue= 34, YValue1=32},
};
}
```



Refer to our [Blazor Charts](#) feature tour page for its groundbreaking feature representations and also explore our [Blazor Chart Example](#) to know various chart types and how to represent time-dependent data, showing trends at equal intervals.

See Also

- [Data Label](#)
- [Tooltip](#)

Bubble in Blazor Charts Component

Bubble

[Bubble Chart](#) is similar to the Scatter chart but it also visualizes the third parameter by its size. To render a bubble series, set series [Type](#) as [Bubble](#). It visualizes data with three parameters such as [XName](#), [YName](#) and [Size](#). The bubble size depends on third parameter.

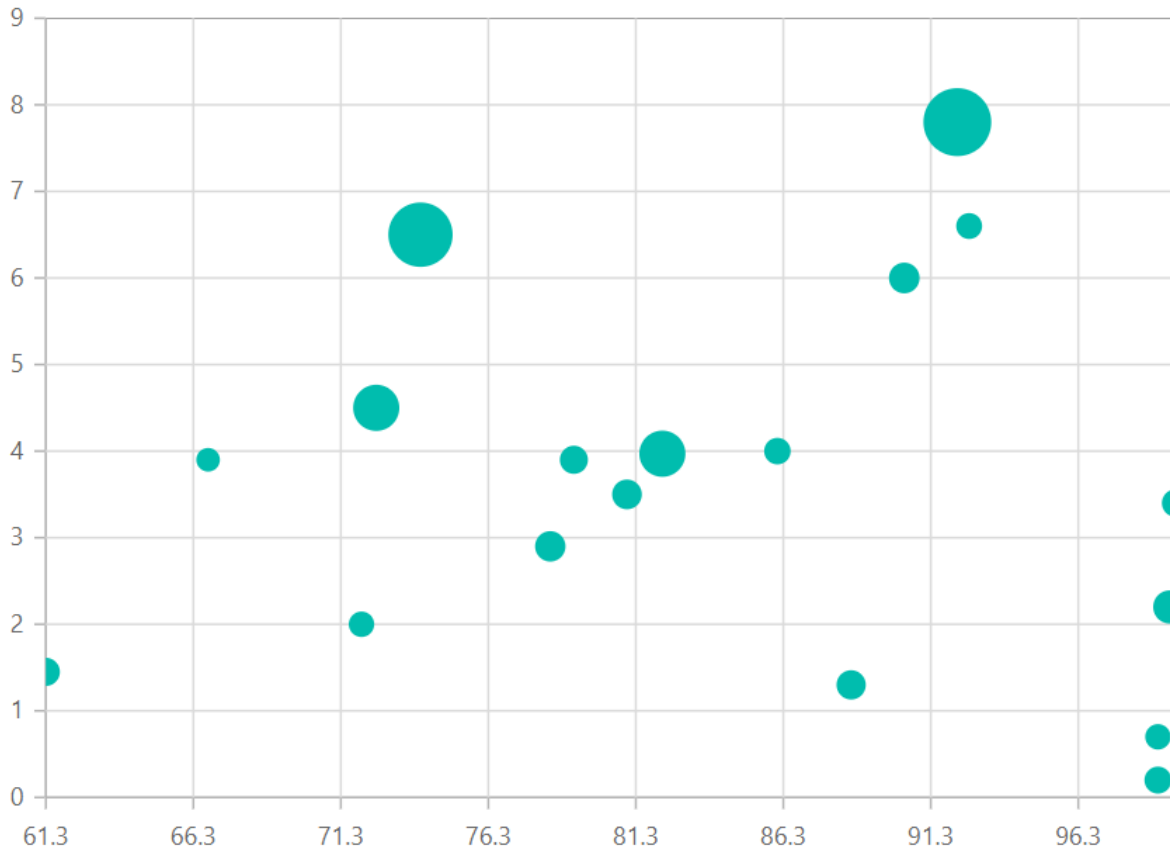
Refer to our [Blazor Bubble Charts](#) feature tour page to know about its other groundbreaking feature representations and also explore our [Blazor Bubble Chart Example](#) to know how to render and configure the bubble type charts.

Size Mapping

The [Size](#) property can be used to map the size value specified from datasource.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart>
<ChartSeriesCollection>
<ChartSeries DataSource="@SalesReports" XName="X" YName="Y"
Type="ChartSeriesType.Bubble">
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code{
public class ChartData
{
public double X { get; set; }
public double Y { get; set; }
public string Text { get; set; }
}
public List<ChartData> SalesReports = new List<ChartData>
{
new ChartData { X= 92.2, Y= 7.8, Text= "China" },
new ChartData { X= 74, Y= 6.5, Text= "India" },
new ChartData { X= 90.4, Y= 6.0, Text= "Indonesia" },
new ChartData { X= 99.4, Y= 2.2, Text= "US" },
new ChartData { X= 88.6, Y= 1.3, Text= "Brazil" },
new ChartData { X= 99, Y= 0.7, Text= "Germany" },
new ChartData { X= 72, Y= 2.0, Text= "Egypt" },
new ChartData { X= 99.6, Y= 3.4, Text= "Russia" },
new ChartData { X= 99, Y= 0.2, Text= "Japan" },
new ChartData { X= 86.1, Y= 4.0, Text= "Mexico" },
new ChartData { X= 92.6, Y= 6.6, Text= "Philippines" },
new ChartData { X= 61.3, Y= 1.45, Text= "Nigeria" },
new ChartData { X= 82.2, Y= 3.97, Text= "Hong Kong" },
new ChartData { X= 79.2, Y= 3.9, Text= "Netherland" },
new ChartData { X= 72.5, Y= 4.5, Text= "Jordan" },
new ChartData { X= 81, Y= 3.5, Text= "Australia" },
new ChartData { X= 66.8, Y= 3.9, Text= "Mongolia" },
new ChartData { X= 78.4, Y= 2.9, Text= "Taiwan" }
};
}
```

Series Customization

The following properties can be used to customize the [Bubble](#) series.

- [Fill](#) – Specifies the color of series.
- [Opacity](#) – Specifies the opacity of [Fill](#).

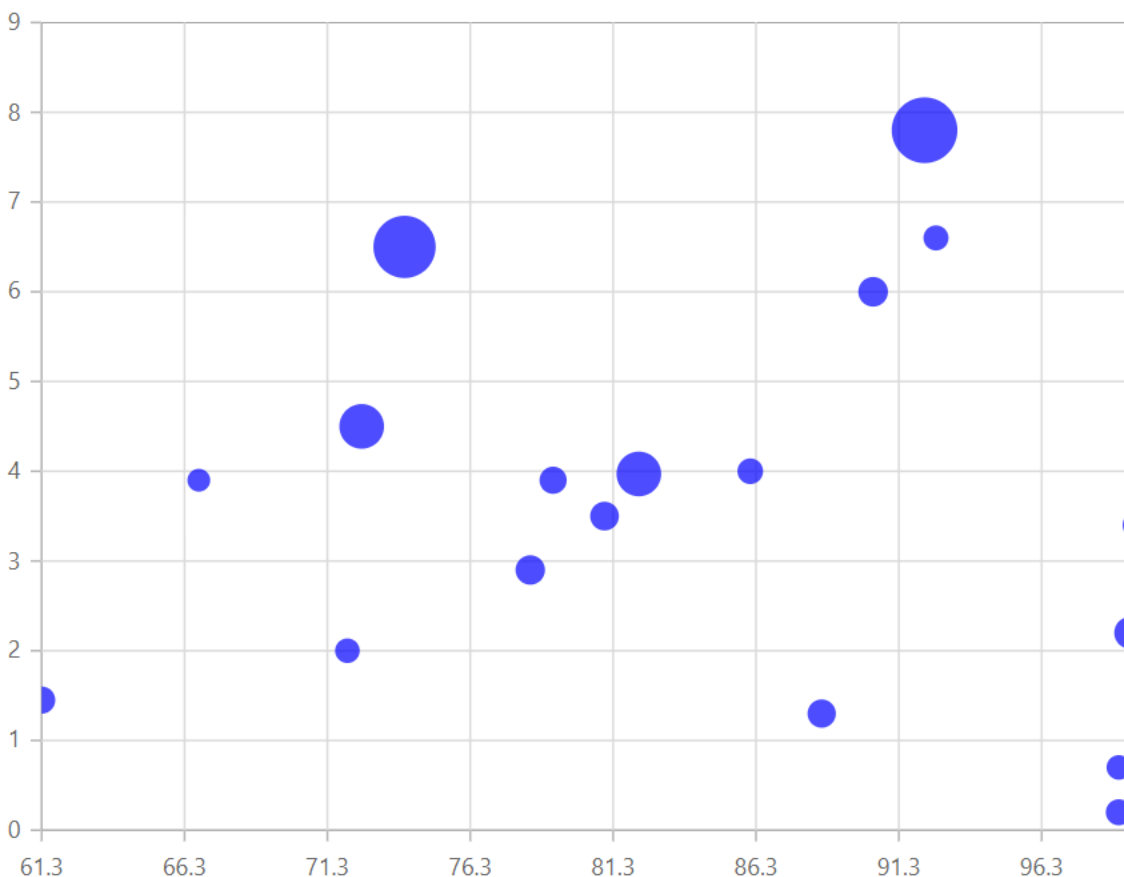
ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart>
<ChartSeriesCollection>
<ChartSeries DataSource="@SalesReports" XName="X" YName="Y" Opacity="0.7"
Fill="blue" Size="Size" Type="ChartSeriesType.Bubble">
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code{
public class ChartData
{
public double X { get; set; }
public double Y { get; set; }
public double Size { get; set; }
public string Text { get; set; }
}
public List<ChartData> SalesReports = new List<ChartData>
{
```

```

new ChartData { X= 92.2, Y= 7.8, Size= 1.347, Text= "China" },
new ChartData { X= 74, Y= 6.5, Size= 1.241, Text= "India" },
new ChartData { X= 90.4, Y= 6.0, Size= 0.238, Text= "Indonesia" },
new ChartData { X= 99.4, Y= 2.2, Size= 0.312, Text= "US" },
new ChartData { X= 88.6, Y= 1.3, Size= 0.197, Text= "Brazil" },
new ChartData { X= 99, Y= 0.7, Size= 0.0818, Text= "Germany" },
new ChartData { X= 72, Y= 2.0, Size= 0.0826, Text= "Egypt" },
new ChartData { X= 99.6, Y= 3.4, Size= 0.143, Text= "Russia" },
new ChartData { X= 99, Y= 0.2, Size= 0.128, Text= "Japan" },
new ChartData { X= 86.1, Y= 4.0, Size= 0.115, Text= "Mexico" },
new ChartData { X= 92.6, Y= 6.6, Size= 0.096, Text= "Philippines" },
new ChartData { X= 61.3, Y= 1.45, Size= 0.162, Text= "Nigeria" },
new ChartData { X= 82.2, Y= 3.97, Size= 0.7, Text= "Hong Kong" },
new ChartData { X= 79.2, Y= 3.9, Size= 0.162, Text= "Netherland" },
new ChartData { X= 72.5, Y= 4.5, Size= 0.7, Text= "Jordan" },
new ChartData { X= 81, Y= 3.5, Size= 0.21, Text= "Australia" },
new ChartData { X= 66.8, Y= 3.9, Size= 0.028, Text= "Mongolia" },
new ChartData { X= 78.4, Y= 2.9, Size= 0.231, Text= "Taiwan" }
};
}

```



Refer to our [Blazor Charts](#) feature tour page for its groundbreaking feature representations and also explore our [Blazor Chart Example](#) to know various chart types and how to represent time-dependent data, showing trends at equal intervals.

See Also

- [Data Label](#)
- [Tooltip](#)

Polar in Blazor Charts Component

Polar

[Polar Chart](#) series visualizes data in terms of values and angles. It provides options for visual comparison between several quantitative or qualitative aspects of a situation. To render a polar chart, set the series [Type](#) to [Polar](#).

Refer to our [Blazor Polar Chart](#) feature tour page to know about its other groundbreaking feature representations. Explore our [Blazor Polar Chart Example](#) to know how to render and configure the Polar type chart.

Draw Types

To change the series plotting type to [Line](#), [Column](#), [Area](#), [RangeColumn](#), [Spline](#), [Scatter](#), [StackingArea](#) and [StackingColumn](#) use the Polar's [DrawType](#) property. [DrawType](#) is set to [Line](#) by default.

Line

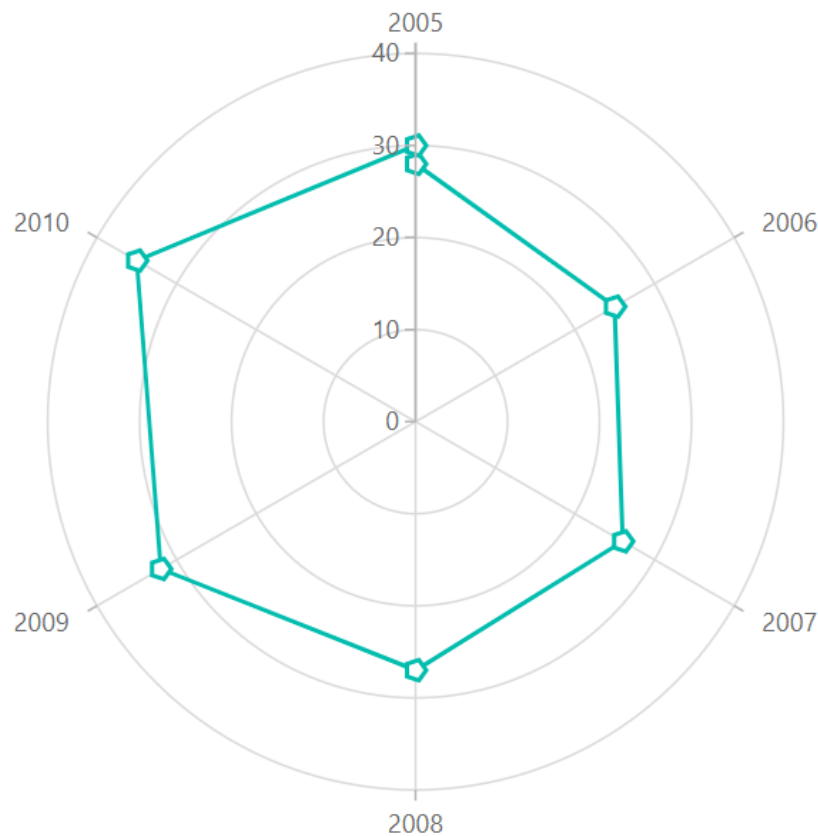
To render a [line](#) series in [Polar Chart](#), specify the [DrawType](#) property to [Line](#). [IsClosed](#) property specifies whether to join start and end point of a line series used in [Polar Chart](#) to form a closed path. The default value of [IsClosed](#) property is **true**.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart>
  <ChartSeriesCollection>
    <ChartSeries DataSource="@SalesReports" XName="X" Width="2" YName="Y"
      Type="ChartSeriesType.Polar" DrawType="ChartDrawType.Line">
      <ChartMarker Height="10" Width="10" Visible="true"
        Shape="ChartShape.Pentagon"></ChartMarker>
    </ChartSeries>
  </ChartSeriesCollection>
</SfChart>

@code{
public class ChartData
{
    public double X { get; set; }
    public double Y { get; set; }
}

public List<ChartData> SalesReports = new List<ChartData>
{
    new ChartData{ X= 2005, Y= 28 },
    new ChartData{ X= 2006, Y= 25 },
    new ChartData{ X= 2007, Y= 26 },
    new ChartData{ X= 2008, Y= 27 },
    new ChartData{ X= 2009, Y= 32 },
    new ChartData{ X= 2010, Y= 35 },
    new ChartData{ X= 2011, Y= 30 }
};
}
```



Spline

To render a [spline](#) series in [Polar Chart](#), specify the [DrawType](#) property to [Spline](#).

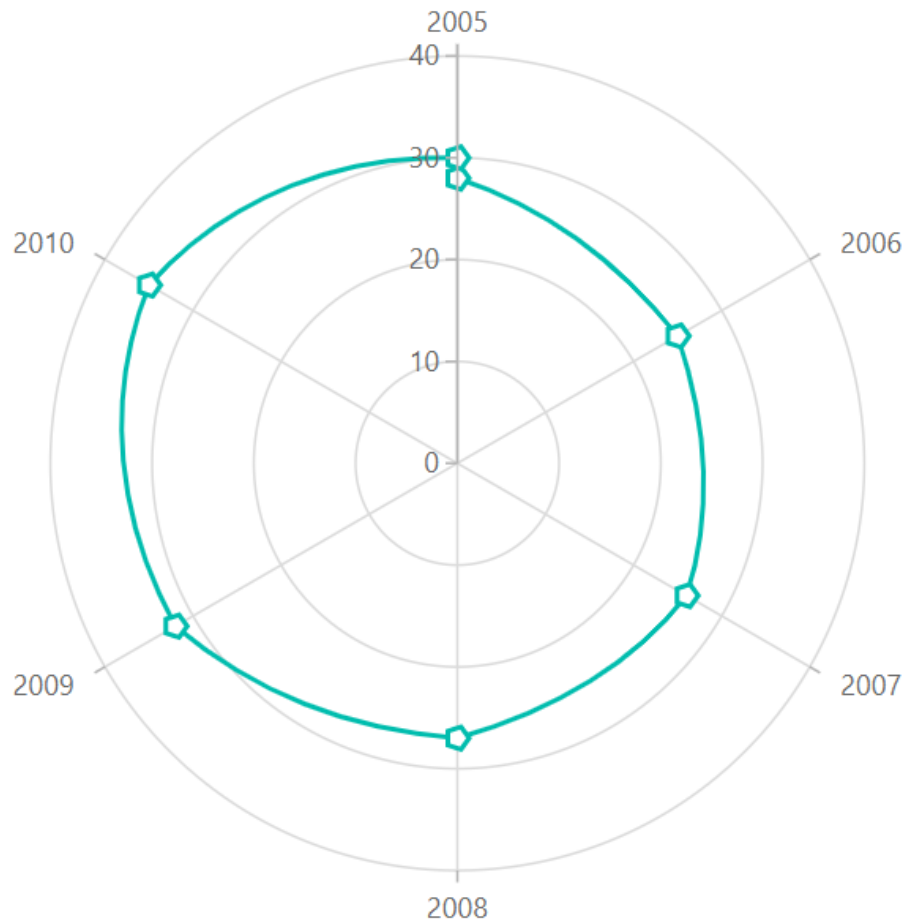
ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart>
<ChartSeriesCollection>
<ChartSeries DataSource="@SalesReports" XName="X" Width="2" Opacity="1"
YName="Y"
Type="ChartSeriesType.Polar" DrawType="ChartDrawType.Spline">
<ChartMarker Height="10" Width="10" Visible="true"
Shape="ChartShape.Pentagon"></ChartMarker>
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code{
public class ChartData
{
public double X { get; set; }
public double Y { get; set; }
}
public List<ChartData> SalesReports = new List<ChartData>
{
new ChartData{ X= 2005, Y= 28 },
new ChartData{ X= 2006, Y= 25 },
new ChartData{ X= 2007, Y= 26 },
```

```

new ChartData{ X= 2008, Y= 27 },
new ChartData{ X= 2009, Y= 32 },
new ChartData{ X= 2010, Y= 35 },
new ChartData{ X= 2011, Y= 30 }
};
}

```



Area

To render a [area](#) series in [Polar Chart](#), specify the [DrawType](#) property to [Area](#).

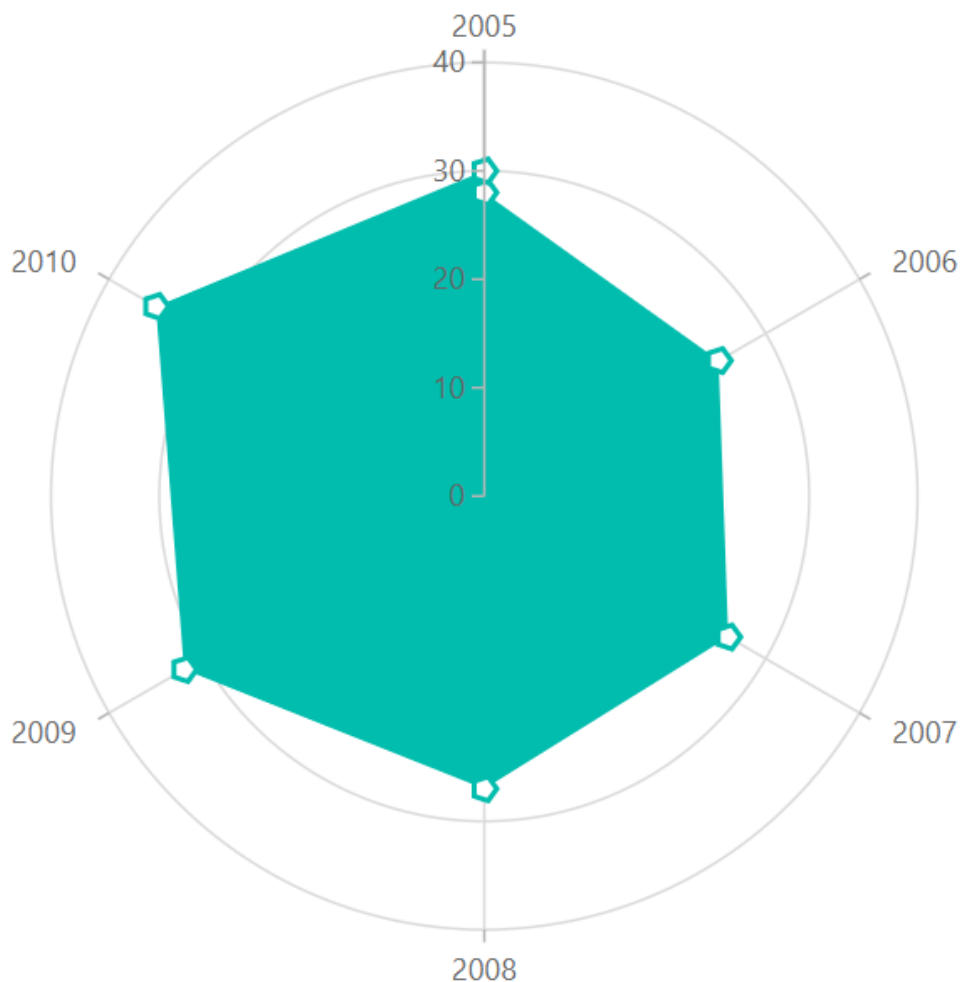
ASPX-CS

```

@using Syncfusion.Blazor.Charts
<SfChart>
<ChartSeriesCollection>
<ChartSeries DataSource="@SalesReports" XName="X" Width="2" Opacity="1"
YName="Y"
Type="ChartSeriesType.Polar" DrawType="ChartDrawType.Area">
<ChartMarker Height="10" Width="10" Visible="true"
Shape="ChartShape.Pentagon"></ChartMarker>
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code{

```

```
public class ChartData
{
    public double X { get; set; }
    public double Y { get; set; }
}
public List<ChartData> SalesReports = new List<ChartData>
{
    new ChartData{ X= 2005, Y= 28 },
    new ChartData{ X= 2006, Y= 25 },
    new ChartData{ X= 2007, Y= 26 },
    new ChartData{ X= 2008, Y= 27 },
    new ChartData{ X= 2009, Y= 32 },
    new ChartData{ X= 2010, Y= 35 },
    new ChartData{ X= 2011, Y= 30 }
};
}
```



Stacked Area

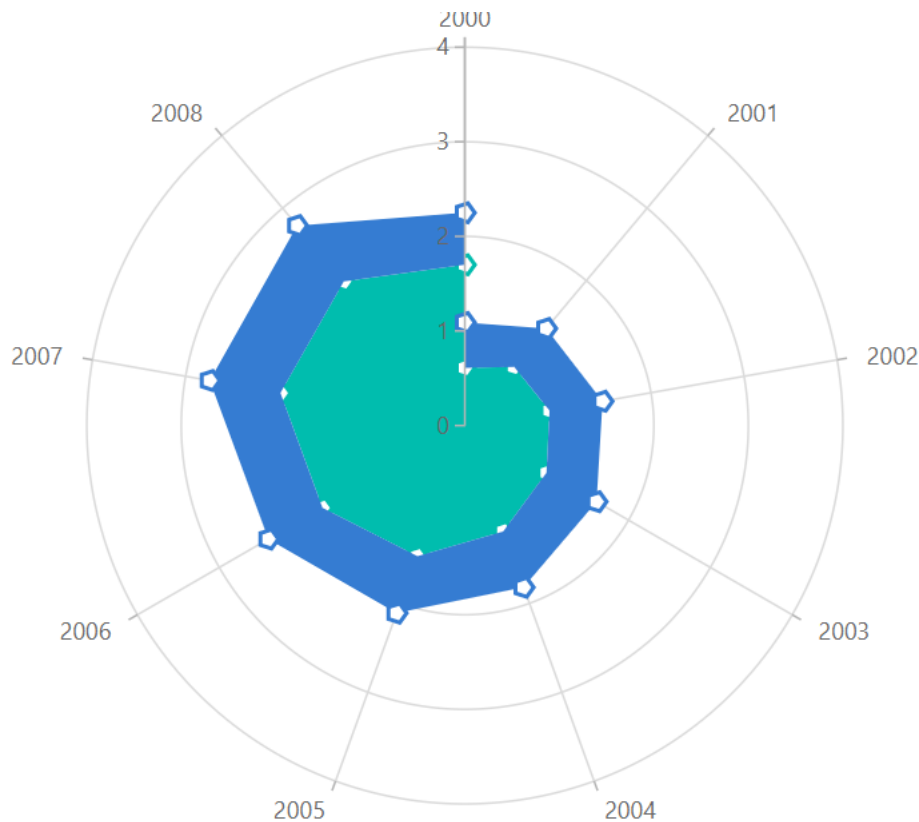
To render a [stacking area](#) series in [Polar Chart](#), specify the [DrawType](#) property to [StackingArea](#).

ASPX-CS

```

@using Syncfusion.Blazor.Charts
<SfChart>
<ChartSeriesCollection>
<ChartSeries DataSource="@SalesReports" XName="X" YName="Y"
Type="ChartSeriesType.Polar" DrawType="ChartDrawType.StackingArea">
<ChartMarker Height="10" Width="10" Visible="true"
Shape="ChartShape.Pentagon"></ChartMarker>
</ChartSeries>
<ChartSeries DataSource="@SalesReports" XName="X" YName="Y1"
Type="ChartSeriesType.Polar" DrawType="ChartDrawType.StackingArea">
<ChartMarker Height="10" Width="10" Visible="true"
Shape="ChartShape.Pentagon"></ChartMarker>
</ChartSeries>
<ChartSeries DataSource="@SalesReports" XName="X" YName="Y2"
Type="ChartSeriesType.Polar" DrawType="ChartDrawType.StackingArea">
<ChartMarker Height="10" Width="10" Visible="true"
Shape="ChartShape.Pentagon"></ChartMarker>
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code{
public class ChartData
{
public double X { get; set; }
public double Y { get; set; }
public double Y1 { get; set; }
public double Y2 { get; set; }
}
public List<ChartData> SalesReports = new List<ChartData>
{
new ChartData{ X=2000, Y= 0.61, Y1= 0.03, Y2= 0.48},
new ChartData{ X=2001, Y= 0.81, Y1= 0.05, Y2= 0.53 },
new ChartData{ X=2002, Y= 0.91, Y1= 0.06, Y2= 0.57 },
new ChartData{ X=2003, Y= 1, Y1= 0.09, Y2= 0.61 },
new ChartData{ X=2004, Y= 1.19, Y1= 0.14, Y2= 0.63 },
new ChartData{ X=2005, Y= 1.47, Y1= 0.20, Y2= 0.64 },
new ChartData{ X=2006, Y= 1.74, Y1= 0.29, Y2= 0.66 },
new ChartData{ X=2007, Y= 1.98, Y1= 0.46, Y2= 0.76 },
new ChartData{ X=2008, Y= 1.99, Y1= 0.64, Y2= 0.77 },
new ChartData{ X=2009, Y= 1.70, Y1= 0.75, Y2= 0.55 }
};
}

```



Column

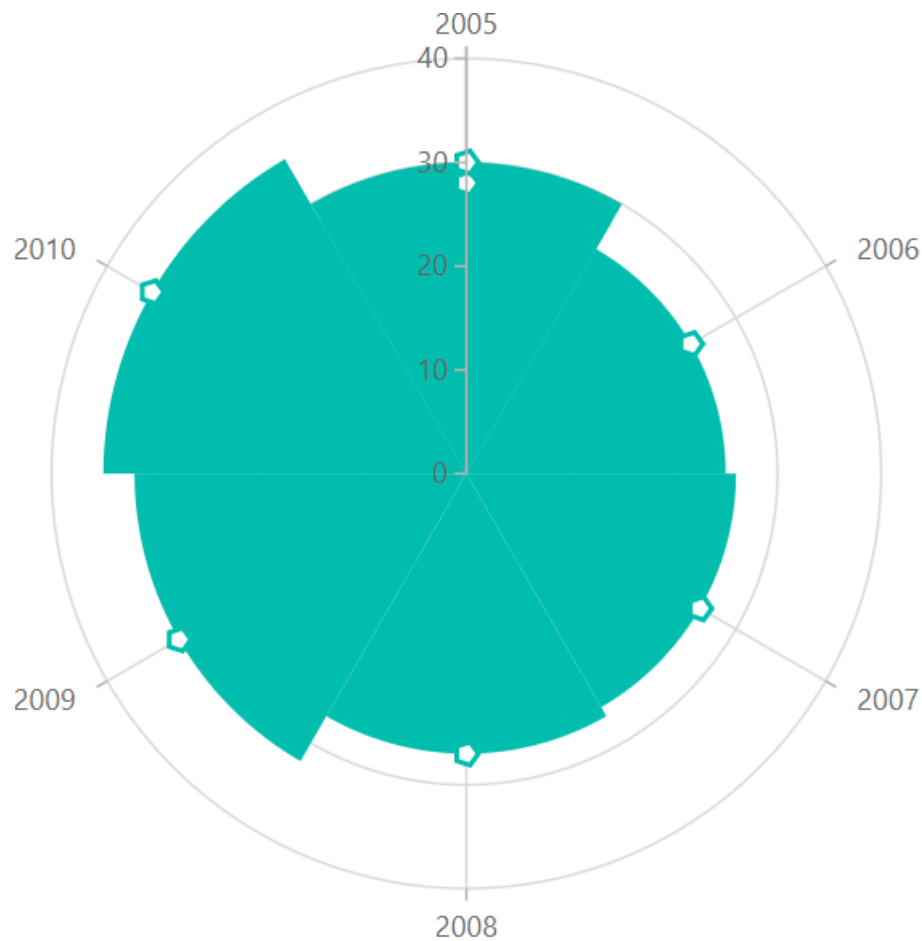
To render a [column](#) series in [Polar Chart](#), specify the [DrawType](#) property to [Column](#).

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart >
<ChartSeriesCollection>
<ChartSeries DataSource="@SalesReports" XName="X" YName="Y"
Type="ChartSeriesType.Polar" DrawType="ChartDrawType.Column">
<ChartMarker Height="10" Width="10" Visible="true"
Shape="ChartShape.Pentagon"></ChartMarker>
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code{
public class ChartData
{
public double X { get; set; }
public double Y { get; set; }
}
public List<ChartData> SalesReports = new List<ChartData>
{
new ChartData{ X= 2005, Y= 28 },
new ChartData{ X= 2006, Y= 25 },
new ChartData{ X= 2007, Y= 26 },
new ChartData{ X= 2008, Y= 27 },
new ChartData{ X= 2009, Y= 32 },
new ChartData{ X= 2010, Y= 35 },
}
```



```
new ChartData{ X= 2011, Y= 30 }
};
}
```



Stacked Column

To render a [stacking column](#) series in [Polar Chart](#), specify the [DrawType](#) property to [StackingColumn](#).

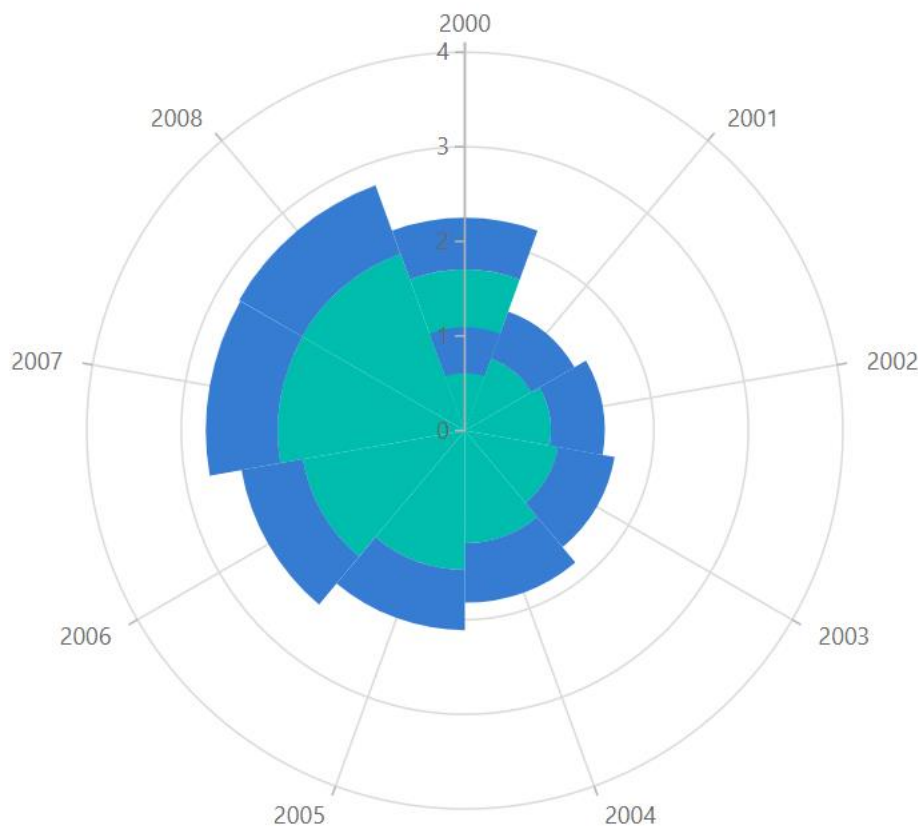
ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart>
<ChartSeriesCollection>
<ChartSeries DataSource="@SalesReports" XName="X" YName="Y"
Type="ChartSeriesType.Polar" DrawType="ChartDrawType.StackingColumn">
</ChartSeries>
<ChartSeries DataSource="@SalesReports" XName="X" YName="Y1"
Type="ChartSeriesType.Polar" DrawType="ChartDrawType.StackingColumn">
</ChartSeries>
<ChartSeries DataSource="@SalesReports" XName="X" YName="Y2"
Type="ChartSeriesType.Polar" DrawType="ChartDrawType.StackingColumn">
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code{
```

```

public class ChartData
{
    public double X { get; set; }
    public double Y { get; set; }
    public double Y1 { get; set; }
    public double Y2 { get; set; }
}
public List<ChartData> SalesReports = new List<ChartData>
{
    new ChartData{ X=2000, Y= 0.61, Y1= 0.03, Y2= 0.48},
    new ChartData{ X=2001, Y= 0.81, Y1= 0.05, Y2= 0.53 },
    new ChartData{ X=2002, Y= 0.91, Y1= 0.06, Y2= 0.57 },
    new ChartData{ X=2003, Y= 1, Y1= 0.09, Y2= 0.61 },
    new ChartData{ X=2004, Y= 1.19, Y1= 0.14, Y2= 0.63 },
    new ChartData{ X=2005, Y= 1.47, Y1= 0.20, Y2= 0.64 },
    new ChartData{ X=2006, Y= 1.74, Y1= 0.29, Y2= 0.66 },
    new ChartData{ X=2007, Y= 1.98, Y1= 0.46, Y2= 0.76 },
    new ChartData{ X=2008, Y= 1.99, Y1= 0.64, Y2= 0.77 },
    new ChartData{ X=2009, Y= 1.70, Y1= 0.75, Y2= 0.55 }
};

```

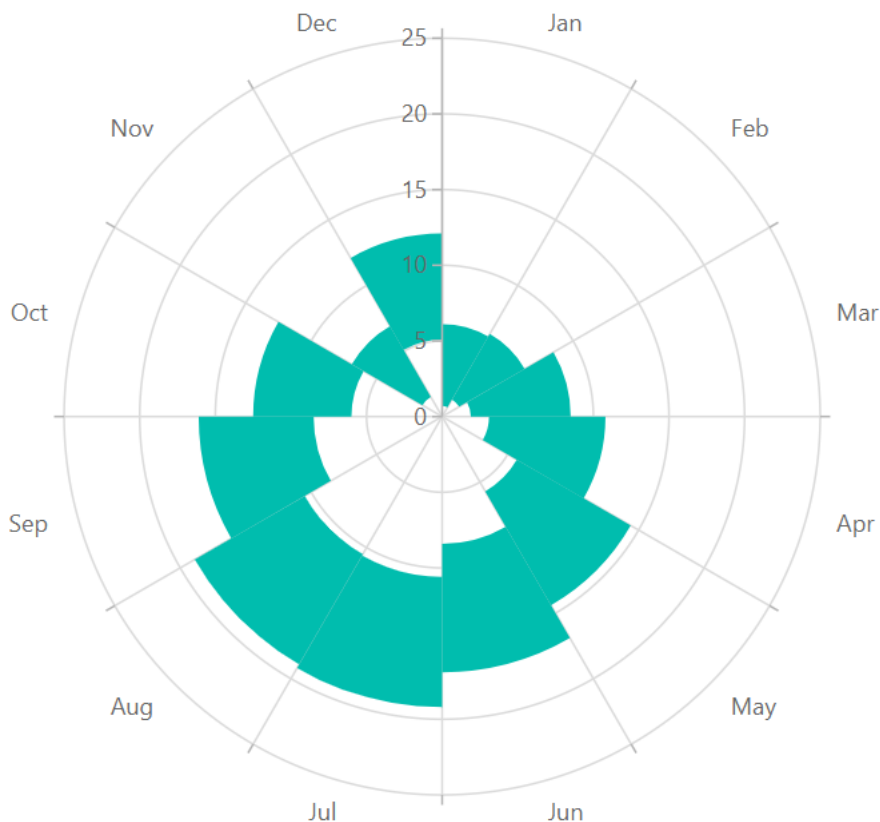


Range Column

To render a [range column](#) series in [Polar Chart](#), specify the [DrawType](#) property to [RangeColumn](#).

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart>
<ChartPrimaryXAxis ValueType="Syncfusion.Blazor.Charts.ValueType.Category">
</ChartPrimaryXAxis>
<ChartSeriesCollection>
<ChartSeries DataSource="@WeatherReports" XName="X" High="High" Low="Low"
Type="ChartSeriesType.Polar" DrawType="ChartDrawType.RangeColumn">
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code{
public class ChartData
{
public string X { get; set; }
public double Low { get; set; }
public double High { get; set; }
}
public List<ChartData> WeatherReports = new List<ChartData>
{
new ChartData{ X="Jan", Low= 0.7, High= 6.1 },
new ChartData{ X="Feb", Low=1.3, High=6.3 },
new ChartData{ X="Mar", Low= 1.9, High= 8.5 },
new ChartData{ X= "Apr", Low= 3.1, High= 10.8 },
new ChartData{ X="May", Low= 5.7, High= 14.40 },
new ChartData { X= "Jun", Low= 8.4, High= 16.90 },
new ChartData { X= "Jul", Low= 10.6,High= 19.20 },
new ChartData { X= "Aug", Low= 10.5,High= 18.9 },
new ChartData { X= "Sep", Low= 8.5, High= 16.1 },
new ChartData { X= "Oct", Low= 6.0, High= 12.5 },
new ChartData { X= "Nov", Low= 1.5, High= 6.9 },
new ChartData { X= "Dec", Low= 5.1, High= 12.1 }
};
}
```



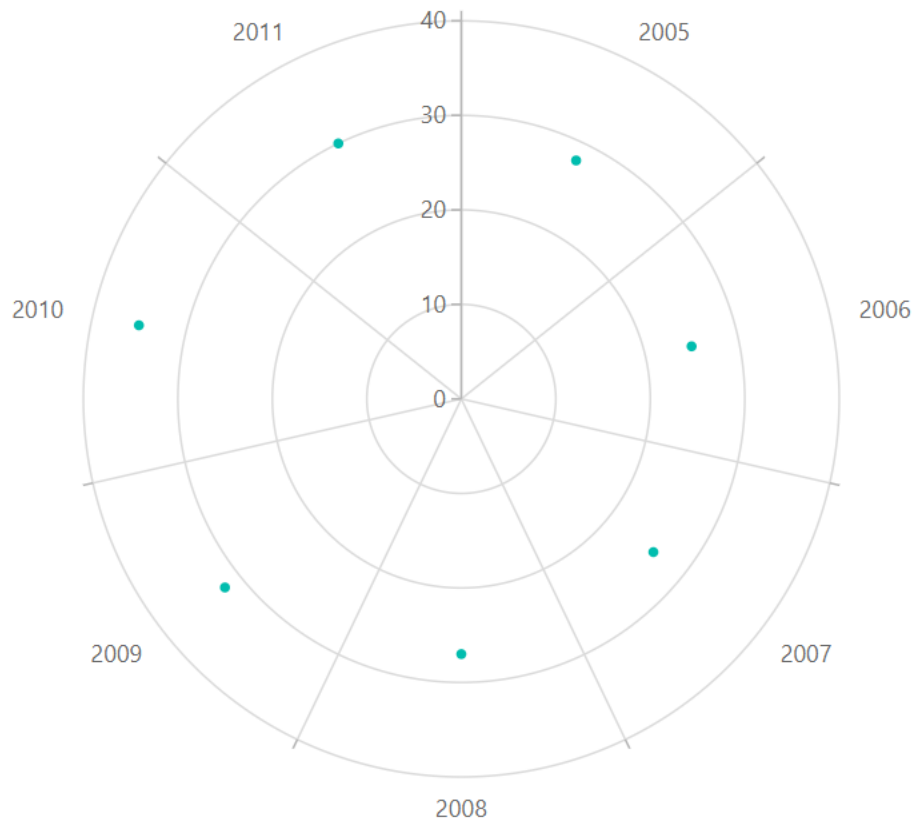
Scatter

To render a [scatter](#) series in [Polar Chart](#), specify the [DrawType](#) property to [Scatter](#).

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart>
<ChartPrimaryXAxis ValueType="Syncfusion.Blazor.Charts.ValueType.Category">
</ChartPrimaryXAxis>
<ChartSeriesCollection>
<ChartSeries DataSource="@SalesReports" XName="X" YName="Y"
Type="ChartSeriesType.Polar" DrawType="ChartDrawType.Scatter">
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code{
public class ChartData
{
public double X { get; set; }
public double Y { get; set; }
}
public List<ChartData> SalesReports = new List<ChartData>
{
new ChartData{ X= 2005, Y= 28 },
new ChartData{ X= 2006, Y= 25 },
new ChartData{ X= 2007, Y= 26 },
new ChartData{ X= 2008, Y= 27 },
}
```

```
new ChartData{ X= 2009, Y= 32 },
new ChartData{ X= 2010, Y= 35 },
new ChartData{ X= 2011, Y= 30 }
};
}
```



Series Customization

Start Angle

To customize the start angle of the [Polar Chart](#) use [StartAngle](#) property. By default, [StartAngle](#) value is 0.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart>
<ChartPrimaryXAxis StartAngle="270"
ValueType="Syncfusion.Blazor.Charts.ValueType.Category">
</ChartPrimaryXAxis>
<ChartSeriesCollection>
<ChartSeries DataSource="@SalesReports" XName="X" YName="Y"
Type="ChartSeriesType.Polar" DrawType="ChartDrawType.Line">
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code{
public class ChartData
{
public double X { get; set; }
}
```

```

public double Y { get; set; }
}
public List<ChartData> SalesReports = new List<ChartData>
{
    new ChartData{ X= 2005, Y= 28 },
    new ChartData{ X= 2006, Y= 25 },
    new ChartData{ X= 2007, Y= 26 },
    new ChartData{ X= 2008, Y= 27 },
    new ChartData{ X= 2009, Y= 32 },
    new ChartData{ X= 2010, Y= 35 },
    new ChartData{ X= 2011, Y= 30 }
};
}

```



Radius

To customize the radius of the [Polar Chart](#) use [Coefficient](#) property. By default, [Coefficient](#) value is **100**.

ASPX-CS

```

@using Syncfusion.Blazor.Charts
<SfChart>
<ChartPrimaryXAxis Coefficient="40"
    ValueType="Syncfusion.Blazor.Charts.ValueType.Category">
</ChartPrimaryXAxis>
<ChartSeriesCollection>

```

```

<ChartSeries DataSource="@SalesReports" XName="X" YName="Y"
Type="ChartSeriesType.Polar" DrawType="ChartDrawType.Line">
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code{
public class ChartData
{
public double X { get; set; }
public double Y { get; set; }
}
public List<ChartData> SalesReports = new List<ChartData>
{
new ChartData{ X= 2005, Y= 28 },
new ChartData{ X= 2006, Y= 25 },
new ChartData{ X= 2007, Y= 26 },
new ChartData{ X= 2008, Y= 27 },
new ChartData{ X= 2009, Y= 32 },
new ChartData{ X= 2010, Y= 35 },
new ChartData{ X= 2011, Y= 30 }
};
}

```



Refer to our [Blazor Charts](#) feature tour page for its groundbreaking feature representations and also explore our [Blazor Chart Example](#) to know various chart types and how to represent time-dependent data, showing trends at equal intervals.

See Also

- [Data Label](#)
- [Tooltip](#)

Radar in Blazor Charts Component

Radar

[Radar](#) series visualizes data in terms of values and angles. It provides option for visual comparison between several quantitative or qualitative aspects of a situation. To render a radar chart, set the series

[Type](#) to [Radar](#). To render a [Line](#) series in [Radar Chart](#), specify the [DrawType](#) property to [Line](#). [IsClosed](#) property specifies whether to join start and end point of a line series used in [Radar Chart](#) to form a closed path. The default value of [IsClosed](#) property is **true**.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart>
<ChartPrimaryXAxis ValueType="Syncfusion.Blazor.Charts.ValueType.Category">
</ChartPrimaryXAxis>
<ChartSeriesCollection>
<ChartSeries DataSource="@SalesReports" XName="X" YName="Y"
Type="ChartSeriesType.Radar" DrawType="ChartDrawType.Line">
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code{
public class ChartData
{
public double X { get; set; }
public double Y { get; set; }
}
public List<ChartData> SalesReports = new List<ChartData>
{
new ChartData{ X= 2005, Y= 28 },
new ChartData{ X= 2006, Y= 25 },
new ChartData{ X= 2007, Y= 26 },
new ChartData{ X= 2008, Y= 27 },
new ChartData{ X= 2009, Y= 32 },
new ChartData{ X= 2010, Y= 35 },
new ChartData{ X= 2011, Y= 30 }
};
}
```




Refer to our [Blazor Radar Chart](#) feature tour page to know about its other groundbreaking feature representations. Explore our [Blazor Radar Chart Example](#) to know how to render and configure the Radar type chart.

Series Customization

Start Angle

To customize the start angle of the [Radar Chart](#) use [StartAngle](#) property. By default, [StartAngle](#) value is 0.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart>
  <ChartPrimaryXAxis StartAngle="270"
  ValueType="Syncfusion.Blazor.Charts.ValueType.Category">
  </ChartPrimaryXAxis>
  <ChartSeriesCollection>
    <ChartSeries DataSource="@SalesReports" XName="X" YName="Y"
    Type="ChartSeriesType.Radar" DrawType="ChartDrawType.Line">
    </ChartSeries>
  </ChartSeriesCollection>
</SfChart>
@code{
public class ChartData
{
  public double X { get; set; }
  public double Y { get; set; }
}
```

```

}
public List<ChartData> SalesReports = new List<ChartData>
{
    new ChartData{ X= 2005, Y= 28 },
    new ChartData{ X= 2006, Y= 25 },
    new ChartData{ X= 2007, Y= 26 },
    new ChartData{ X= 2008, Y= 27 },
    new ChartData{ X= 2009, Y= 32 },
    new ChartData{ X= 2010, Y= 35 },
    new ChartData{ X= 2011, Y= 30 }
};
}

```



Coefficient in axis

To customize the radius of the [Radar Chart](#), use [Coefficient](#) property. By default, [Coefficient](#) value is **100**.

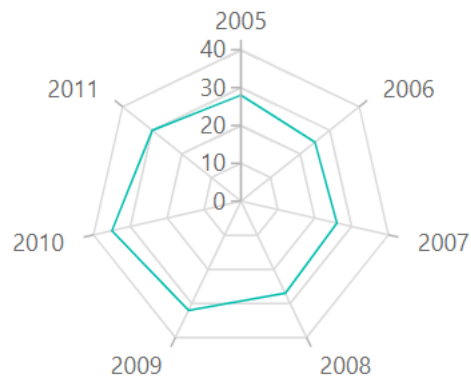
ASPX-CS

```

@using Syncfusion.Blazor.Charts
<SfChart>
<ChartPrimaryXAxis Coefficient="40"
ValueType="Syncfusion.Blazor.Charts.ValueType.Category">
</ChartPrimaryXAxis>
<ChartSeriesCollection>
<ChartSeries DataSource="@SalesReports" XName="X" YName="Y"
Type="ChartSeriesType.Radar" DrawType="ChartDrawType.Line">
</ChartSeries>

```

```
</ChartSeriesCollection>
</SfChart>
@code{
public class ChartData
{
public double X { get; set; }
public double Y { get; set; }
}
public List<ChartData> SalesReports = new List<ChartData>
{
new ChartData{ X= 2005, Y= 28 },
new ChartData{ X= 2006, Y= 25 },
new ChartData{ X= 2007, Y= 26 },
new ChartData{ X= 2008, Y= 27 },
new ChartData{ X= 2009, Y= 32 },
new ChartData{ X= 2010, Y= 35 },
new ChartData{ X= 2011, Y= 30 }
};
}
```



Refer to our [Blazor Charts](#) feature tour page for its groundbreaking feature representations and also explore our [Blazor Chart Example](#) to know various chart types and how to represent time-dependent data, showing trends at equal intervals.

See Also

- [Data Label](#)
- [Tooltip](#)

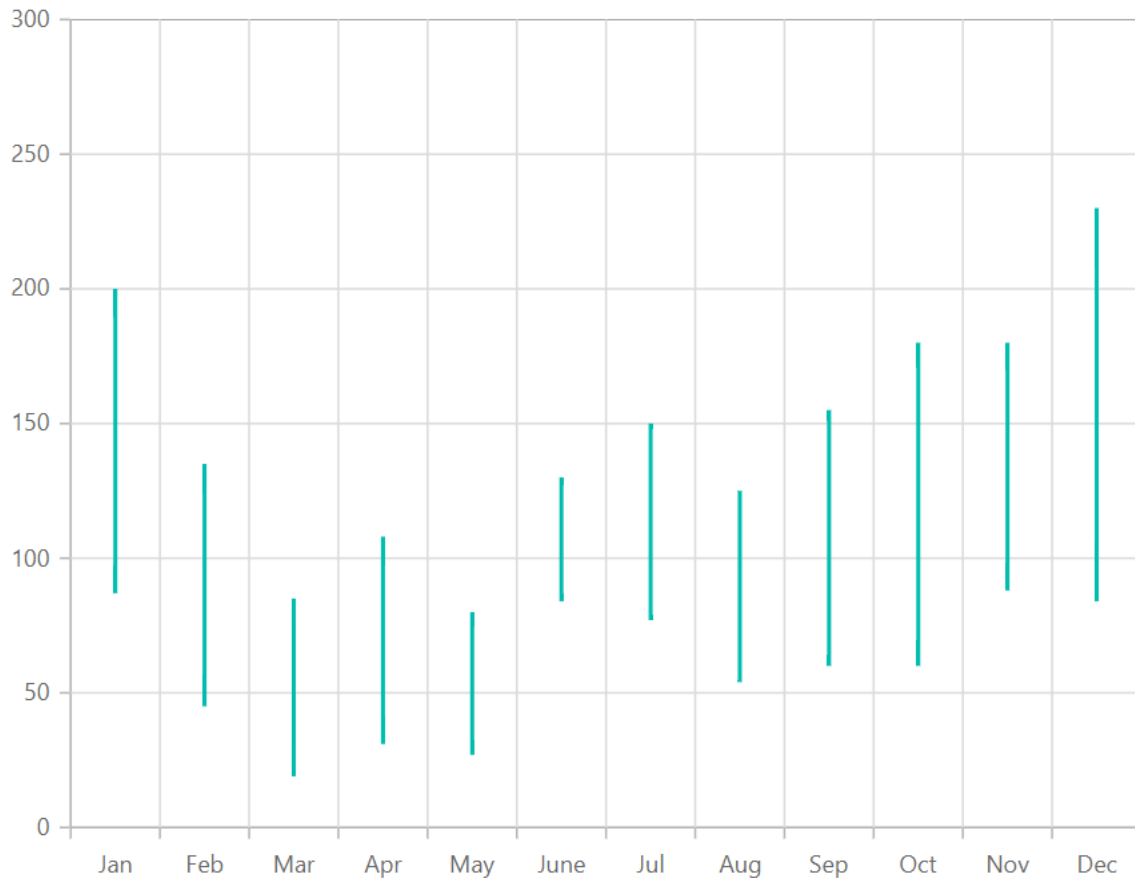
Hilo in Blazor Charts Component

Hilo

[Hilo](#) series illustrates the price movements in stock using the higher and lower values and it can be rendered by specifying the series [Type](#) as [Hilo](#). Hilo series requires three fields (XName, High and Low) to show the higher and lower price in the stock.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart>
<ChartPrimaryXAxis ValueType="Syncfusion.Blazor.Charts.ValueType.Category"/>
<ChartSeriesCollection>
<ChartSeries DataSource="@StockDetails" XName="X" High="High" Low="Low"
Type="ChartSeriesType.Hilo">
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code{
public class Data
{
public string X { get; set; }
public double Y { get; set; }
public double High { get; set; }
public double Low { get; set; }
}
public List<Data> StockDetails = new List<Data>
{
new Data{ X= "Jan", Low= 87, High= 200 },
new Data{ X= "Feb", Low= 45, High= 135 },
new Data{ X= "Mar", Low= 19, High= 85 },
new Data{ X= "Apr", Low= 31, High= 108 },
new Data{ X= "May", Low= 27, High= 80 },
new Data{ X= "June",Low= 84, High= 130 },
new Data{ X= "Jul", Low= 77, High=150 },
new Data{ X= "Aug", Low= 54, High= 125 },
new Data{ X= "Sep", Low= 60, High= 155 },
new Data{ X= "Oct", Low= 60, High= 180 },
new Data{ X= "Nov", Low= 88, High= 180 },
new Data{ X= "Dec", Low= 84, High= 230 }
};
}
```



Refer to our [Blazor Hilo Chart](#) feature tour page to know about its other groundbreaking feature representations and also explore our [Blazor Hilo Chart Example](#) to know how to render and configure the Hilo type series.

Series Customization

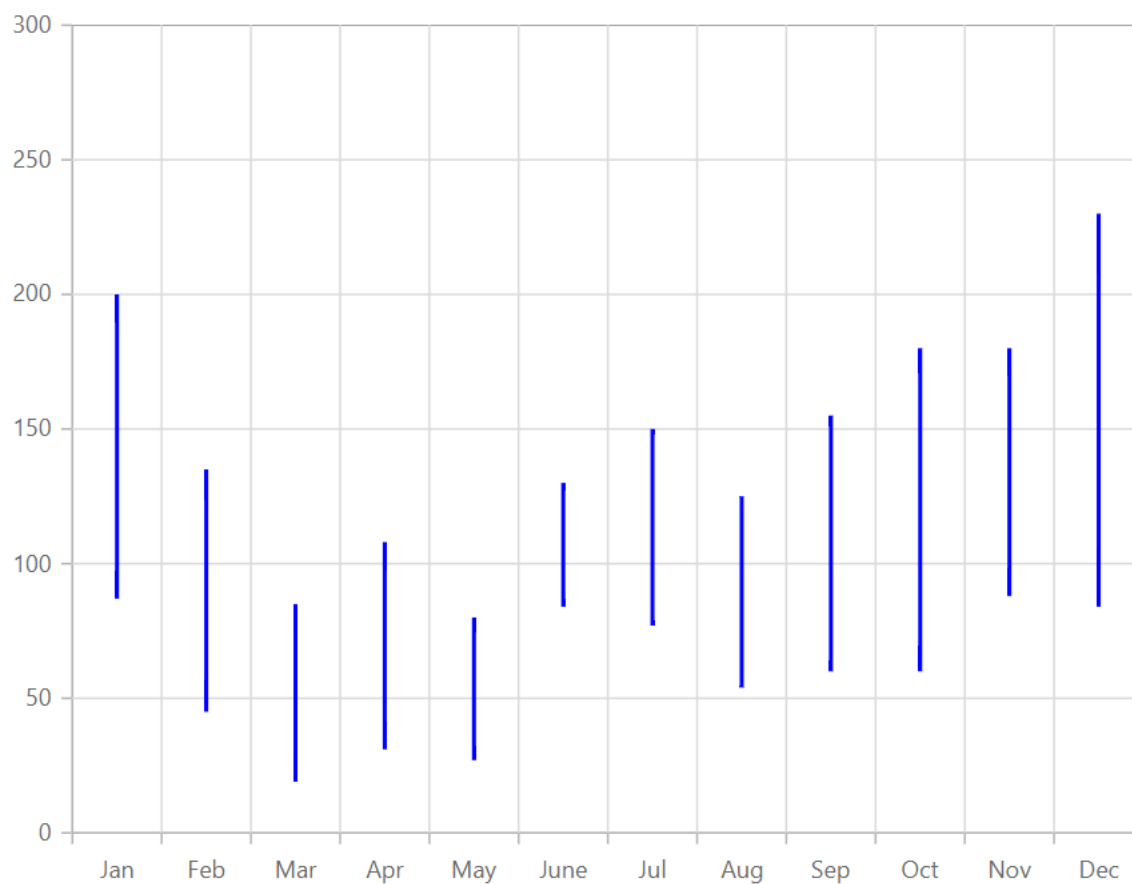
The following properties can be used to customize the [Hilo](#) series.

- [Fill](#) – Specifies the color of the series.
- [Opacity](#) – Specifies the opacity of [Fill](#).

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart>
<ChartPrimaryXAxis ValueType="Syncfusion.Blazor.Charts.ValueType.Category"
/>
<ChartSeriesCollection>
<ChartSeries DataSource="@StockDetails" XName="X" High="High" Low="Low"
Fill="blue" Type="ChartSeriesType.Hilo">
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code{
public class Data
{
```

```
public string X { get; set; }
public double Y { get; set; }
public double High { get; set; }
public double Low { get; set; }
}
public List<Data> StockDetails = new List<Data>
{
    new Data{ X= "Jan", Low= 87, High= 200 },
    new Data{ X= "Feb", Low= 45, High= 135 },
    new Data{ X= "Mar", Low= 19, High= 85 },
    new Data{ X= "Apr", Low= 31, High= 108 },
    new Data{ X= "May", Low= 27, High= 80 },
    new Data{ X= "June", Low= 84, High= 130 },
    new Data{ X= "Jul", Low= 77, High=150 },
    new Data{ X= "Aug", Low= 54, High= 125 },
    new Data{ X= "Sep", Low= 60, High= 155 },
    new Data{ X= "Oct", Low= 60, High= 180 },
    new Data{ X= "Nov", Low= 88, High= 180 },
    new Data{ X= "Dec", Low= 84, High= 230 }
};
}
```



Refer to our [Blazor Charts](#) feature tour page for its groundbreaking feature representations and also explore our [Blazor Chart Example](#) to know various chart types and how to represent time-dependent data, showing trends at equal intervals.

See Also

- [Data Label](#)
- [Tooltip](#)

High Low Open Close in Blazor Charts Component

High Low Open Close

[HiloOpenClose](#) series is used to represent the **Low, High, Open** and **Closing** values over time and it can be rendered by specifying the series [Type](#) as [HiloOpenClose](#).

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart>
<ChartPrimaryXAxis ValueType="Syncfusion.Blazor.Charts.ValueType.Category"
/>
<ChartSeriesCollection>
<ChartSeries DataSource="@StockDetails" XName="X" High="High" Low="Low"
Open="Open" Close="Close" Type="ChartSeriesType.HiloOpenClose">
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code{
public class Data
{
public string X { get; set; }
public double Y { get; set; }
public double High { get; set; }
public double Low { get; set; }
public double Open { get; set; }
public double Close { get; set; }
}
public List<Data> StockDetails = new List<Data>
{
new Data{ X= "Jan", Open= 120, High= 160, Low= 100, Close= 140 },
new Data{ X= "Feb", Open= 150, High= 190, Low= 130, Close= 170 },
new Data{ X= "Mar", Open= 130, High= 170, Low= 110, Close= 150 },
new Data{ X= "Apr", Open= 160, High= 180, Low= 120, Close= 140 },
new Data{ X= "May", Open= 150, High= 170, Low= 110, Close= 130 }
};
}
```

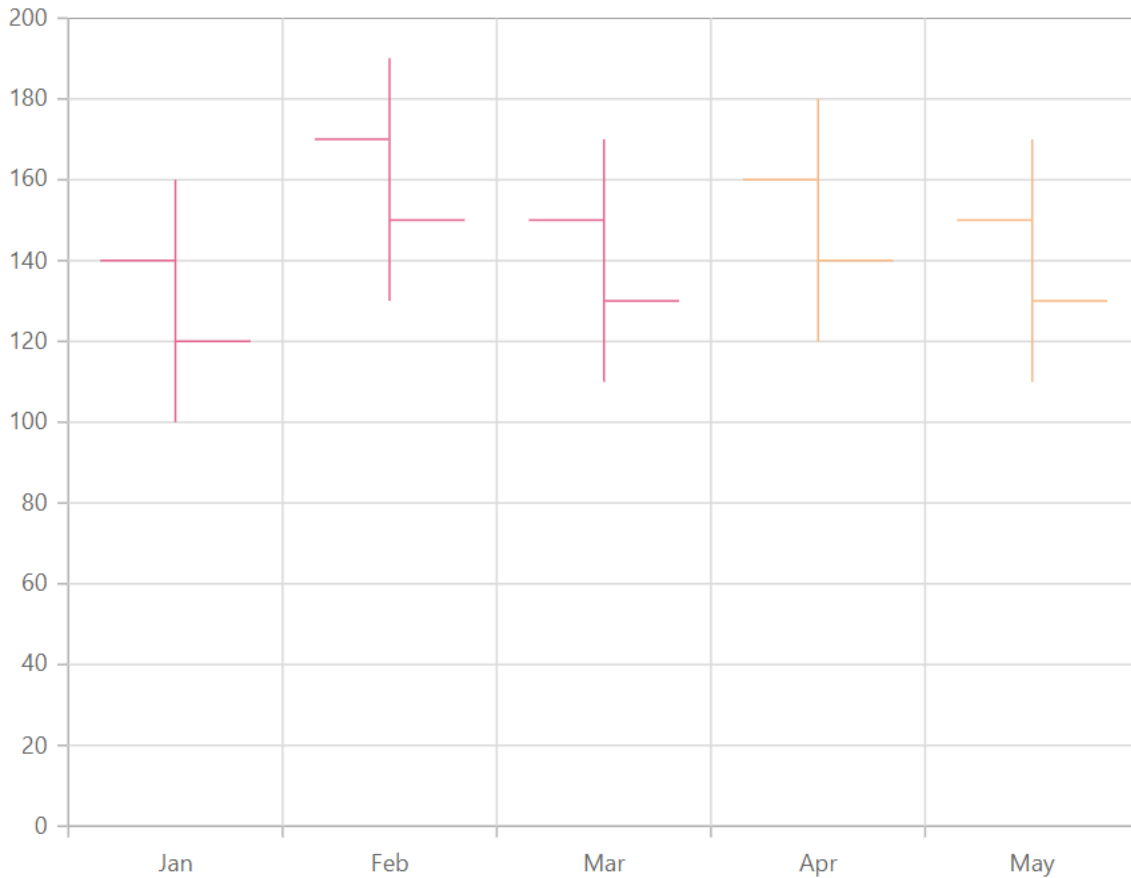
Series Customization

In [HiloOpenClose](#) series, [BullFillColor](#) property is used to fill the segment when the open value is greater than the close value and [BearFillColor](#) property is used to fill the segment when the open value is less than the close value. By default, [BullFillColor](#) is set as **green** and [BearFillColor](#) is set as **red**.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
```

```
<SfChart>
<ChartPrimaryXAxis ValueType="Syncfusion.Blazor.Charts.ValueType.Category"/>
<ChartSeriesCollection>
<ChartSeries DataSource="@StockDetails" XName="X" High="High" Width="5"
BearFillColor="darkred" BullFillColor="darkgreen" Low="Low" Open="Open"
Close="Close" Type="ChartSeriesType.HiloOpenClose">
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code{
public class Data
{
public string X { get; set; }
public double Y { get; set; }
public double High { get; set; }
public double Low { get; set; }
public double Open { get; set; }
public double Close { get; set; }
}
public List<Data> StockDetails = new List<Data>
{
new Data{ X= "Jan", Open= 120, High= 160, Low= 100, Close= 140 },
new Data{ X= "Feb", Open= 150, High= 190, Low= 130, Close= 170 },
new Data{ X= "Mar", Open= 130, High= 170, Low= 110, Close= 150 },
new Data{ X= "Apr", Open= 160, High= 180, Low= 120, Close= 140 },
new Data{ X= "May", Open= 150, High= 170, Low= 110, Close= 130 }
};
}
```

Refer to our [Blazor Charts](#) feature tour page for its groundbreaking feature representations and also explore our [Blazor Chart Example](#) to know various chart types and how to represent time-dependent data, showing trends at equal intervals.

See Also

- [Data Label](#)
- [Tooltip](#)

Candle in Blazor Charts Component

Candle

[Candle](#) series is similar to Hilo Open Close series. It is used to represent the **Low, High, Open and Closing** prices over time. To render a candle series, set series [Type](#) as [Candle](#).

Hollow Candle

[Candle](#) series allows to visually compare the current price with previous price by customizing its appearance. Candles are filled/left as hollow based on the following criteria.

<!-- markdownlint-disable MD033 -->

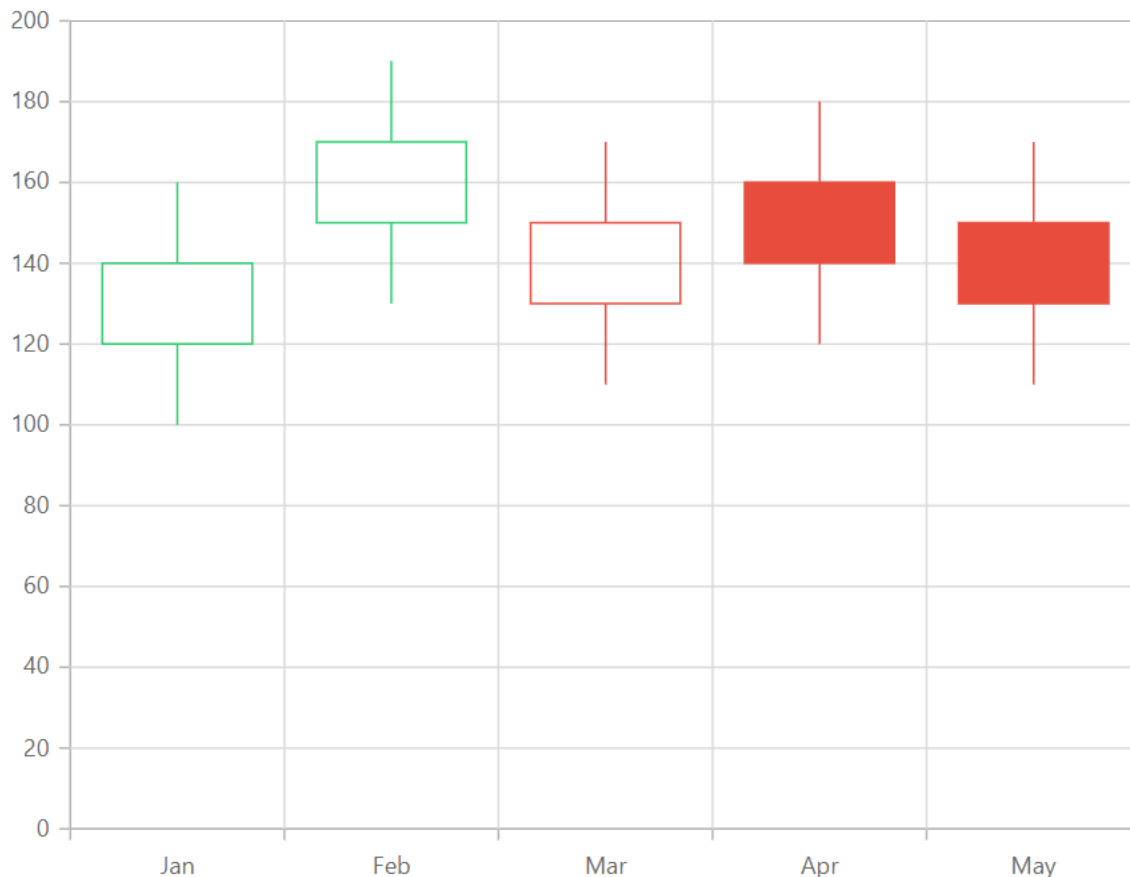
States	Description
Filled	Candle sticks are filled when the close value is lesser than the open value.

Unfilled	Candle sticks are unfilled when the close value is greater than the open value.
----------	---

The color of the candle will be defined by comparing with previous values. [BullFillColor](#) property is used to apply when the current closing value is greater than the previous closing value and [BearFillColor](#) will be applied when the current closing value is less than the previous closing value. By default, [BullFillColor](#) is **green** and [BearFillColor](#) is **red**.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart>
<ChartPrimaryXAxis ValueType="Syncfusion.Blazor.Charts.ValueType.Category">
</ChartPrimaryXAxis>
<ChartSeriesCollection>
<ChartSeries DataSource="@StockDetails" XName="X" High="High" Low="Low"
Type="ChartSeriesType.Candle" Open="Open" Close="Close">
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code{
public class Data
{
public string X { get; set; }
public double Y { get; set; }
public double High { get; set; }
public double Low { get; set; }
public double Open { get; set; }
public double Close { get; set; }
}
public List<Data> StockDetails = new List<Data>
{
new Data{ X= "Jan", Open= 120, High= 160, Low= 100, Close= 140 },
new Data{ X= "Feb", Open= 150, High= 190, Low= 130, Close= 170 },
new Data{ X= "Mar", Open= 130, High= 170, Low= 110, Close= 150 },
new Data{ X= "Apr", Open= 160, High= 180, Low= 120, Close= 140 },
new Data{ X= "May", Open= 150, High= 170, Low= 110, Close= 130 }
};
}
```



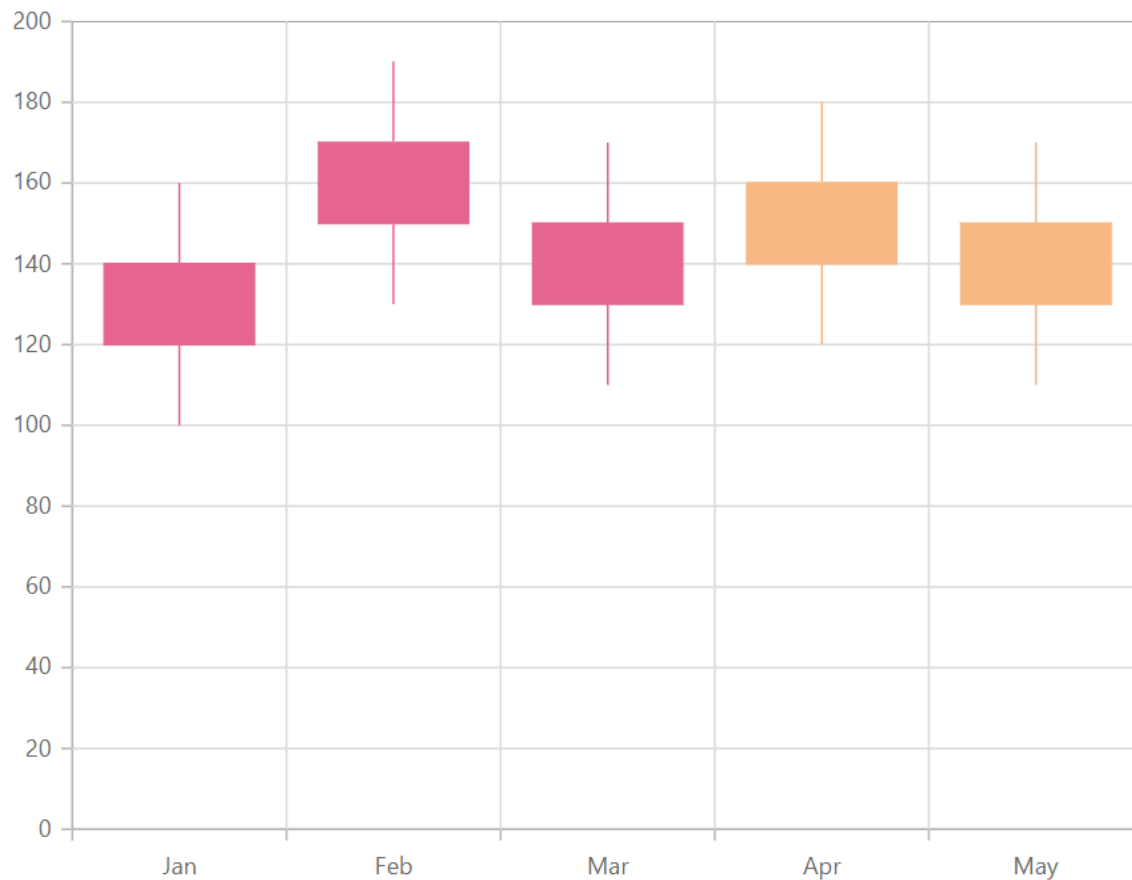
Solid Candles

[EnableSolidCandles](#) property is used to enable/disable the solid candles. By default, it is set as **false**. The fill color of the candle will be defined by its opening and closing values. [BearFillColor](#) will be applied when the opening value is less than the closing value. [BullFillColor](#) will be applied when the opening value is greater than closing value.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart>
<ChartPrimaryXAxis ValueType="Syncfusion.Blazor.Charts.ValueType.Category">
</ChartPrimaryXAxis>
<ChartSeriesCollection>
<ChartSeries DataSource="@StockDetails" XName="X" High="High"
BearFillColor="#e56590" BullFillColor="#f8b883" EnableSolidCandles="true"
Low="Low" Type="ChartSeriesType.Candle" Open="Open" Close="Close">
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code{
public class Data
{
public string X { get; set; }
public double Y { get; set; }
public double High { get; set; }
public double Low { get; set; }
}
```

```
public double Open { get; set; }  
public double Close { get; set; }  
}  
public List<Data> StockDetails = new List<Data>  
{  
    new Data{ X= "Jan", Open= 120, High= 160, Low= 100, Close= 140 },  
    new Data{ X= "Feb", Open= 150, High= 190, Low= 130, Close= 170 },  
    new Data{ X= "Mar", Open= 130, High= 170, Low= 110, Close= 150 },  
    new Data{ X= "Apr", Open= 160, High= 180, Low= 120, Close= 140 },  
    new Data{ X= "May", Open= 150, High= 170, Low= 110, Close= 130 }  
};  
}
```



Refer to our [Blazor Charts](#) feature tour page for its groundbreaking feature representations and also explore our [Blazor Chart Example](#) to know various chart types and how to represent time-dependent data, showing trends at equal intervals.

See Also

- [Data Label](#)
- [Tooltip](#)

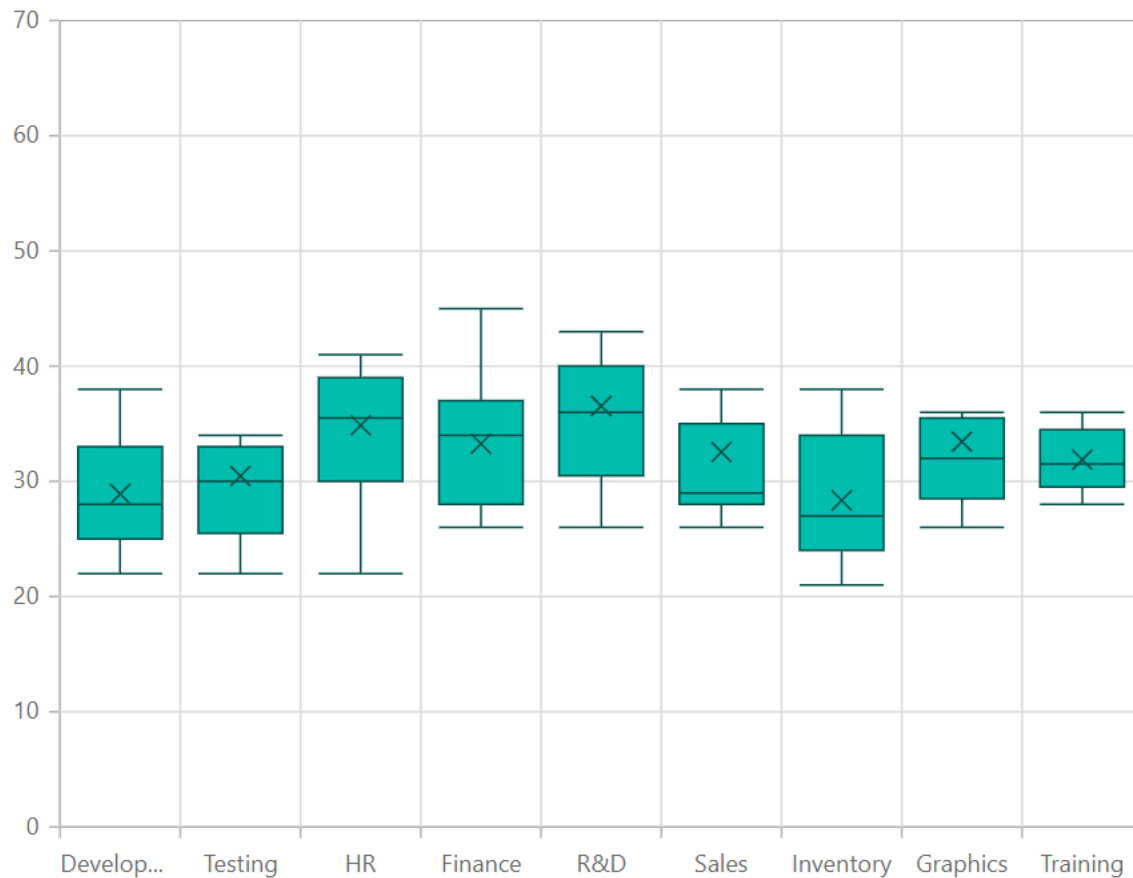
Box and Whisker in Blazor Charts Component

Box and Whisker

[Box and Whisker Chart](#) is used to visualize the variation in a set of data and it can be rendered by specifying the series [Type](#) as [BoxAndWhisker](#). The property [YName](#) requires a number of data or it should contain minimum of five values.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart>
<ChartPrimaryXAxis ValueType="Syncfusion.Blazor.Charts.ValueType.Category">
</ChartPrimaryXAxis>
<ChartSeriesCollection>
<ChartSeries DataSource="@ExpenseDetails" XName="XValue" YName="YValue"
Type="ChartSeriesType.BoxAndWhisker">
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code{
public class ChartData
{
public string XValue { get; set; }
public double[] YValue { get; set; }
}
public List<ChartData> ExpenseDetails = new List<ChartData>
{
new ChartData { XValue = "Development", YValue = new double[]{ 22, 22, 23,
25, 25, 25, 26, 27, 27, 28, 28, 29, 30, 32, 34, 32, 34, 36, 35, 38 } },
new ChartData { XValue = "Testing", YValue = new double[] { 22, 33, 23, 25,
26, 28, 29, 30, 34, 33, 32, 31, 50 } },
new ChartData { XValue = "HR", YValue = new double[] { 22, 24, 25, 30, 32,
34, 36, 38, 39, 41, 35, 36, 40, 56 } },
new ChartData { XValue = "Finance", YValue = new double[] { 26, 27, 28, 30,
32, 34, 35, 37, 35, 37, 45 } },
new ChartData { XValue = "R&D", YValue = new double[] { 26, 27, 29, 32, 34,
35, 36, 37, 38, 39, 41, 43, 58 } },
new ChartData { XValue = "Sales", YValue = new double[] { 27, 26, 28, 29,
29, 29, 32, 35, 32, 38, 53 } },
new ChartData { XValue = "Inventory", YValue = new double[] { 21, 23, 24,
25, 26, 27, 28, 30, 34, 36, 38 } },
new ChartData { XValue = "Graphics", YValue = new double[] { 26, 28, 29, 30,
32, 33, 35, 36, 52 } },
new ChartData { XValue = "Training", YValue = new double[] { 28, 29, 30, 31,
32, 34, 35, 36 } }
};
}
```



Refer to our [Blazor Box and Whisker Charts](#) feature tour page to know about its other groundbreaking feature representations. Explore our [Blazor Box and Whisker Chart Example](#) to know how to render and configure the box and whisker type charts.

Box Plot

To change the rendering mode of the Box and Whisker series, use the [BoxPlotMode](#) property. The default [BoxPlotMode](#) is [Exclusive](#).

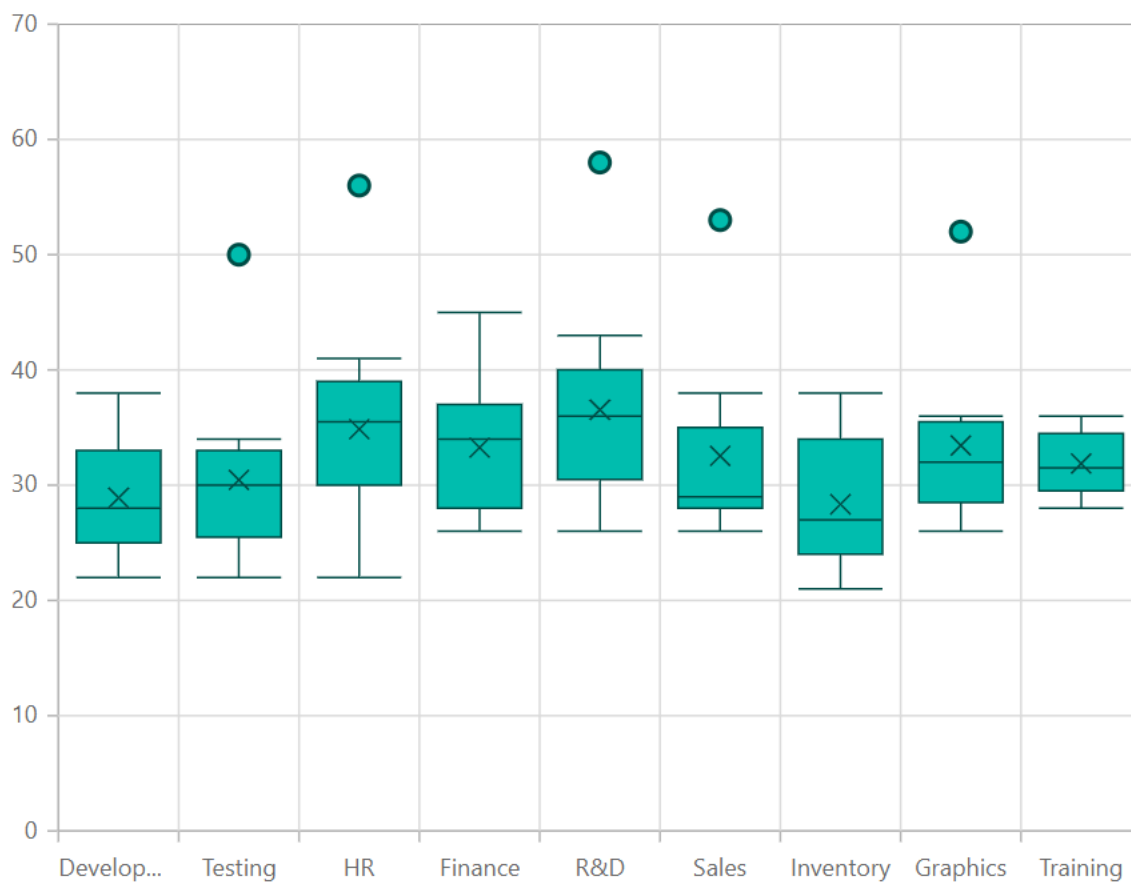
ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart>
<ChartPrimaryXAxis Value="Syncfusion.Blazor.Charts.ValueType.Category">
</ChartPrimaryXAxis>
<ChartSeriesCollection>
<ChartSeries DataSource="@ExpenseDetails" XName="XValue" YName="YValue"
Type="ChartSeriesType.BoxAndWhisker">
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code{
public class ChartData
{
public string XValue {get; set;}
public double[] YValue {get; set; }
}
```

```

public List<ChartData> ExpenseDetails = new List<ChartData>
{
    new ChartData { XValue = "Development", YValue = new double[] { 22, 22, 23,
    25, 25, 25, 26, 27, 27, 28, 28, 29, 30, 32, 34, 32, 34, 36, 35, 38 } },
    new ChartData { XValue = "Testing", YValue = new double[] { 22, 33, 23, 25,
    26, 28, 29, 30, 34, 33, 32, 31, 50 } },
    new ChartData { XValue = "HR", YValue = new double[] { 22, 24, 25, 30, 32,
    34, 36, 38, 39, 41, 35, 36, 40, 56 } },
    new ChartData { XValue = "Finance", YValue = new double[] { 26, 27, 28, 30,
    32, 34, 35, 37, 35, 37, 45 } },
    new ChartData { XValue = "R&D", YValue = new double[] { 26, 27, 29, 32, 34,
    35, 36, 37, 38, 39, 41, 43, 58 } },
    new ChartData { XValue = "Sales", YValue = new double[] { 27, 26, 28, 29,
    29, 29, 32, 35, 32, 38, 53 } },
    new ChartData { XValue = "Inventory", YValue = new double[] { 21, 23, 24,
    25, 26, 27, 28, 30, 34, 36, 38 } },
    new ChartData { XValue = "Graphics", YValue = new double[] { 26, 28, 29, 30,
    32, 33, 35, 36, 52 } },
    new ChartData { XValue = "Training", YValue = new double[] { 28, 29, 30, 31,
    32, 34, 35, 36 } }
};
}

```

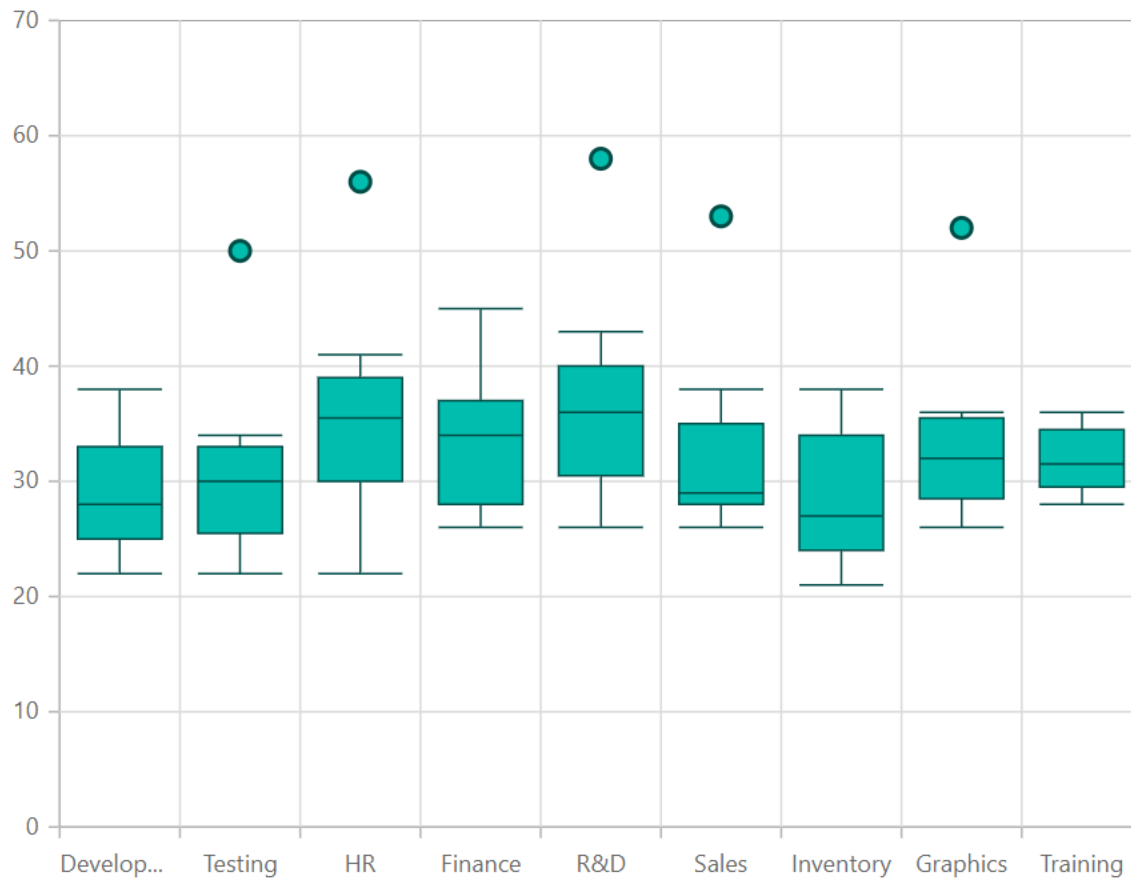


Show Mean

In Box and Whisker series, [ShowMean](#) property is used to show the box and whisker average value. The default value of [ShowMean](#) is **false**.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart>
<ChartPrimaryXAxis ValueType="Syncfusion.Blazor.Charts.ValueType.Category">
</ChartPrimaryXAxis>
<ChartSeriesCollection>
<ChartSeries DataSource="@ExpenseDetails" XName="XValue" YName="YValue"
Type="ChartSeriesType.BoxAndWhisker" ShowMean="false">
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code{
public class ChartData
{
public string XValue { get; set; }
public double[] YValue { get; set; }
}
public List<ChartData> ExpenseDetails = new List<ChartData>
{
new ChartData { XValue = "Development", YValue = new double[]{ 22, 22, 23,
25, 25, 25, 26, 27, 27, 28, 28, 29, 30, 32, 34, 32, 34, 36, 35, 38 } },
new ChartData { XValue = "Testing", YValue = new double[] { 22, 33, 23, 25,
26, 28, 29, 30, 34, 33, 32, 31, 50 } },
new ChartData { XValue = "HR", YValue = new double[] { 22, 24, 25, 30, 32,
34, 36, 38, 39, 41, 35, 36, 40, 56 } },
new ChartData { XValue = "Finance", YValue = new double[] { 26, 27, 28, 30,
32, 34, 35, 37, 35, 37, 45 } },
new ChartData { XValue = "R&D", YValue = new double[] { 26, 27, 29, 32, 34,
35, 36, 37, 38, 39, 41, 43, 58 } },
new ChartData { XValue = "Sales", YValue = new double[] { 27, 26, 28, 29,
29, 29, 32, 35, 32, 38, 53 } },
new ChartData { XValue = "Inventory", YValue = new double[] { 21, 23, 24,
25, 26, 27, 28, 30, 34, 36, 38 } },
new ChartData { XValue = "Graphics", YValue = new double[] { 26, 28, 29, 30,
32, 33, 35, 36, 52 } },
new ChartData { XValue = "Training", YValue = new double[] { 28, 29, 30, 31,
32, 34, 35, 36 } }
};
}
```

Refer to our [Blazor Charts](#) feature tour page for its groundbreaking feature representations and also explore our [Blazor Chart Example](#) to know various chart types and how to represent time-dependent data, showing trends at equal intervals.

See Also

- [Data Label](#)
- [Tooltip](#)

Waterfall in Blazor Charts Component

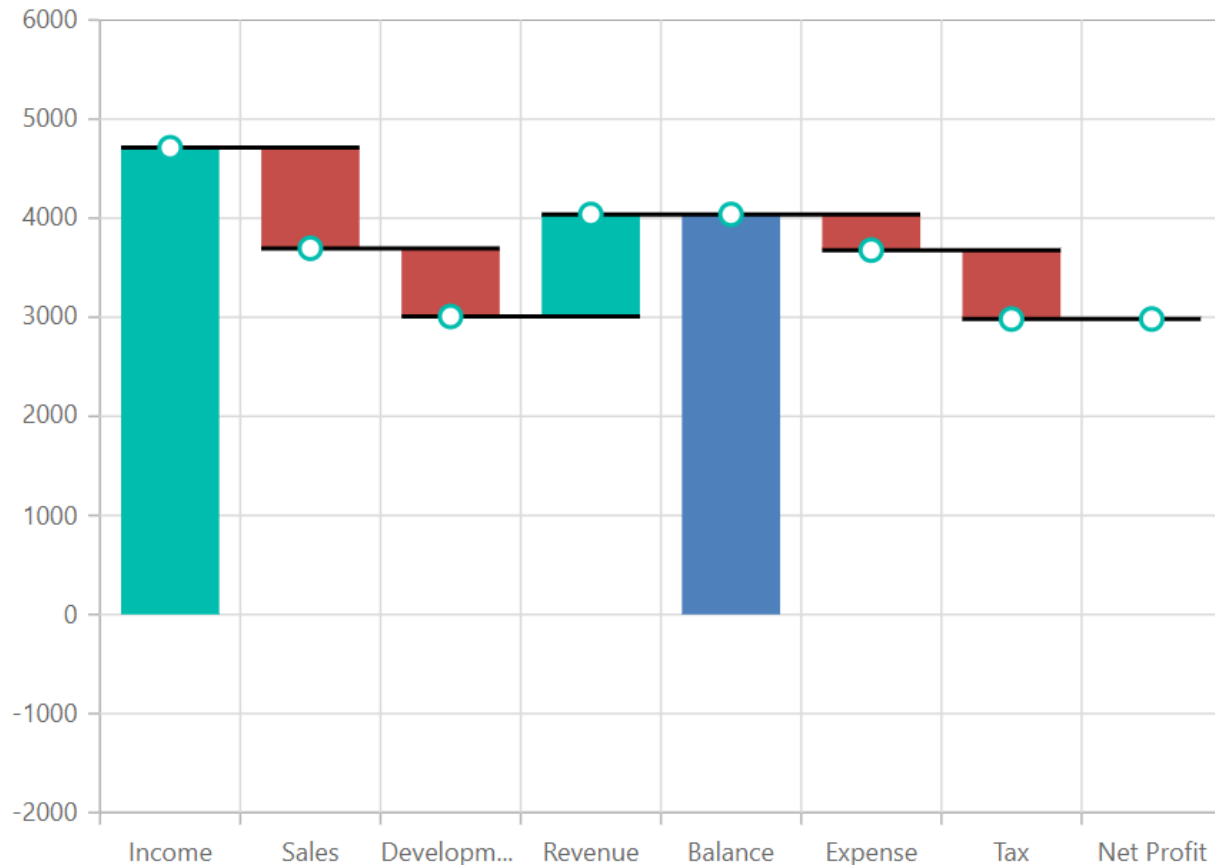
Waterfall

Waterfall Chart helps to understand the cumulative effect of the sequentially introduced positive and negative values. To render a waterfall series, set the series [Type](#) as **Waterfall**. [IntermediateSumIndexes](#) property of waterfall is used to represent the in-between sum values and [SumIndexes](#) is used to represent the cumulative sum values.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart>
<ChartPrimaryXAxis ValueTypes="Syncfusion.Blazor.Charts.ValueType.Category">
</ChartPrimaryXAxis>
<ChartSeriesCollection>
```

```
<ChartSeries DataSource="@SalesReports" XName="XValue" YName="YValue"
Type="ChartSeriesType.Waterfall" IntermediateSumIndexes="@index"
SumIndexes="@sumIndex">
<ChartMarker Height="10" Width="10" Visible="true"></ChartMarker>
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code{
double[] index = new double[] { 4 };
double[] sumIndex = new double[] { 8 };
public class ChartData
{
public string XValue { get; set; }
public double YValue { get; set; }
}
public List<ChartData> SalesReports = new List<ChartData>
{
new ChartData { XValue = "Income", YValue = 4711 },
new ChartData { XValue = "Sales", YValue = -1015 },
new ChartData { XValue = "Development", YValue = -688 },
new ChartData { XValue = "Revenue", YValue = 1030 },
new ChartData { XValue = "Balance" },
new ChartData { XValue = "Expense", YValue = -361 },
new ChartData { XValue = "Tax", YValue = -695 },
new ChartData { XValue = "Net Profit" },
};
}
```



Explore our [Blazor Waterfall Chart Example](#) to know how to render and configure the Waterfall type chart.

Series Customization

The negative changes of waterfall charts is represented by using [NegativeFillColor](#) and the summary changes are represented by using [SummaryFillColor](#) properties respectively. By default, the [NegativeFillColor](#) is #E94649 and the [SummaryFillColor](#) is #4E81BC.

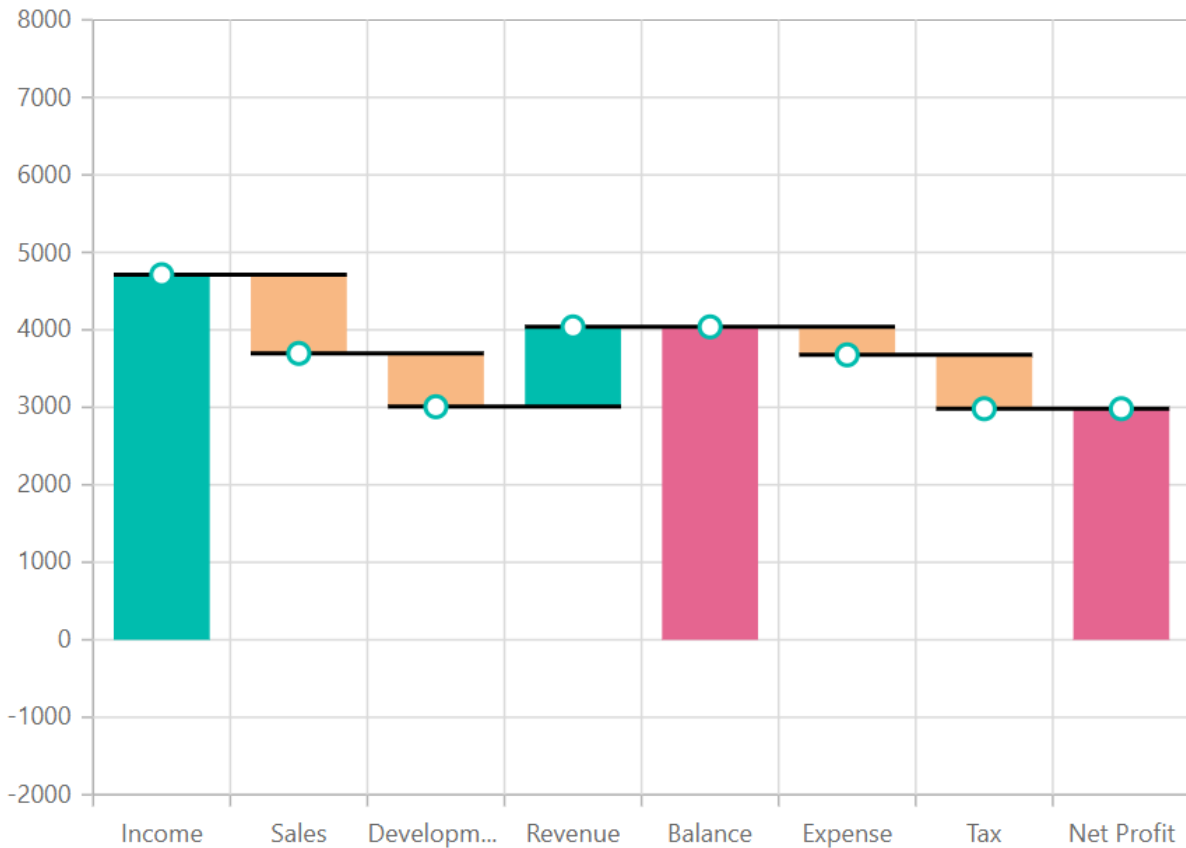
ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart>
<ChartPrimaryXAxis ValueType="Syncfusion.Blazor.Charts.ValueType.Category">
</ChartPrimaryXAxis>
<ChartSeriesCollection>
<ChartSeries DataSource="@ExpenseDetails" XName="XValue"
SummaryFillColor="#e56590" NegativeFillColor="#f8b883"
YName="YValue" Type="ChartSeriesType.Waterfall"
IntermediateSumIndexes="@intermediateSumIndexes" SumIndexes="@sumIndexes">
<ChartMarker Height="10" Width="10" Visible="true"></ChartMarker>
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code{
double[] intermediateSumIndexes = new double[] { 4 };
double[] sumIndexes = new double[] { 7 };
public class ChartData
```

```

{
    public string XValue;
    public double YValue;
}
public List<ChartData> ExpenseDetails = new List<ChartData>
{
    new ChartData { XValue = "Income", YValue = 4711 },
    new ChartData { XValue = "Sales", YValue = -1015 },
    new ChartData { XValue = "Development", YValue = -688 },
    new ChartData { XValue = "Revenue", YValue = 1030 },
    new ChartData { XValue = "Balance", YValue = 0 },
    new ChartData { XValue = "Expense", YValue = -361 },
    new ChartData { XValue = "Tax", YValue = -695 },
    new ChartData { XValue = "Net Profit", YValue = 0 },
};
}

```



Refer to our [Blazor Charts](#) feature tour page for its groundbreaking feature representations and also explore our [Blazor Chart Example](#) to know various chart types and how to represent time-dependent data, showing trends at equal intervals.

See Also

- [Data Label](#)
- [Tooltip](#)

Histogram in Blazor Charts Component

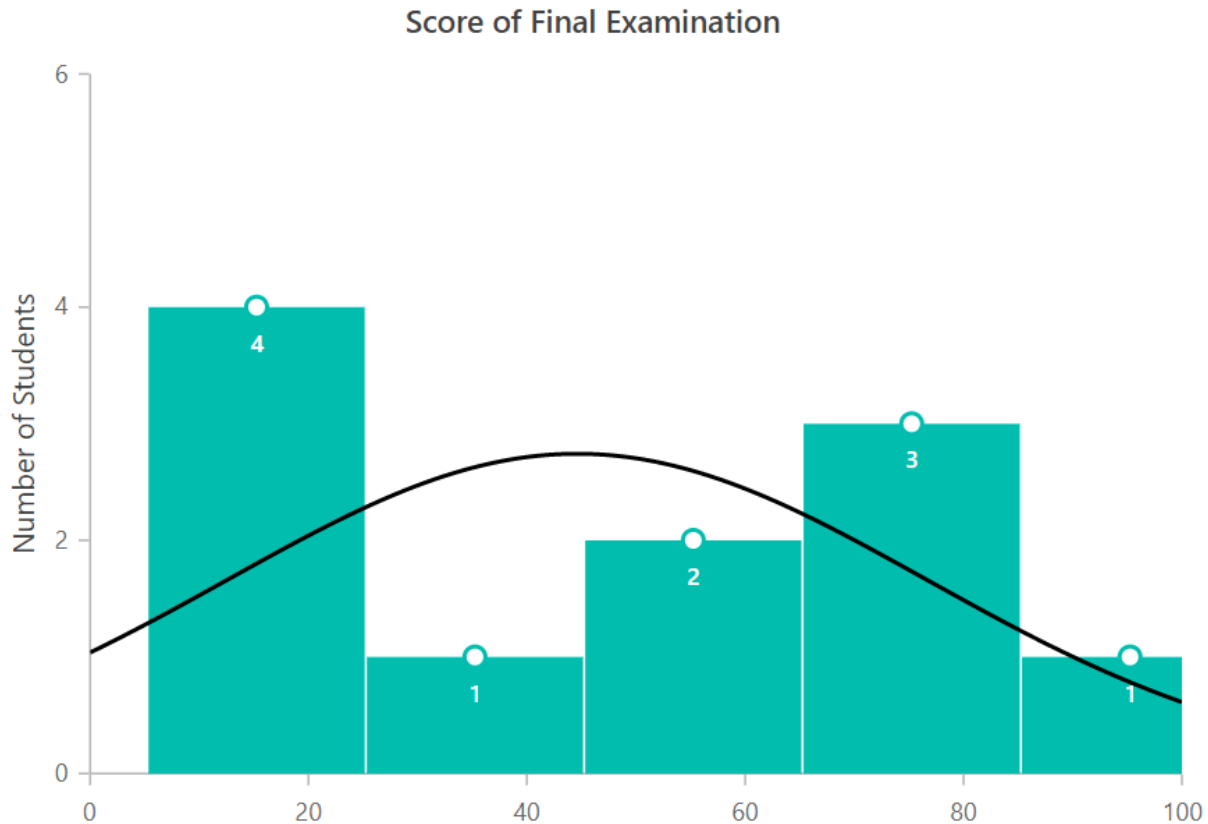
Histogram

[Histogram Chart](#) can provide a visual display of large amounts of data that are difficult to understand in a tabular or spreadsheet form and it can be rendered by specifying the series [Type](#) to [Histogram](#).

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart Title="Score of Final Examination">
  <ChartArea><ChartAreaBorder Width="0"></ChartAreaBorder></ChartArea>
  <ChartPrimaryXAxis Minimum="0" Maximum="100">
    <ChartAxisMajorGridLines Width="0"></ChartAxisMajorGridLines>
  </ChartPrimaryXAxis>
  <ChartPrimaryYAxis Minimum="0" Maximum="6" Interval="2" Title="Number of
  Students">
    <ChartAxisMajorGridLines Width="0"></ChartAxisMajorGridLines>
  </ChartPrimaryYAxis>
  <ChartSeriesCollection>
    <ChartSeries DataSource="@ExamScores" YName="Y"
    Type="ChartSeriesType.Histogram" BinInterval="20"
    ShowNormalDistribution="true" ColumnWidth="0.99">
      <ChartMarker Visible="true" Height="10" Width="10">
        <ChartDataLabel Visible="true"
        Position="Syncfusion.Blazor.Charts.LabelPosition.Top">
          <ChartDataLabelFont Color="#ffffff" FontWeight="600"></ChartDataLabelFont>
        </ChartDataLabel>
      </ChartMarker>
    </ChartSeries>
  </ChartSeriesCollection>
  <ChartTooltipSettings Enable="true"></ChartTooltipSettings>
</SfChart>

@code{
public class Data
{
  public double Y { get; set; }
}
public List<Data> ExamScores = new List<Data>
{
  new Data { Y=5.250},
  new Data { Y=7.750},
  new Data { Y=8.275},
  new Data { Y=9.750},
  new Data { Y=36.250},
  new Data { Y=46.250},
  new Data { Y=56.250},
  new Data { Y=66.500},
  new Data { Y=76.625},
  new Data { Y=80.000},
  new Data { Y=97.750}
};
}
```



Refer to our [Blazor Charts](#) feature tour page for its groundbreaking feature representations and also explore our [Blazor Chart Example](#) to know various chart types and how to represent time-dependent data, showing trends at equal intervals.

See Also

- [Data Label](#)
- [Tooltip](#)

Error Bar in Blazor Charts Component

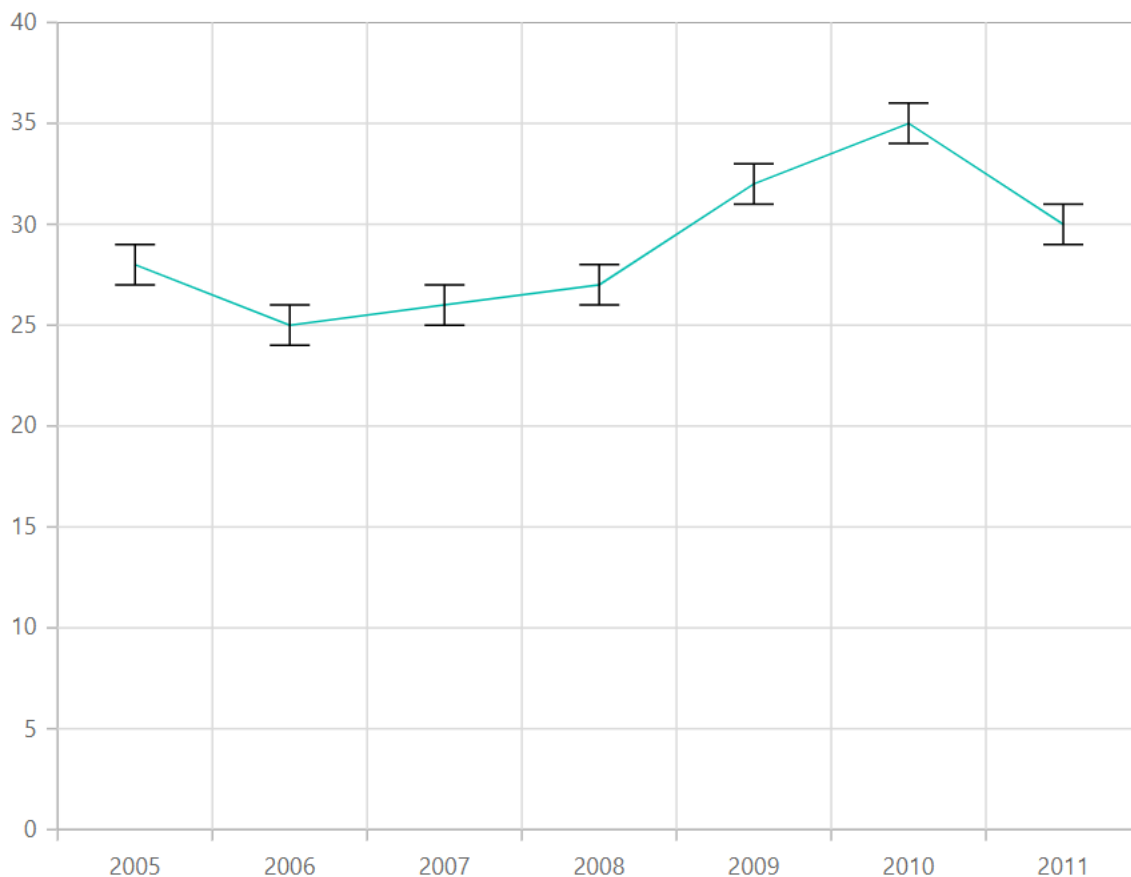
Error Bar

Error bars are graphical representations of the variability of data that are used on graphs to indicate the error or uncertainty in a reported measurement. To render the error bar for the series, set [Visible](#) property to **true** in [ChartErrorBarSettings](#).

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart>
<ChartPrimaryXAxis ValueType="Syncfusion.Blazor.Charts.ValueType.Category">
</ChartPrimaryXAxis>
<ChartSeriesCollection>
<ChartSeries DataSource="@SalesReports" XName="X" YName="Y"
Type="ChartSeriesType.Line">
<ChartErrorBarSettings Visible="true">
</ChartErrorBarSettings>
```

```
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code{
public class ChartData
{
public double X { get; set; }
public double Y {get; set; }
}
public List<ChartData> SalesReports = new List<ChartData>
{
new ChartData{ X= 2005, Y= 28 },
new ChartData{ X= 2006, Y= 25 },
new ChartData{ X= 2007, Y= 26 },
new ChartData{ X= 2008, Y= 27 },
new ChartData{ X= 2009, Y= 32 },
new ChartData{ X= 2010, Y= 35 },
new ChartData{ X= 2011, Y= 30 }
};
}
```



Error Bar Type

To change the error bar rendering type, set [Type](#) property in [ChartErrorBarSettings](#) and to change the error bar line length, use [VerticalError](#) property in [ChartErrorBarSettings](#).

ASPX-CS

```

@using Syncfusion.Blazor.Charts
<SfChart>
<ChartPrimaryXAxis ValueType="Syncfusion.Blazor.Charts.ValueType.Category">
</ChartPrimaryXAxis>
<ChartSeriesCollection>
<ChartSeries DataSource="@SalesReports" XName="X" YName="Y"
Type="ChartSeriesType.Line">
<ChartErrorBarSettings Visible="true" Type="ErrorBarType.Percentage"
VerticalError="4">
</ChartErrorBarSettings>
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code{
public class ChartData
{
public double X { get; set; }
public double Y { get; set; }
}
public List<ChartData> SalesReports = new List<ChartData>
{
new ChartData{ X= 2005, Y= 28 },
new ChartData{ X= 2006, Y= 25 },
new ChartData{ X= 2007, Y= 26 },
new ChartData{ X= 2008, Y= 27 },
new ChartData{ X= 2009, Y= 32 },
new ChartData{ X= 2010, Y= 35 },
new ChartData{ X= 2011, Y= 30 }
};
}

```

Customizing Error Bar Type

To customize the error bar type, specify the [Type](#) property to [Custom](#) and then change the [HorizontalNegativeError](#) and [HorizontalPositiveError](#) properties in [ChartErrorBarSettings](#).

ASPX-CS

```

@using Syncfusion.Blazor.Charts
<SfChart>
<ChartPrimaryXAxis ValueType="Syncfusion.Blazor.Charts.ValueType.Category">
</ChartPrimaryXAxis>
<ChartSeriesCollection>
<ChartSeries DataSource="@SalesReports" XName="X" YName="Y"
Type="ChartSeriesType.Line">
<ChartErrorBarSettings Visible="true" Type="ErrorBarType.Custom"
Mode="ErrorBarMode.Both"
VerticalPositiveError="2" HorizontalPositiveError="1"
VerticalNegativeError="2" HorizontalNegativeError="1">
</ChartErrorBarSettings>
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code{
public class ChartData
{
public double X { get; set; }
}

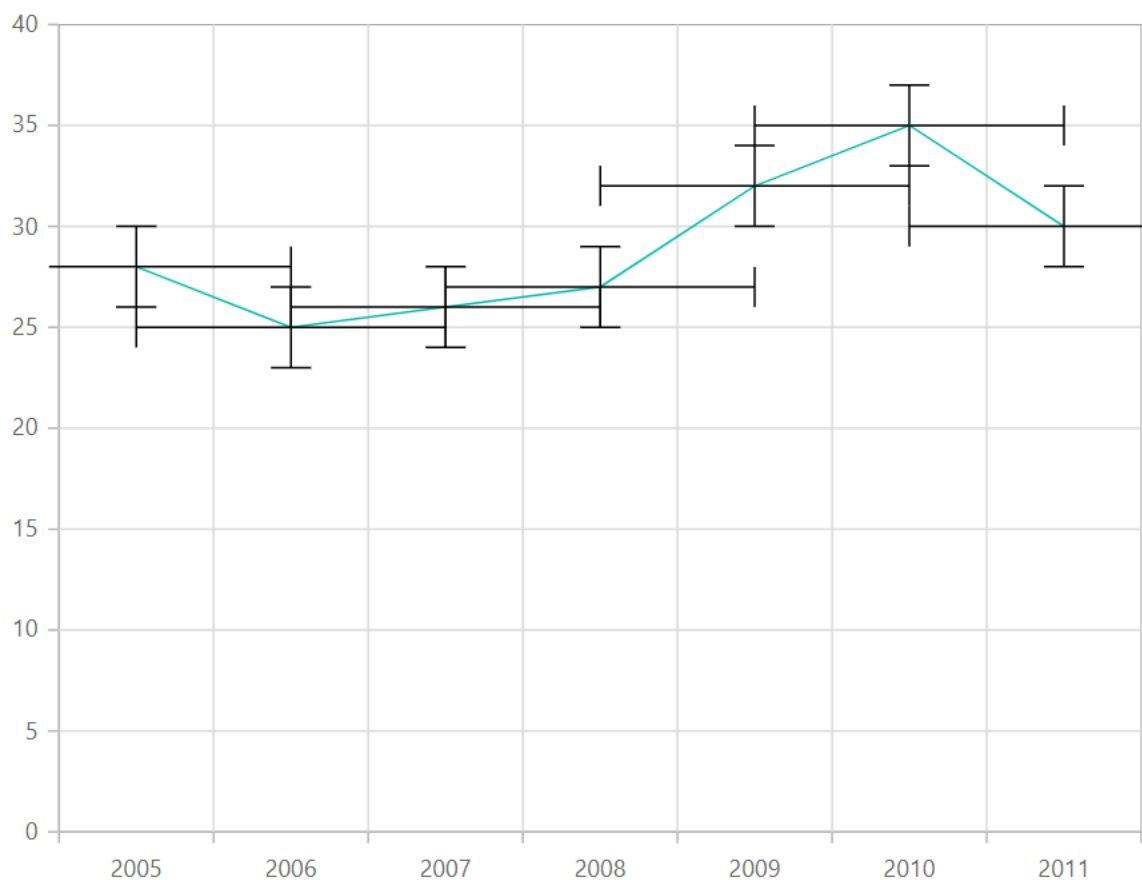
```



```

public double Y { get; set; }
}
public List<ChartData> SalesReports = new List<ChartData>
{
    new ChartData{ X= 2005, Y= 28 },
    new ChartData{ X= 2006, Y= 25 },
    new ChartData{ X= 2007, Y= 26 },
    new ChartData{ X= 2008, Y= 27 },
    new ChartData{ X= 2009, Y= 32 },
    new ChartData{ X= 2010, Y= 35 },
    new ChartData{ X= 2011, Y= 30 }
};
}

```



Error Bar Mode

Error bar mode is used to define whether the error bar line should be drawn horizontally, vertically, or on both sides. To change the error bar mode, use [Mode](#) property in [ChartErrorBarSettings](#).

ASPX-CS

```

@using Syncfusion.Blazor.Charts
<SfChart>
<ChartPrimaryXAxis ValueType="Syncfusion.Blazor.Charts.ValueType.Category">
</ChartPrimaryXAxis>

```

```

<ChartSeriesCollection>
<ChartSeries DataSource="@SalesReports" XName="X" YName="Y"
Type="ChartSeriesType.Line">
<ChartErrorBarSettings Visible="true" Mode="ErrorBarMode.Horizontal">
</ChartErrorBarSettings>
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code{
public class ChartData
{
public double X { get; set; }
public double Y { get; set; }
}
public List<ChartData> SalesReports = new List<ChartData>
{
new ChartData{ X= 2005, Y= 28 },
new ChartData{ X= 2006, Y= 25 },
new ChartData{ X= 2007, Y= 26 },
new ChartData{ X= 2008, Y= 27 },
new ChartData{ X= 2009, Y= 32 },
new ChartData{ X= 2010, Y= 35 },
new ChartData{ X= 2011, Y= 30 }
};
}

```

Error Bar Direction

To change the error bar direction to plus, minus, or both sides, use [Direction](#) property in [ChartErrorBarSettings](#).

ASPX-CS

```

@using Syncfusion.Blazor.Charts
<SfChart>
<ChartPrimaryXAxis ValueType="Syncfusion.Blazor.Charts.ValueType.Category">
</ChartPrimaryXAxis>
<ChartSeriesCollection>
<ChartSeries DataSource="@SalesReports" XName="X" YName="Y"
Type="ChartSeriesType.Line">
<ChartErrorBarSettings Visible="true" Mode="ErrorBarMode.Both"
Direction="ErrorBarDirection.Minus">
</ChartErrorBarSettings>
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code{
public class ChartData
{
public double X { get; set; }
public double Y { get; set; }
}
public List<ChartData> SalesReports = new List<ChartData>
{
new ChartData{ X= 2005, Y= 28 },
new ChartData{ X= 2006, Y= 25 },
new ChartData{ X= 2007, Y= 26 },

```

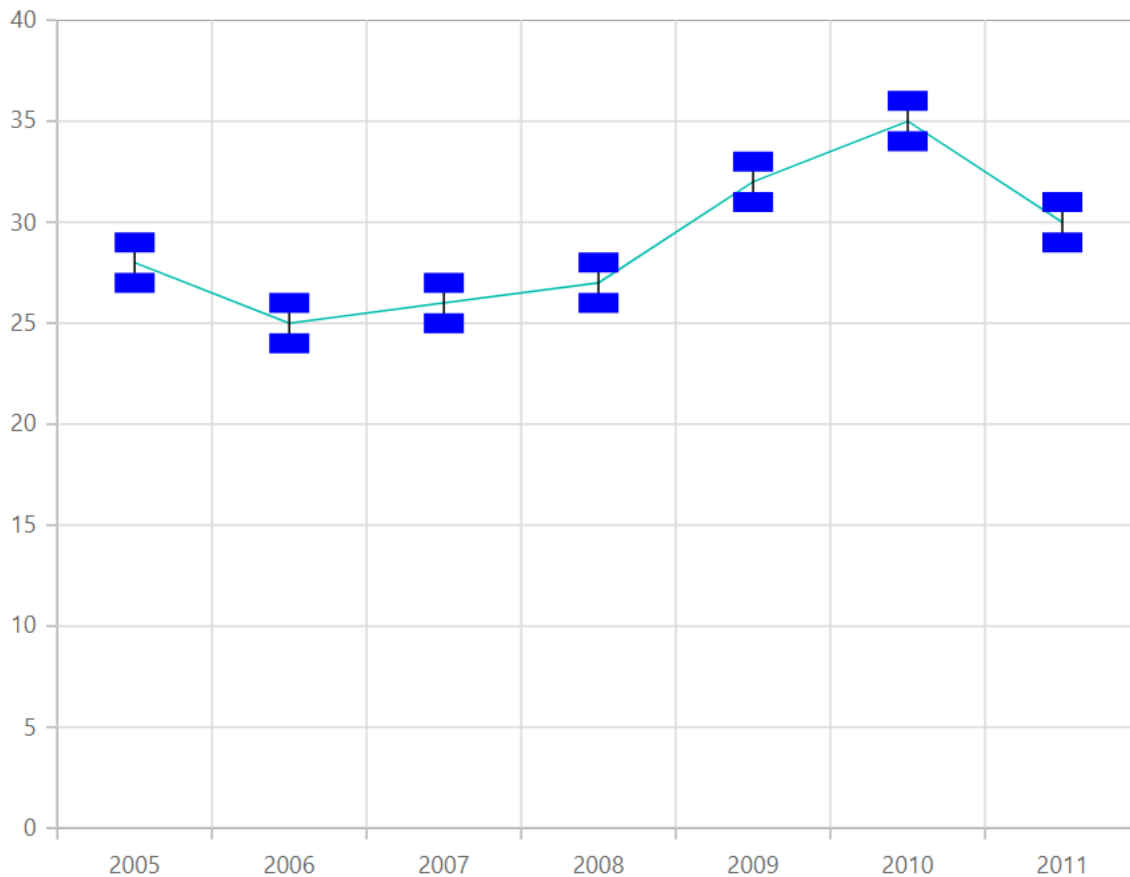
```
new ChartData{ X= 2008, Y= 27 },
new ChartData{ X= 2009, Y= 32 },
new ChartData{ X= 2010, Y= 35 },
new ChartData{ X= 2011, Y= 30 }
};
}
```

Customizing Error Bar Cap

To customize the error bar cap [Length](#), [Width](#) and [Color](#) properties in [ChartErrorBarCapSettings](#).

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart>
<ChartPrimaryXAxis ValueType="Syncfusion.Blazor.Charts.ValueType.Category">
</ChartPrimaryXAxis>
<ChartSeriesCollection>
<ChartSeries DataSource="@SalesReports" XName="X" YName="Y"
Type="ChartSeriesType.Line">
<ChartErrorBarSettings Visible="true">
<ChartErrorBarCapSettings Length="10" Width="10"
Color="#0000ff"></ChartErrorBarCapSettings>
</ChartErrorBarSettings>
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code{
public class ChartData
{
public double X { get; set; }
public double Y { get; set; }
}
public List<ChartData> SalesReports = new List<ChartData>
{
new ChartData{ X= 2005, Y= 28 },
new ChartData{ X= 2006, Y= 25 },
new ChartData{ X= 2007, Y= 26 },
new ChartData{ X= 2008, Y= 27 },
new ChartData{ X= 2009, Y= 32 },
new ChartData{ X= 2010, Y= 35 },
new ChartData{ X= 2011, Y= 30 }
};
}
```



Refer to our [Blazor Charts](#) feature tour page for its groundbreaking feature representations and also explore our [Blazor Chart Example](#) to know various chart types and how to represent time-dependent data, showing trends at equal intervals.

See Also

- [Data label](#)
- [Tooltip](#)

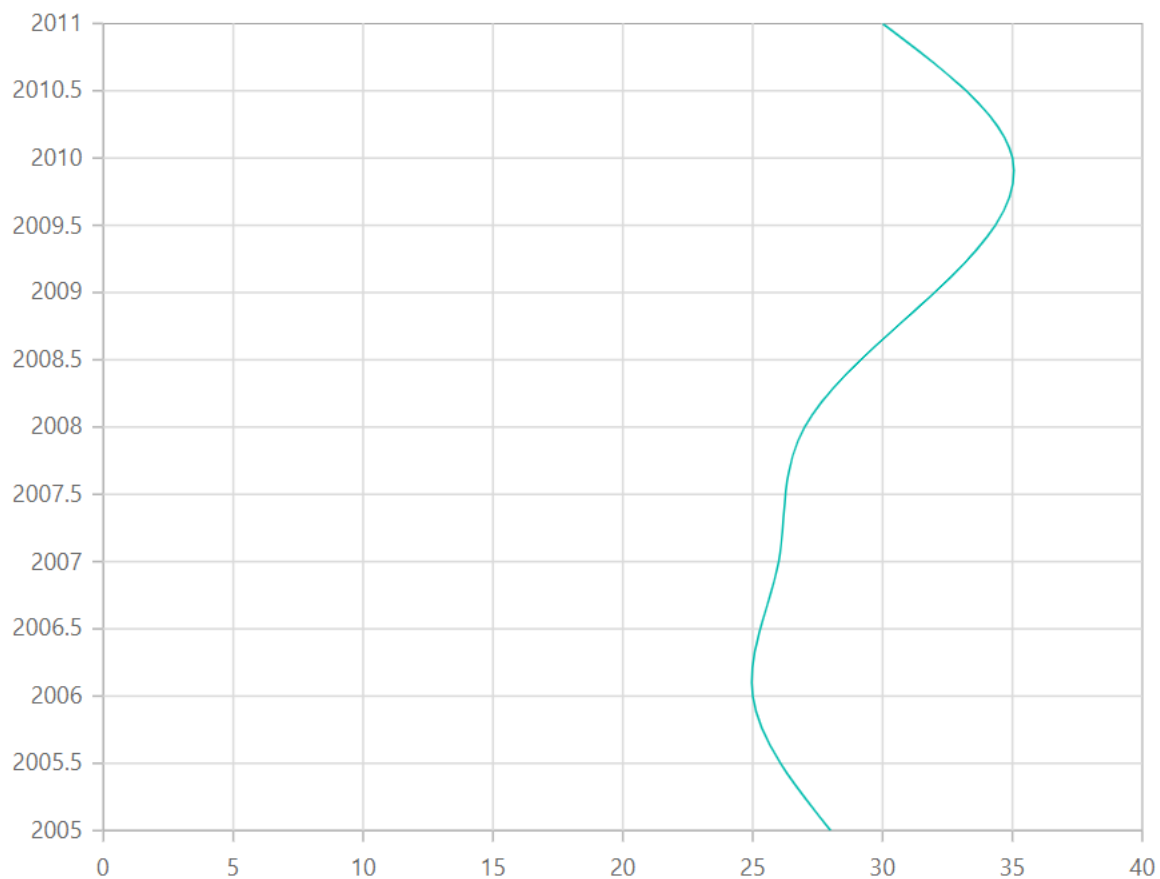
Vertical Chart in Blazor Charts Component

One can draw a vertical chart by changing the axis orientation, and all series types support this option. To render a chart vertically, use the [IsTransposed](#) property.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart IsTransposed="true">
  <ChartSeriesCollection>
    <ChartSeries DataSource="@SalesReports" XName="X" YName="Y"
      Type="ChartSeriesType.Spline">
    </ChartSeries>
  </ChartSeriesCollection>
</SfChart>
@code{
public class ChartData
```

```
{  
    public double X { get; set; }  
    public double Y { get; set; }  
}  
public List<ChartData> SalesReports = new List<ChartData>  
{  
    new ChartData{ X= 2005, Y= 28 },  
    new ChartData{ X= 2006, Y= 25 },  
    new ChartData{ X= 2007, Y= 26 },  
    new ChartData{ X= 2008, Y= 27 },  
    new ChartData{ X= 2009, Y= 32 },  
    new ChartData{ X= 2010, Y= 35 },  
    new ChartData{ X= 2011, Y= 30 }  
};  
}
```



Refer to our [Blazor Charts](#) feature tour page for its groundbreaking feature representations and also explore our [Blazor Chart Example](#) to know various chart types and how to represent time-dependent data, showing trends at equal intervals.

See Also

- [Data Label](#)
- [Tooltip](#)

Pareto in Blazor Charts Component

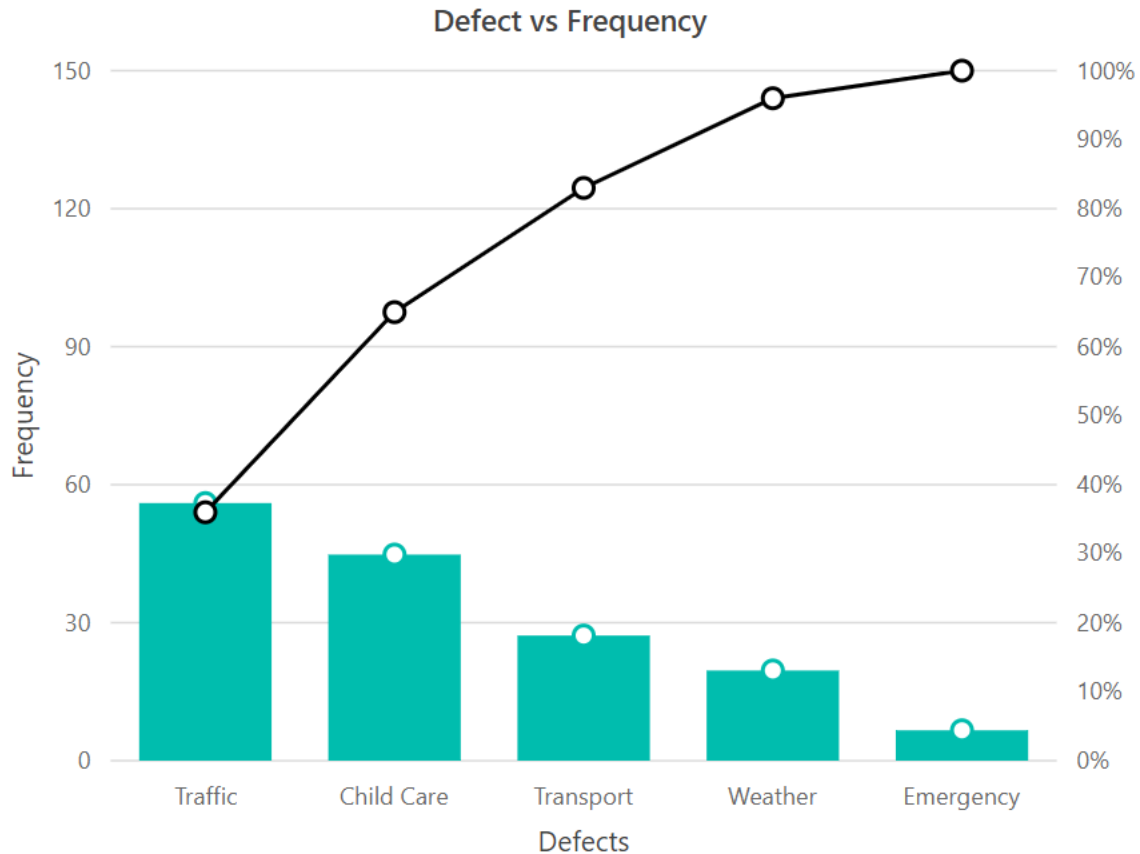
Pareto

[Pareto Chart](#) is used to find the cumulative values of data in different categories. It is a combination of [Column](#) and [Line](#) series. The initial values are represented by column chart and the cumulative values are represented by line chart. To render a pareto chart, set series [Type](#) to [Pareto](#).

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart Title="Defect vs Frequency">
  <ChartPrimaryXAxis ValueType="Syncfusion.Blazor.Charts.ValueType.Category"
    Interval="1" Title="Defects">
    <ChartAxisLineStyle Width="0"></ChartAxisLineStyle>
    <ChartAxisMajorTickLines Width="0"></ChartAxisMajorTickLines>
    <ChartAxisMajorGridLines Width="0"></ChartAxisMajorGridLines>
    <ChartAxisMinorTickLines Width="0"></ChartAxisMinorTickLines>
    <ChartAxisMinorGridLines Width="0"></ChartAxisMinorGridLines>
  </ChartPrimaryXAxis>
  <ChartPrimaryYAxis Title="Frequency" Minimum="0" Maximum="150"
    Interval="30">
    <ChartAxisLineStyle Width="0"></ChartAxisLineStyle>
    <ChartAxisMajorTickLines Width="0"></ChartAxisMajorTickLines>
    <ChartAxisMajorGridLines Width="1"></ChartAxisMajorGridLines>
    <ChartAxisMinorTickLines Width="0"></ChartAxisMinorTickLines>
    <ChartAxisMinorGridLines Width="1"></ChartAxisMinorGridLines>
  </ChartPrimaryYAxis>
  <ChartSeriesCollection>
    <ChartSeries DataSource="@Data" XName="X" YName="Y" Name="Defect" Width="2"
      Type="ChartSeriesType.Pareto">
      <ChartMarker Visible="true" Height="10" Width="10">
      </ChartMarker>
    </ChartSeries>
  </ChartSeriesCollection>
  <ChartLegendSettings Visible="false"></ChartLegendSettings>
</SfChart>

@code{
public class ChartData
{
public string X { get; set; }
public double Y { get; set; }
}
public List<ChartData> Data = new List<ChartData>
{
new ChartData { X= "Traffic", Y= 56 },
new ChartData { X= "Child Care", Y= 44.8 },
new ChartData { X= "Transport", Y= 27.2 },
new ChartData { X= "Weather", Y= 19.6 },
new ChartData { X= "Emergency", Y= 6.6 }
};
}
```



Refer to our [Blazor Charts](#) feature tour page for its groundbreaking feature representations and also explore our [Blazor Chart Example](#) to know various chart types and how to represent time-dependent data, showing trends at equal intervals.

See Also

- [Data Label](#)
- [Tooltip](#)

Mixed Chart in Blazor Charts Component

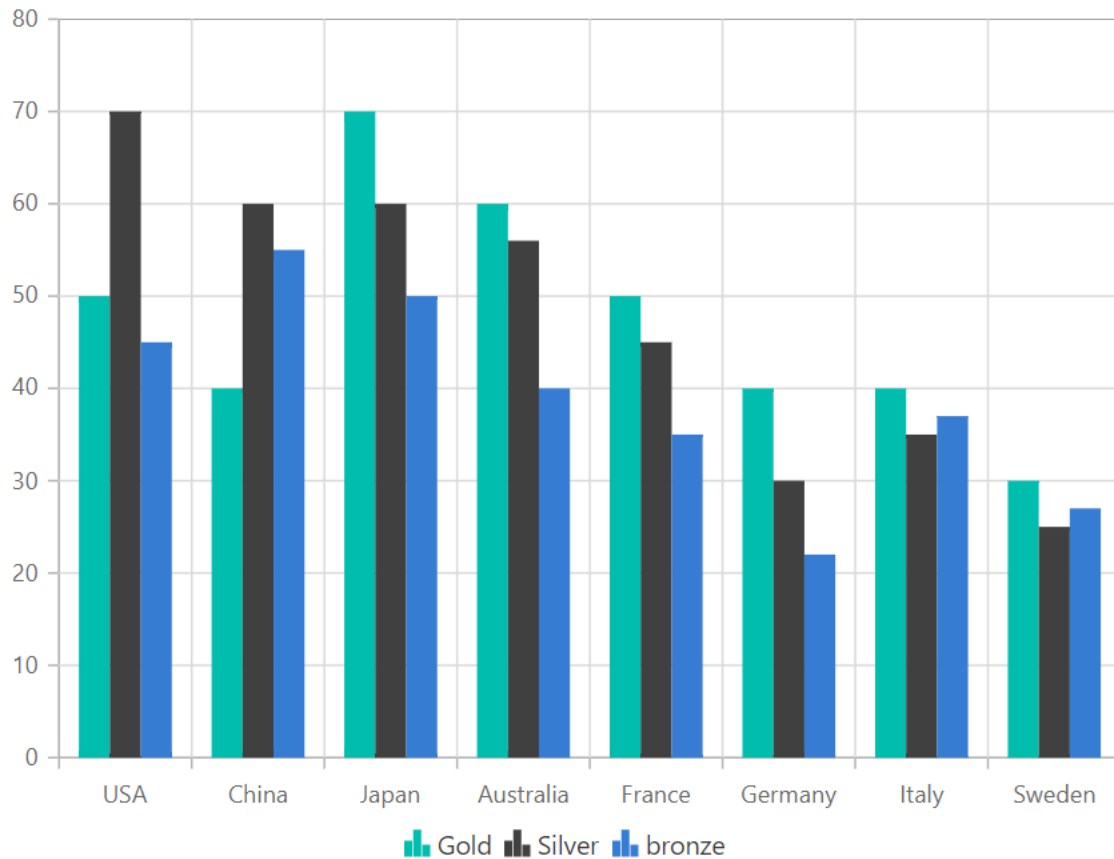
Multiple Chart Series

The [ChartSeries](#) property allows to add multiple series to the chart. The series are rendered in the order they were added to the series array.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart>
  <ChartPrimaryXAxis
    ValueType="Syncfusion.Blazor.Charts.ValueType.Category"></ChartPrimaryXAxis>
  <ChartSeriesCollection>
    <ChartSeries DataSource="@MedalDetails" Name="Gold" XName="Country"
      Width="2" Opacity="1" YName="Gold" Type="ChartSeriesType.Column">
    </ChartSeries>
```

```
<ChartSeries DataSource="@MedalDetails" Name="Silver" XName="Country"
Width="2" Opacity="1" YName="Silver" Type="ChartSeriesType.Column">
</ChartSeries>
<ChartSeries DataSource="@MedalDetails" Name="bronze" XName="Country"
Width="2" Opacity="1" YName="Bronze" Type="ChartSeriesType.Column">
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code{
public class ChartData
{
public string Country { get; set; }
public double Gold { get; set; }
public double Silver { get; set; }
public double Bronze { get; set; }
}
public List<ChartData> MedalDetails = new List<ChartData>
{
new ChartData{ Country= "USA", Gold=50, Silver=70, Bronze=45 },
new ChartData{ Country="China", Gold=40, Silver= 60, Bronze=55 },
new ChartData{ Country= "Japan", Gold=70, Silver= 60, Bronze=50 },
new ChartData{ Country= "Australia", Gold=60, Silver= 56, Bronze=40 },
new ChartData{ Country= "France", Gold=50, Silver= 45, Bronze=35 },
new ChartData{ Country= "Germany", Gold=40, Silver=30, Bronze=22 },
new ChartData{ Country= "Italy", Gold=40, Silver=35, Bronze=37 },
new ChartData{ Country= "Sweden", Gold=30, Silver=25, Bronze=27 }
};
}
```

Combination Chart Series

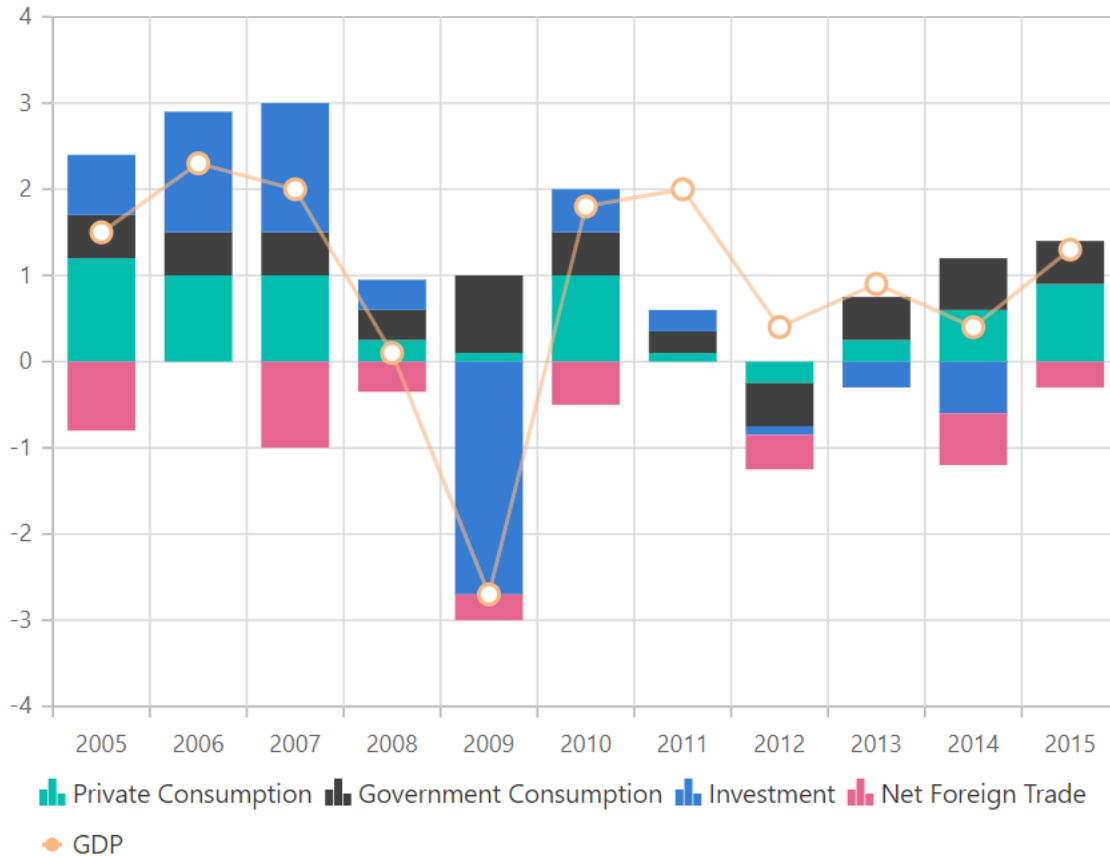
A chart can be created by combining several chart types such as line, column, and so on.

Bar series cannot be combined with any other series as the axis orientation is different from other series.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart>
<ChartPrimaryXAxis
ValueTypes="Syncfusion.Blazor.Charts.ValueType.Category"></ChartPrimaryXAxis>
<ChartSeriesCollection>
<ChartSeries DataSource="@SalesReports" Name="Private Consumption" XName="X"
YName="Y" Type="ChartSeriesType.StackingColumn">
</ChartSeries>
<ChartSeries DataSource="@SalesReports" Name="Government Consumption"
XName="X" YName="Y1" Type="ChartSeriesType.StackingColumn">
</ChartSeries>
<ChartSeries DataSource="@SalesReports" Name="Investment" XName="X"
YName="Y2" Type="ChartSeriesType.StackingColumn">
</ChartSeries>
<ChartSeries DataSource="@SalesReports" Name="Net Foreign Trade" XName="X"
YName="Y3" Type="ChartSeriesType.StackingColumn">
</ChartSeries>
<ChartSeries DataSource="@SalesReports" Name="GDP" XName="X" YName="Y4"
Width="2" Opacity="0.6" Type="ChartSeriesType.Line">
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
```

```
<ChartMarker Visible="true" Height="10" Width="10"></ChartMarker>
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code{
public class ChartData
{
public string X { get; set; }
public double Y { get; set; }
public double Y1 { get; set; }
public double Y2 { get; set; }
public double Y3 { get; set; }
public double Y4 { get; set; }
}
public List<ChartData> SalesReports = new List<ChartData>
{
new ChartData { X= "2005", Y= 1.2, Y1= 0.5, Y2= 0.7, Y3= -0.8, Y4= 1.5 },
new ChartData { X= "2006", Y= 1, Y1= 0.5, Y2= 1.4, Y3= 0, Y4= 2.3 },
new ChartData { X= "2007", Y= 1, Y1= 0.5, Y2= 1.5, Y3= -1, Y4= 2 },
new ChartData { X= "2008", Y= 0.25, Y1= 0.35, Y2= 0.35, Y3= -.35, Y4= 0.1 },
new ChartData { X= "2009", Y= 0.1, Y1= 0.9, Y2= -2.7, Y3= -0.3, Y4= -2.7 },
new ChartData { X= "2010", Y= 1, Y1= 0.5, Y2= 0.5, Y3= -0.5, Y4= 1.8 },
new ChartData { X= "2011", Y= 0.1, Y1= 0.25, Y2= 0.25, Y3= 0, Y4= 2 },
new ChartData { X= "2012", Y= -0.25, Y1= -0.5, Y2= -0.1, Y3= -0.4, Y4= 0.4
},
new ChartData { X= "2013", Y= 0.25, Y1= 0.5, Y2= -0.3, Y3= 0, Y4= 0.9 },
new ChartData { X= "2014", Y= 0.6, Y1= 0.6, Y2= -0.6, Y3= -0.6, Y4= 0.4 },
new ChartData { X= "2015", Y= 0.9, Y1= 0.5, Y2= 0, Y3= -0.3, Y4= 1.3 }
};
}
```



Refer to our [Blazor Charts](#) feature tour page for its groundbreaking feature representations and also explore our [Blazor Chart Example](#) to know various chart types and how to represent time-dependent data, showing trends at equal intervals.

See Also

- [Data Label](#)
- [Tooltip](#)
- [Legend](#)

Markers in Blazor Charts Component

[Data markers](#) are used to provide information about the data points in a series. Each data point can be adorned with a shape.

```
<!-- markdownlint-disable MD036 -->
```

Markers

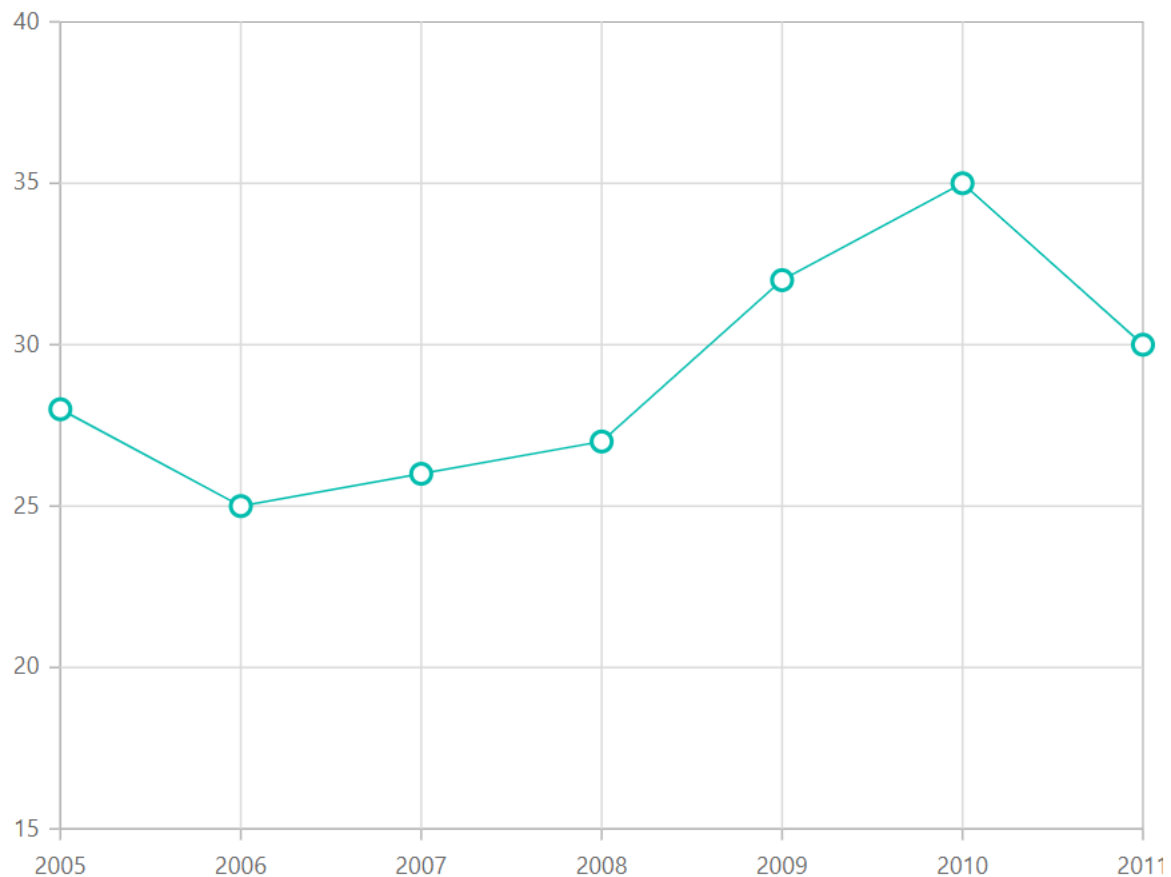
```
<!-- markdownlint-disable MD036 -->
```

Markers can be added to the points by enabling the [Visible](#) property to **true** of [ChartMarker](#).

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart>
<ChartSeriesCollection>
```

```
<ChartSeries DataSource="@ConsumerReports" XName="X" YName="Y"
Type="ChartSeriesType.Line">
  <ChartMarker Visible="true" Height="10" Width="10"/>
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code{
public class ChartData
{
    public double X { get; set; }
    public double Y { get; set; }
}
public List<ChartData> ConsumerReports = new List<ChartData>
{
    new ChartData{ X= 2005, Y= 28 },
    new ChartData{ X= 2006, Y= 25 },
    new ChartData{ X= 2007, Y= 26 },
    new ChartData{ X= 2008, Y= 27 },
    new ChartData{ X= 2009, Y= 32 },
    new ChartData{ X= 2010, Y= 35 },
    new ChartData{ X= 2011, Y= 30 }
};
}
```

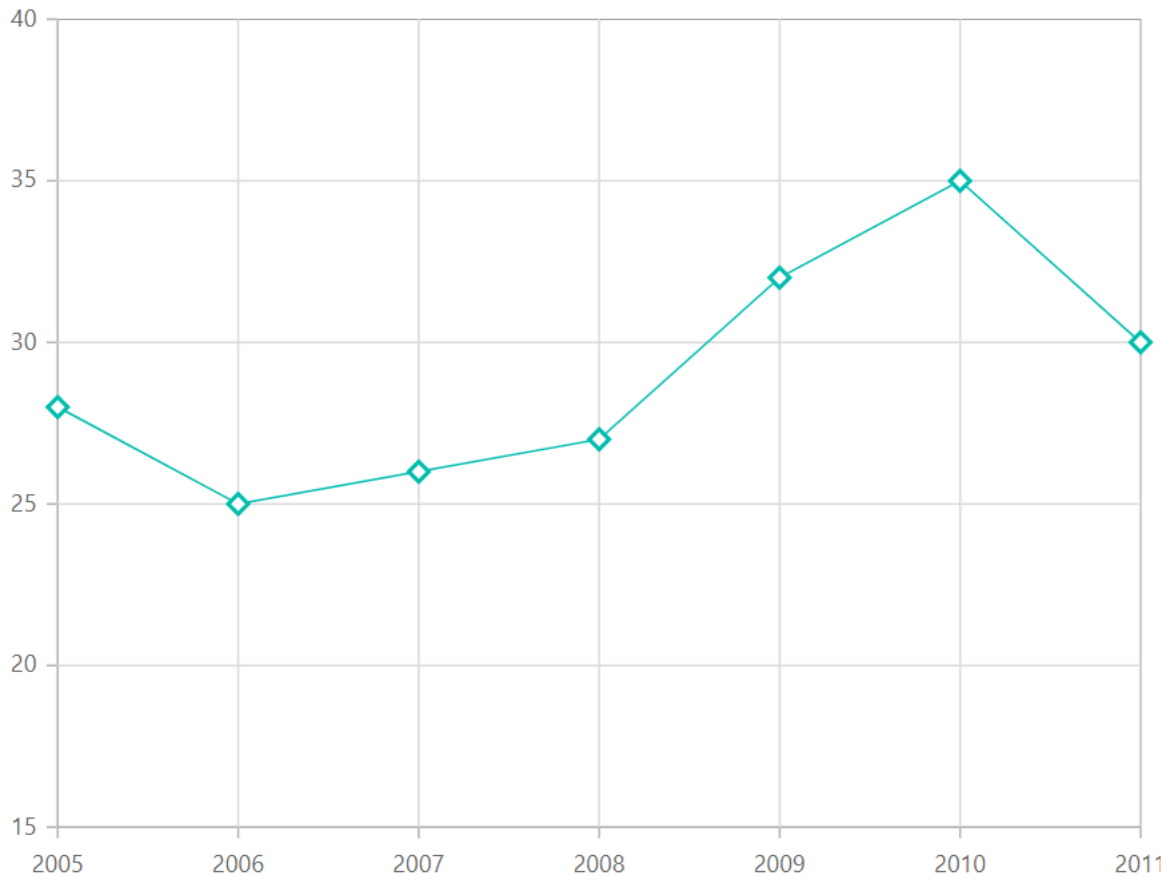


Shape

Markers can be assigned with different shapes such as [Rectangle](#), [Circle](#), [Diamond](#) etc. using the [Shape](#) property.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart>
<ChartSeriesCollection>
<ChartSeries DataSource="@ConsumerReports" XName="X" YName="Y"
Type="ChartSeriesType.Line">
<ChartMarker Visible="true" Height="10" Width="10"
Shape="ChartShape.Diamond"/>
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code{
public class ChartData
{
public double X { get; set; }
public double Y { get; set; }
}
public List<ChartData> ConsumerReports = new List<ChartData>
{
new ChartData{ X= 2005, Y= 28 },
new ChartData{ X= 2006, Y= 25 },
new ChartData{ X= 2007, Y= 26 },
new ChartData{ X= 2008, Y= 27 },
new ChartData{ X= 2009, Y= 32 },
new ChartData{ X= 2010, Y= 35 },
new ChartData{ X= 2011, Y= 30 }
};
}
```



Images

Apart from shapes, one can also add custom images to mark the data point using the [ImageUrl](#) property.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart>
<ChartSeriesCollection>
<ChartSeries DataSource="@ConsumerReports" XName="X" YName="Y"
Type="ChartSeriesType.Line">
<ChartMarker Visible="true" Height="10" Width="10"
ImageUrl="https://ej2.syncfusion.com/demos/src/chart/images/cloud.png">
</ChartMarker>
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code{
public class ChartData
{
public double X { get; set; }
public double Y { get; set; }
}
public List<ChartData> ConsumerReports = new List<ChartData>
{
new ChartData{ X= 2005, Y= 28 },
```

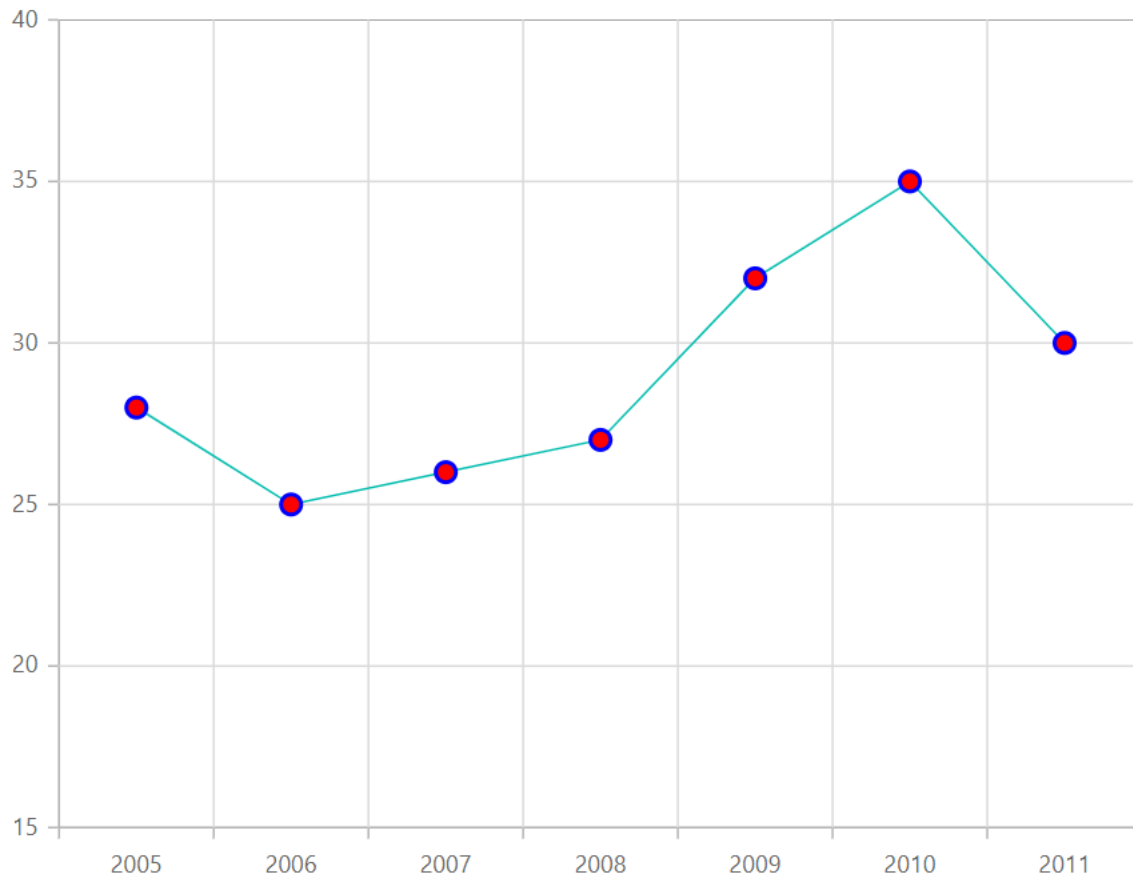
```
new ChartData{ X= 2006, Y= 25 },
new ChartData{ X= 2007, Y= 26 },
new ChartData{ X= 2008, Y= 27 },
new ChartData{ X= 2009, Y= 32 },
new ChartData{ X= 2010, Y= 35 },
new ChartData{ X= 2011, Y= 30 }
};
}
```

Customization

Markers color can be customized using [Fill](#) property and the border color and width can be customized to specified in [ChartMarkerBorder](#).

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart>
<ChartPrimaryXAxis ValueType="Syncfusion.Blazor.Charts.ValueType.Category">
</ChartPrimaryXAxis>
<ChartSeriesCollection>
<ChartSeries DataSource="@ConsumerReports" XName="X" YName="Y"
Type="ChartSeriesType.Line">
<ChartMarker Visible="true" Height="10" Width="10" Fill="red">
<ChartMarkerBorder Width="2" Color="blue"></ChartMarkerBorder>
</ChartMarker>
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code{
public class ChartData
{
public double X { get; set; }
public double Y { get; set; }
}
public List<ChartData> ConsumerReports = new List<ChartData>
{
new ChartData{ X= 2005, Y= 28 },
new ChartData{ X= 2006, Y= 25 },
new ChartData{ X= 2007, Y= 26 },
new ChartData{ X= 2008, Y= 27 },
new ChartData{ X= 2009, Y= 32 },
new ChartData{ X= 2010, Y= 35 },
new ChartData{ X= 2011, Y= 30 }
};
}
```



Refer to our [Blazor Charts](#) feature tour page for its groundbreaking feature representations and also explore our [Blazor Chart Example](#) to know various chart types and how to represent time-dependent data, showing trends at equal intervals.

See Also

- [Tooltip](#)
- [Legend](#)

Data Labels in Blazor Charts Component

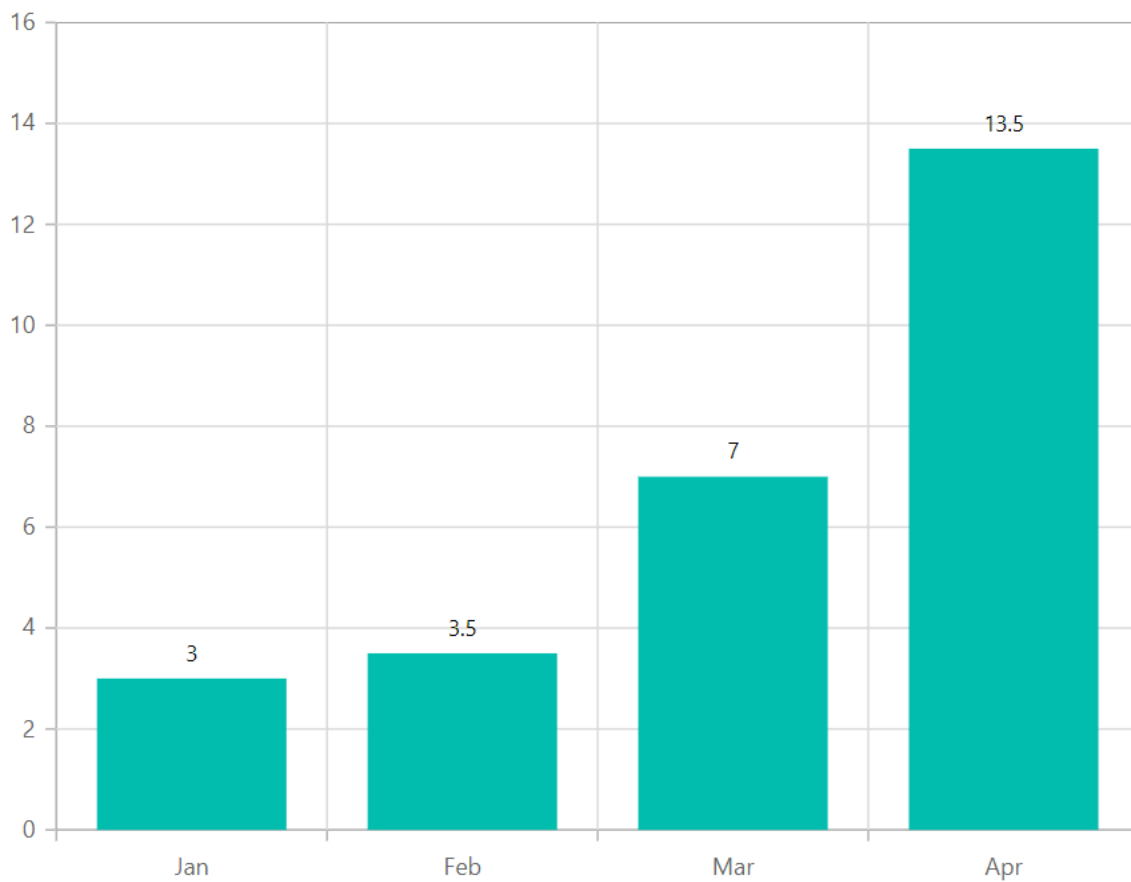
[Data label](#) can be added to a [ChartSeries](#) by enabling the [Visible](#) option in the data label settings. By default, the labels will organize themselves intelligently without overlapping.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart>
<ChartPrimaryXAxis ValueType="Syncfusion.Blazor.Charts.ValueType.Category"/>
<ChartSeriesCollection>
<ChartSeries DataSource="@WeatherReports" XName="X" YName="Y"
Type="ChartSeriesType.Column">
<ChartMarker>
<ChartDataLabel Visible="true"/>
</ChartMarker>
```



```
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code{
public class Data
{
public string X { get; set; }
public double Y { get; set; }
}
public List<Data> WeatherReports = new List<Data>
{
new Data{ X= "Jan", Y= 3 },
new Data{ X= "Feb", Y= 3.5 },
new Data{ X= "Mar", Y= 7 },
new Data{ X= "Apr", Y= 13.5 }
};
}
```



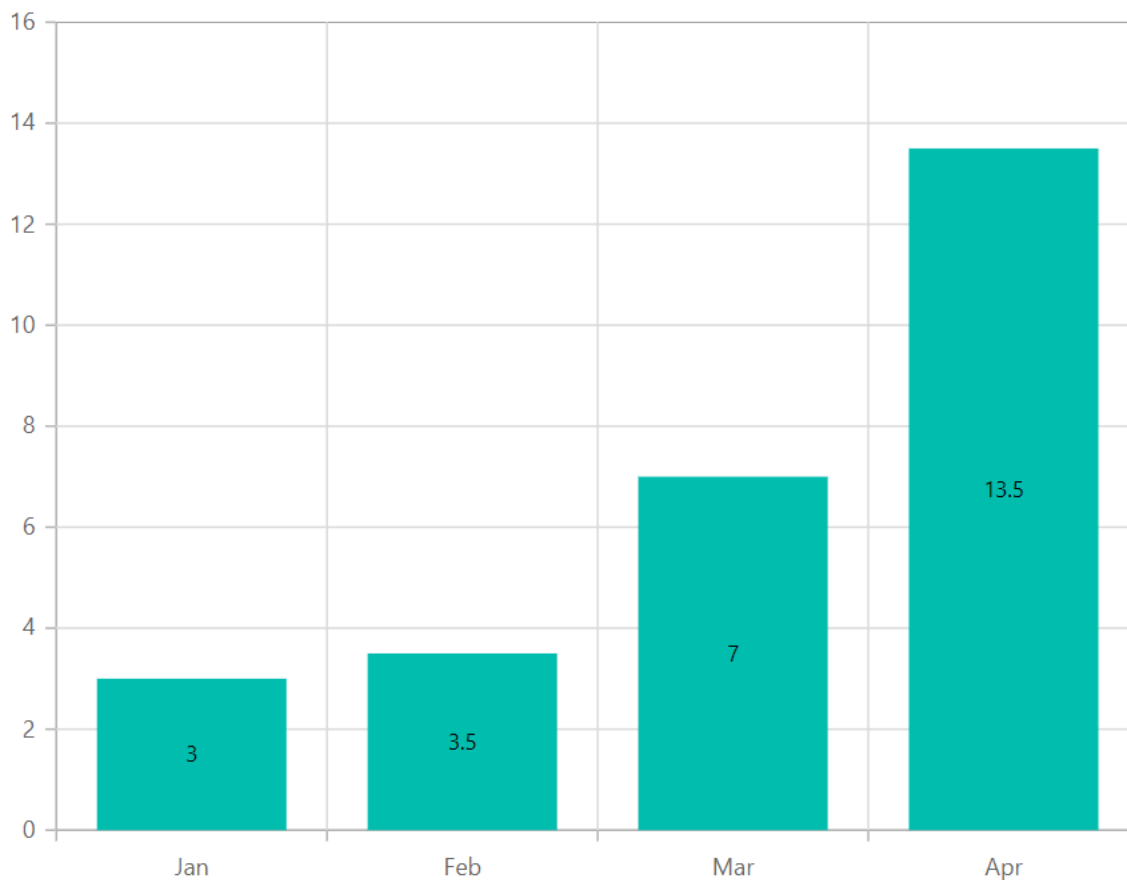
Position

Using [Position](#) property, the label can be placed either on [Top](#), [Middle](#), [Bottom](#) or [Outer](#).

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart>
```

```
<ChartPrimaryXAxis ValueType="Syncfusion.Blazor.Charts.ValueType.Category"
/>
<ChartSeriesCollection>
<ChartSeries DataSource="@WeatherReports" XName="X" YName="Y"
Type="ChartSeriesType.Column">
<ChartMarker>
<ChartDataLabel Visible="true"
Position="Syncfusion.Blazor.Charts.LabelPosition.Middle" />
</ChartMarker>
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code{
public class Data
{
public string X { get; set; }
public double Y { get; set; }
}
public List<Data> WeatherReports = new List<Data>
{
new Data{ X= "Jan", Y= 3 },
new Data{ X= "Feb", Y= 3.5 },
new Data{ X= "Mar", Y= 7 },
new Data{ X= "Apr", Y= 13.5 }
};
}
```



The position **Outer** is applicable only for column and bar series.

Template

Label content can be formatted by using the [Template](#) option. Inside the template, one can add the placeholder text **`\${point.x}`** and **`\${point.y}`** to display the corresponding data point value.

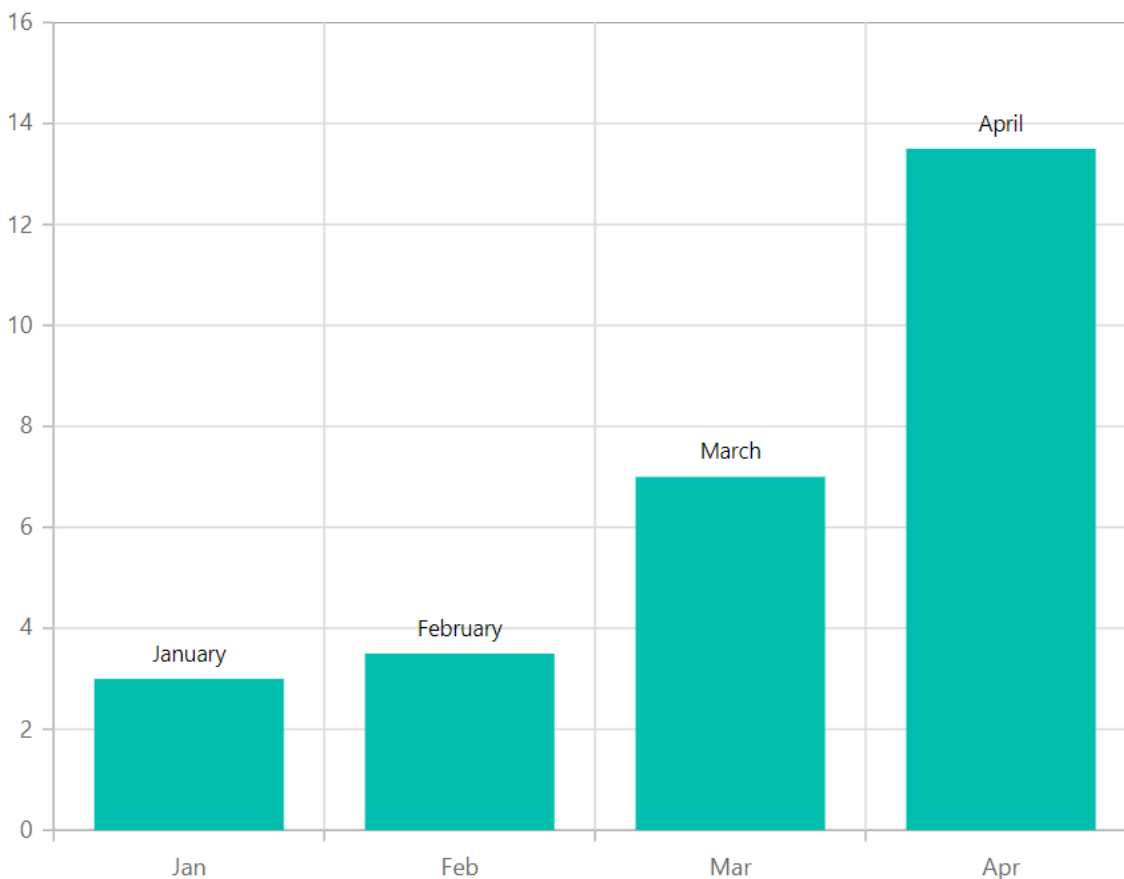
Text Mapping

The [Name](#) property can be used to map text from a datasource to a data label.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart>
  <ChartPrimaryXAxis
    ValueType="Syncfusion.Blazor.Charts.ValueType.Category"></ChartPrimaryXAxis>
  <ChartSeriesCollection>
    <ChartSeries DataSource="@WeatherReports" XName="X" YName="Y"
      Type="ChartSeriesType.Column">
      <ChartMarker>
        <ChartDataLabel Visible="true" Name="Text"></ChartDataLabel>
      </ChartMarker>
    </ChartSeries>
  </ChartSeriesCollection>
</SfChart>
@code{
```

```
public class Data
{
    public string X { get; set; }
    public double Y { get; set; }
    public string Text { get; set; }
}
public List<Data> WeatherReports = new List<Data>
{
    new Data{ X= "Jan", Y= 3, Text= "January" },
    new Data{ X= "Feb", Y= 3.5, Text= "February" },
    new Data{ X= "Mar", Y= 7, Text= "March" },
    new Data{ X= "Apr", Y= 13.5, Text= "April" }
};
```



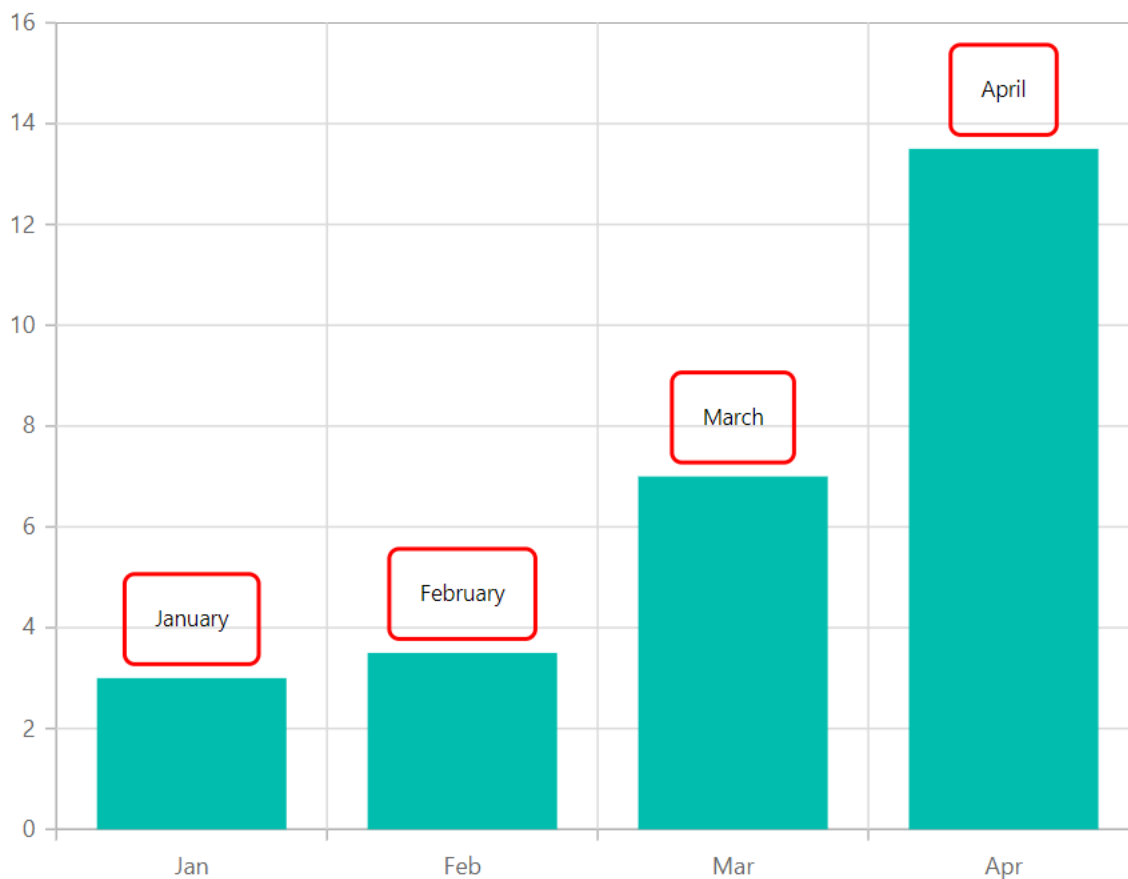
Margin

The [Margin](#) option can be applied to the data label to create space around the element.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart>
<ChartPrimaryXAxis ValueTypes="Syncfusion.Blazor.Charts.ValueType.Category"/>
<ChartSeriesCollection>
```

```
<ChartSeries DataSource="@WeatherReports" XName="X" YName="Y"
Type="ChartSeriesType.Column">
<ChartMarker>
<ChartDataLabel Visible="true" Name="Text">
<ChartDataLabelBorder Width="2" Color="red"/>
<ChartDataLabelMargin Left="15" Right="15" Top="15" Bottom="15"/>
</ChartDataLabel>
</ChartMarker>
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code{
public class Data
{
public string X { get; set; }
public double Y { get; set; }
public string Text { get; set; }
}
public List<Data> WeatherReports = new List<Data>
{
new Data{ X= "Jan", Y= 3, Text= "January" },
new Data{ X= "Feb", Y= 3.5, Text= "February" },
new Data{ X= "Mar", Y= 7, Text= "March" },
new Data{ X= "Apr", Y= 13.5, Text= "April" }
};
}
```



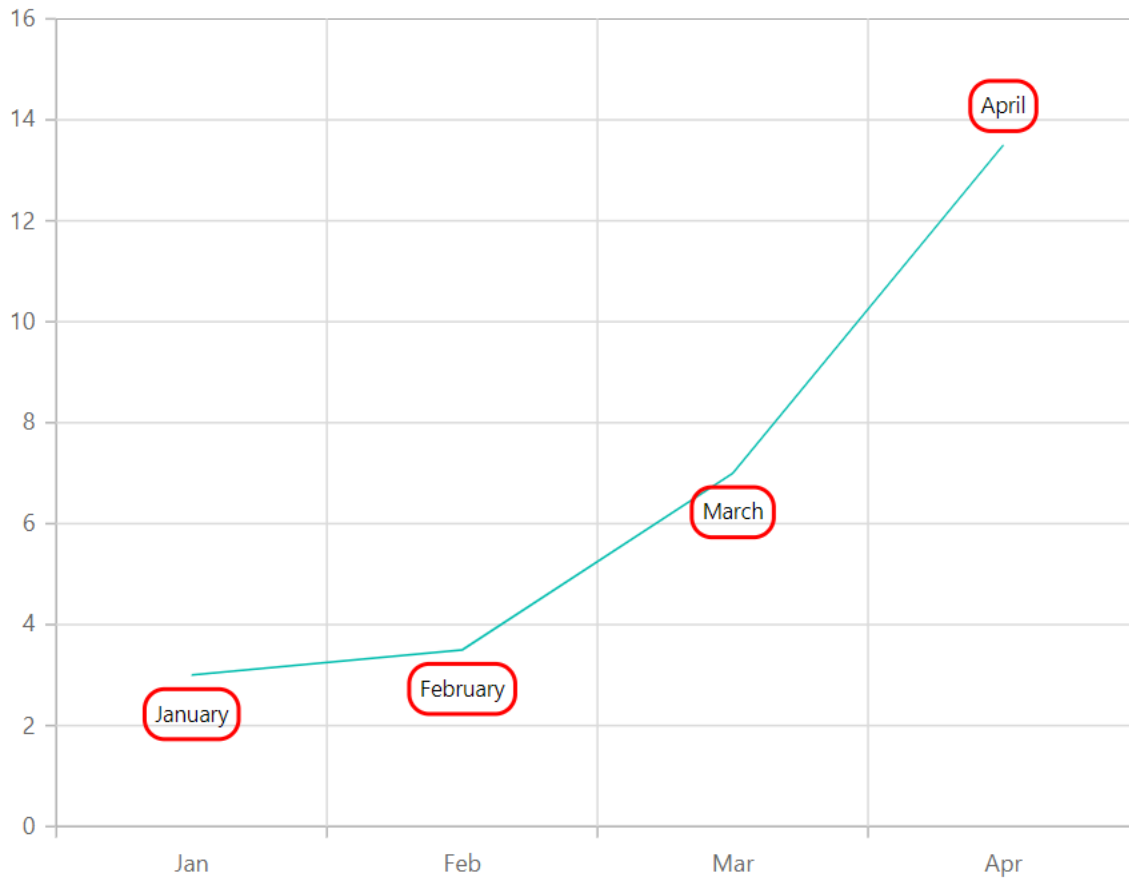
Customization

Data label can be customized using [Fill](#) property and the color and width of data label border can be customized to specified in [ChartDataLabelBorder](#). Rounded corners can also be applied using [Rx](#) and [Ry](#) properties.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart>
<ChartPrimaryXAxis ValueType="Syncfusion.Blazor.Charts.ValueType.Category"/>
<ChartSeriesCollection>
<ChartSeries DataSource="@WeatherReports" XName="X" YName="Y" >
<ChartMarker>
<ChartDataLabel Visible="true" Name="Text" Rx="10" Ry="10">
<ChartDataLabelBorder Width="2" Color="red"></ChartDataLabelBorder>
</ChartDataLabel>
</ChartMarker>
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code{
public class Data
{
public string X { get; set; }
public double Y { get; set; }
}
```

```
public string Text { get; set; }  
}  
public List<Data> WeatherReports = new List<Data>  
{  
    new Data{ X= "Jan", Y= 3, Text= "January" },  
    new Data{ X= "Feb", Y= 3.5, Text= "February" },  
    new Data{ X= "Mar", Y= 7, Text= "March" },  
    new Data{ X= "Apr", Y= 13.5, Text= "April" }  
};  
}
```



Refer to our [Blazor Charts](#) feature tour page for its groundbreaking feature representations and also explore our [Blazor Chart Example](#) to know various chart types and how to represent time-dependent data, showing trends at equal intervals.

See Also

- [Tooltip](#)
- [Legend](#)

<!-- markdownlint-disable MD036 -->

Datalabel Template in Blazor Charts Component

Text and interior information for a point can be bound from a datasource other than the x and y values. The implicit named parameter context can be used to access the aggregate values within the [Template](#). To retrieve aggregate values inside the template, type cast the context as [ChartDataPointInfo](#). The context attribute can also be used to modify the name of this implicit parameter. For example, the data label information can be accessed using context in the template as shown below.

ASPX-CS

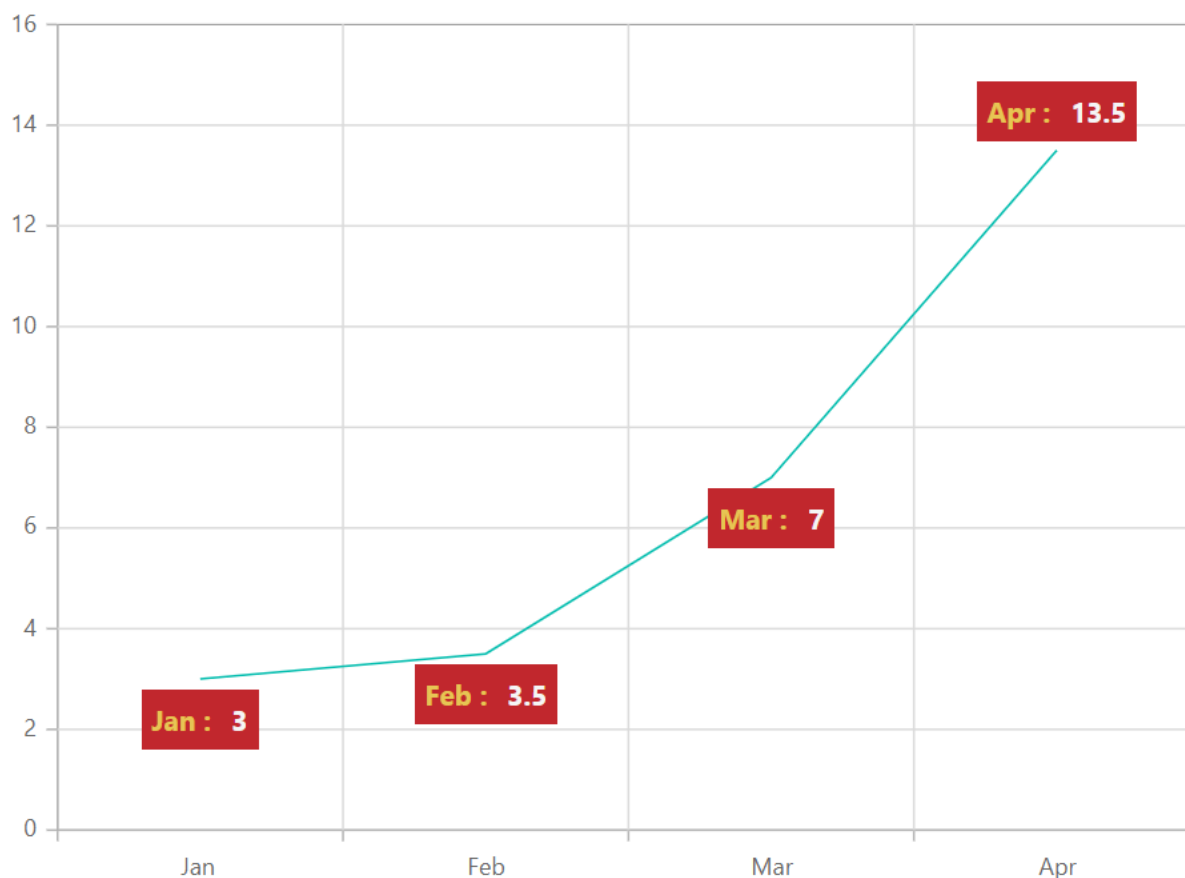
```
<ChartDataLabel Visible="true" Name="Text">
  <Template>
    @{
      var data = context as ChartDataPointInfo;
    }
    <table>
      <tr><td align="center"> @data.Text</td></tr>
    </table>
  </Template>
</ChartDataLabel>
```

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart>
  <ChartPrimaryXAxis ValueType="Syncfusion.Blazor.Charts.ValueType.Category"
  />
  <ChartSeriesCollection>
    <ChartSeries DataSource="@SalesReports" XName="X" YName="Y">
      <ChartMarker>
        <ChartDataLabel Visible="true" Name="Text">
          <Template>
            @{
              var data = context as ChartDataPointInfo;
            }
            <table>
              <tr>
                <td align="center" style="background-color: #C1272D; font-size: 14px; color: #E7C554; font-weight: bold; padding: 5px"> @data.Text :</td>
                <td align="center" style="background-color: #C1272D; font-size: 14px; color: whitesmoke; font-weight: bold; padding: 5px"> @data.Y</td>
              </tr>
            </table>
          </Template>
        </ChartDataLabel>
      </ChartMarker>
    </ChartSeries>
  </ChartSeriesCollection>
</SfChart>
@code{
  public class Data
  {
    public string X { get; set; }
    public double Y { get; set; }
    public string Text { get; set; }
  }
  public List<Data> SalesReports = new List<Data>
```



```
{
  new Data{ X= "Jan", Y= 3, Text= "January" },
  new Data{ X= "Feb", Y= 3.5, Text= "February" },
  new Data{ X= "Mar", Y= 7, Text= "March" },
  new Data{ X= "Apr", Y= 13.5, Text= "April" }
};
}
```



Refer to our [Blazor Charts](#) feature tour page for its groundbreaking feature representations and also explore our [Blazor Chart Example](#) to know various chart types and how to represent time-dependent data, showing trends at equal intervals.

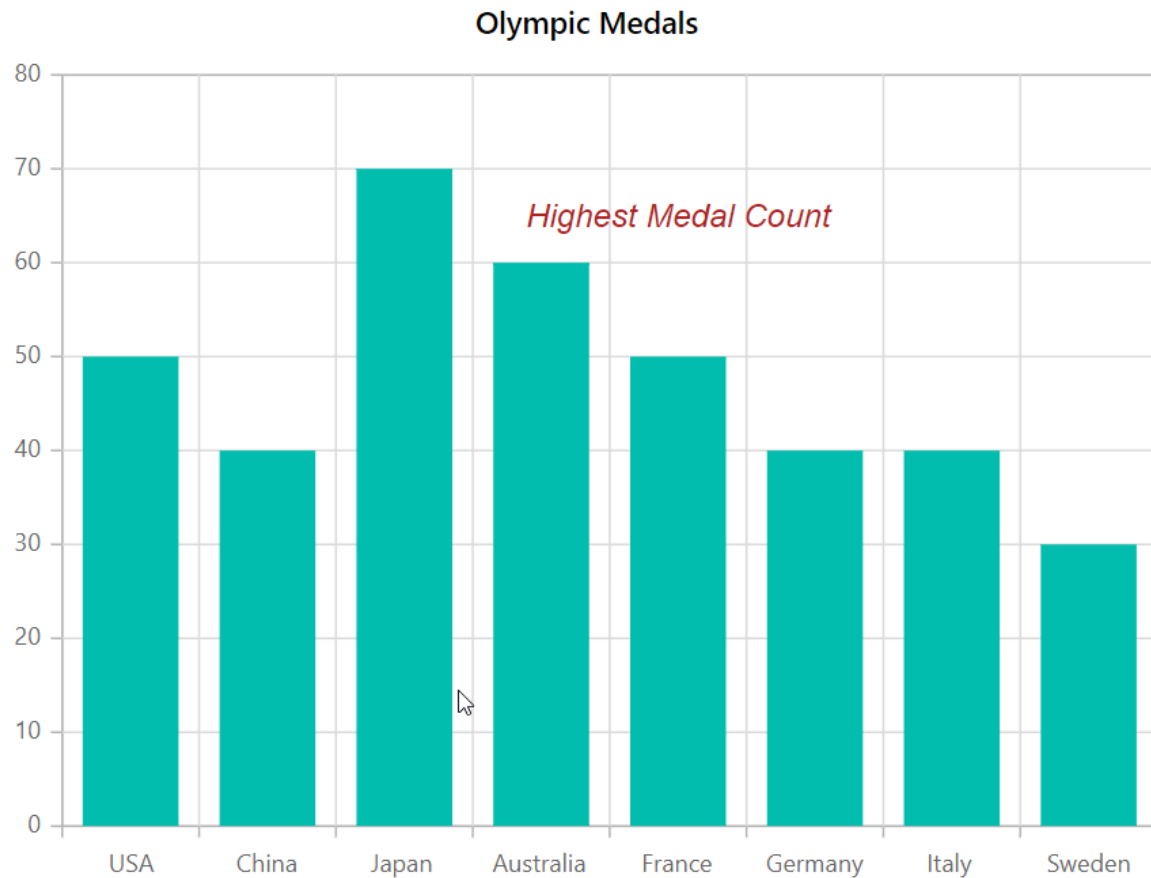
Annotation in Blazor Charts Component

Annotations are texts, shapes, or images that are used to highlight a specific region of interest in a chart. The [ChartAnnotations](#) property allows to add annotations to the chart. Specify the ID of the element that needs to be displayed in the chart area by using the [Content](#) property of the annotation.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart Title="Olympic Medals">
  <ChartPrimaryXAxis Value="Syncfusion.Blazor.Charts.ValueType.Category">
  </ChartPrimaryXAxis>
  <ChartAnnotations>
    <ChartAnnotation X="@data" Y="65" CoordinateUnits="Units.Point">
```

```
<ContentTemplate>
<div style="color: firebrick; font-size: medium; font-style: italic">Highest
Medal Count</div>
</ContentTemplate>
</ChartAnnotation>
</ChartAnnotations>
<ChartSeriesCollection>
<ChartSeries DataSource="@MedalDetails" XName="Country" YName="Gold"
Type="ChartSeriesType.Column">
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code{
string data = "France";
public class ChartData
{
public string Country { get; set; }
public double Gold { get; set; }
}
public List<ChartData> MedalDetails = new List<ChartData>
{
new ChartData{ Country= "USA", Gold=50 },
new ChartData{ Country= "China", Gold=40 },
new ChartData{ Country= "Japan", Gold=70 },
new ChartData{ Country= "Australia", Gold=60},
new ChartData{ Country= "France", Gold=50 },
new ChartData{ Country= "Germany", Gold=40 },
new ChartData{ Country= "Italy", Gold=40 },
new ChartData{ Country= "Sweden", Gold=30 }
};
}
```



Region

The [Region](#) property can be used to insert annotations in relation to a series or a chart. By default, it is positioned with respect to a [Chart](#).

ASPX-CS

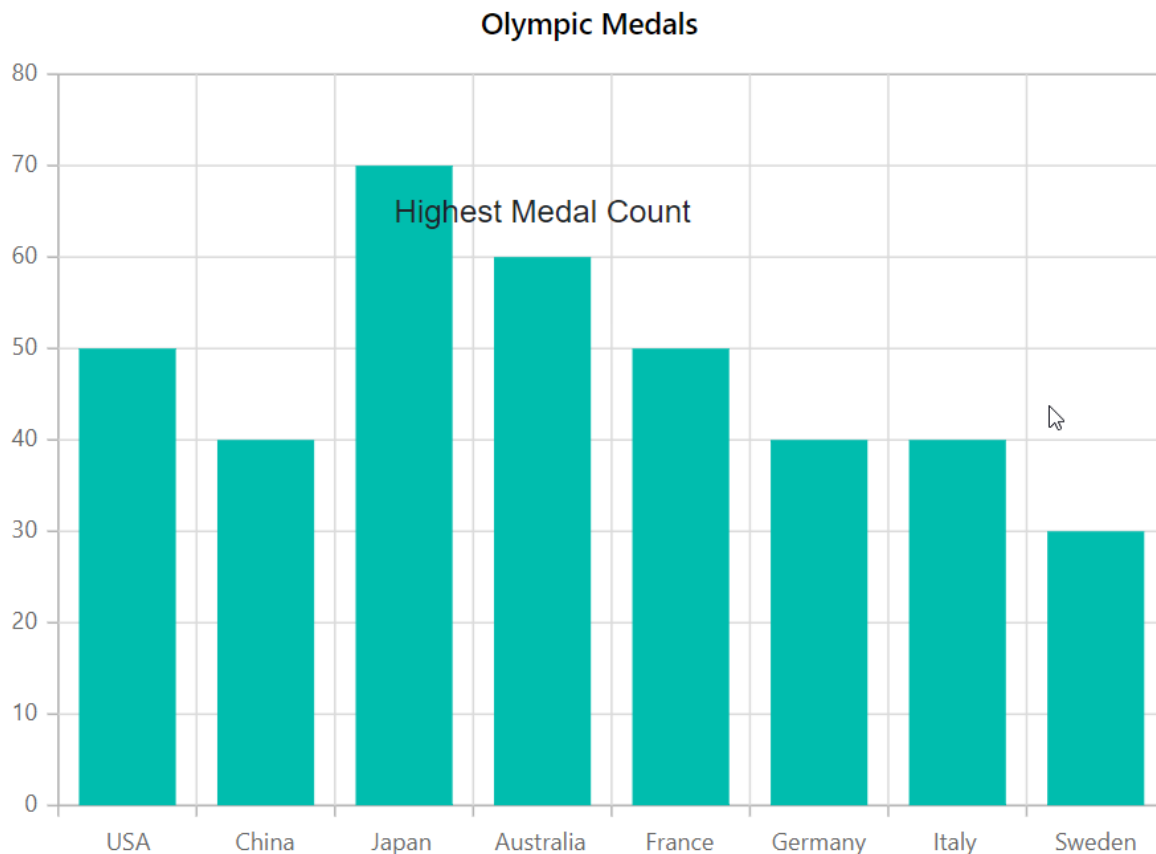
```
@using Syncfusion.Blazor.Charts
<SfChart Title="Olympic Medals">
  <ChartPrimaryXAxis Value="Syncfusion.Blazor.Charts.ValueType.Category">
  </ChartPrimaryXAxis>
  <ChartAnnotations>
    <ChartAnnotation X="@Country" Y="65" Region="Regions.Series"
      CoordinateUnits="Units.Point">
      <ContentTemplate>
        <div>Highest Medal Count</div>
      </ContentTemplate>
    </ChartAnnotation>
  </ChartAnnotations>
  <ChartSeriesCollection>
    <ChartSeries DataSource="@MedalDetails" XName="Country" YName="Gold"
      Type="ChartSeriesType.Column">
    </ChartSeries>
  </ChartSeriesCollection>
</SfChart>

@code{
  string Country = "Australia";
```

```

public class ChartData
{
    public string Country { get; set; }
    public double Gold { get; set; }
}
public List<ChartData> MedalDetails = new List<ChartData>
{
    new ChartData{ Country= "USA", Gold=50 },
    new ChartData{ Country= "China", Gold=40 },
    new ChartData{ Country= "Japan", Gold=70 },
    new ChartData{ Country= "Australia", Gold=60},
    new ChartData{ Country= "France", Gold=50 },
    new ChartData{ Country= "Germany", Gold=40 },
    new ChartData{ Country= "Italy", Gold=40 },
    new ChartData{ Country= "Sweden", Gold=30 }
};
}

```



Co-ordinate Units

The [CoordinateUnits](#) property allows to specify the annotation's coordinate units either in [Pixel](#) or [Point](#).

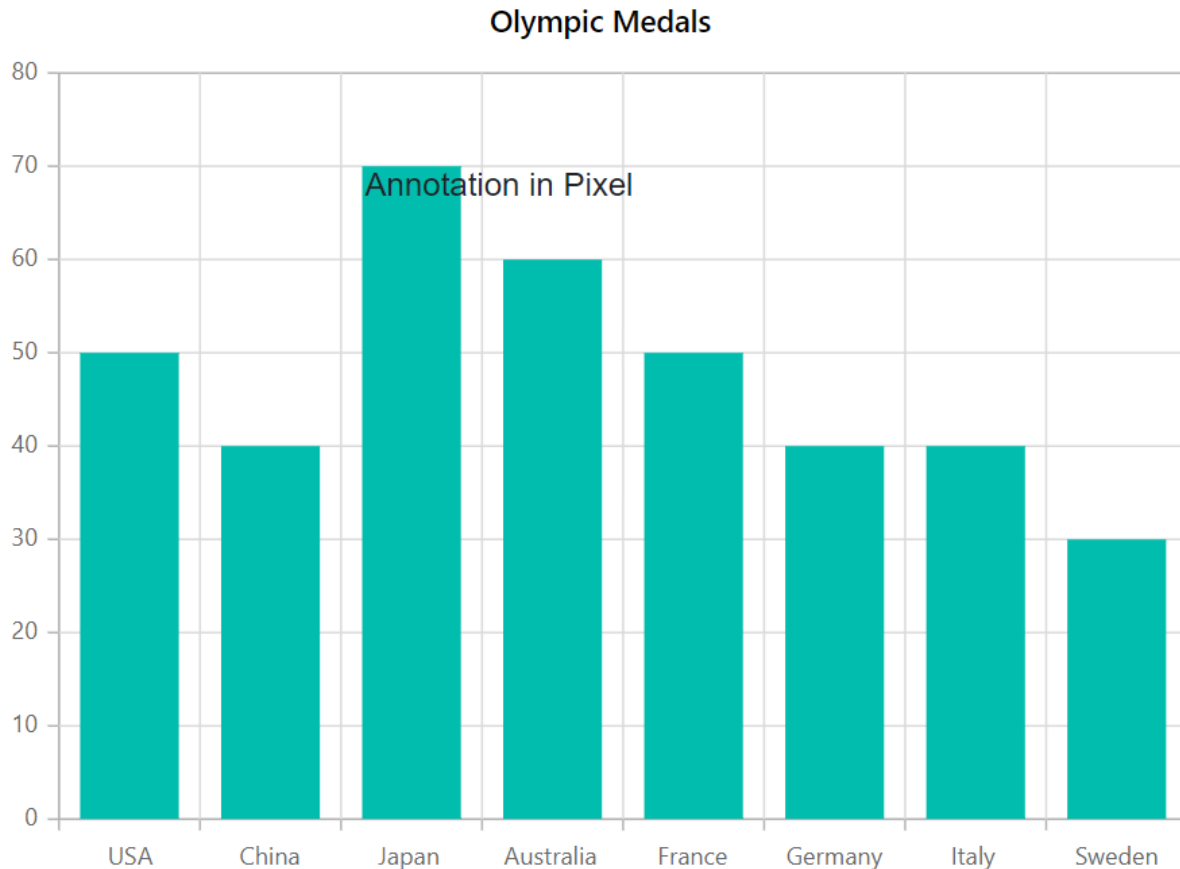
ASPX-CS

```

@using Syncfusion.Blazor.Charts
<SfChart Title="Olympic Medals">
  <ChartPrimaryXAxis ValueType="Syncfusion.Blazor.Charts.ValueType.Category">
  </ChartPrimaryXAxis>

```

```
<ChartAnnotations>
<ChartAnnotation X="250" Y="100" CoordinateUnits="Units.Pixel">
<ContentTemplate>
<div>Annotation in Pixel</div>
</ContentTemplate>
</ChartAnnotation>
</ChartAnnotations>
<ChartSeriesCollection>
<ChartSeries DataSource="@MedalDetails" XName="Country" YName="Gold"
Type="ChartSeriesType.Column">
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code{
public class ChartData
{
public string Country { get; set; }
public double Gold { get; set; }
}
public List<ChartData> MedalDetails = new List<ChartData>
{
new ChartData{ Country= "USA", Gold=50 },
new ChartData{ Country= "China", Gold=40 },
new ChartData{ Country= "Japan", Gold=70 },
new ChartData{ Country= "Australia", Gold=60},
new ChartData{ Country= "France", Gold=50 },
new ChartData{ Country= "Germany", Gold=40 },
new ChartData{ Country= "Italy", Gold=40 },
new ChartData{ Country= "Sweden", Gold=30 }
};
}
```



Refer to our [Blazor Charts](#) feature tour page for its groundbreaking feature representations and also explore our [Blazor Chart Example](#) to know various chart types and how to represent time-dependent data, showing trends at equal intervals.

See Also

- [Tooltip](#)
- [Legend](#)

Appearance in Blazor Charts Component

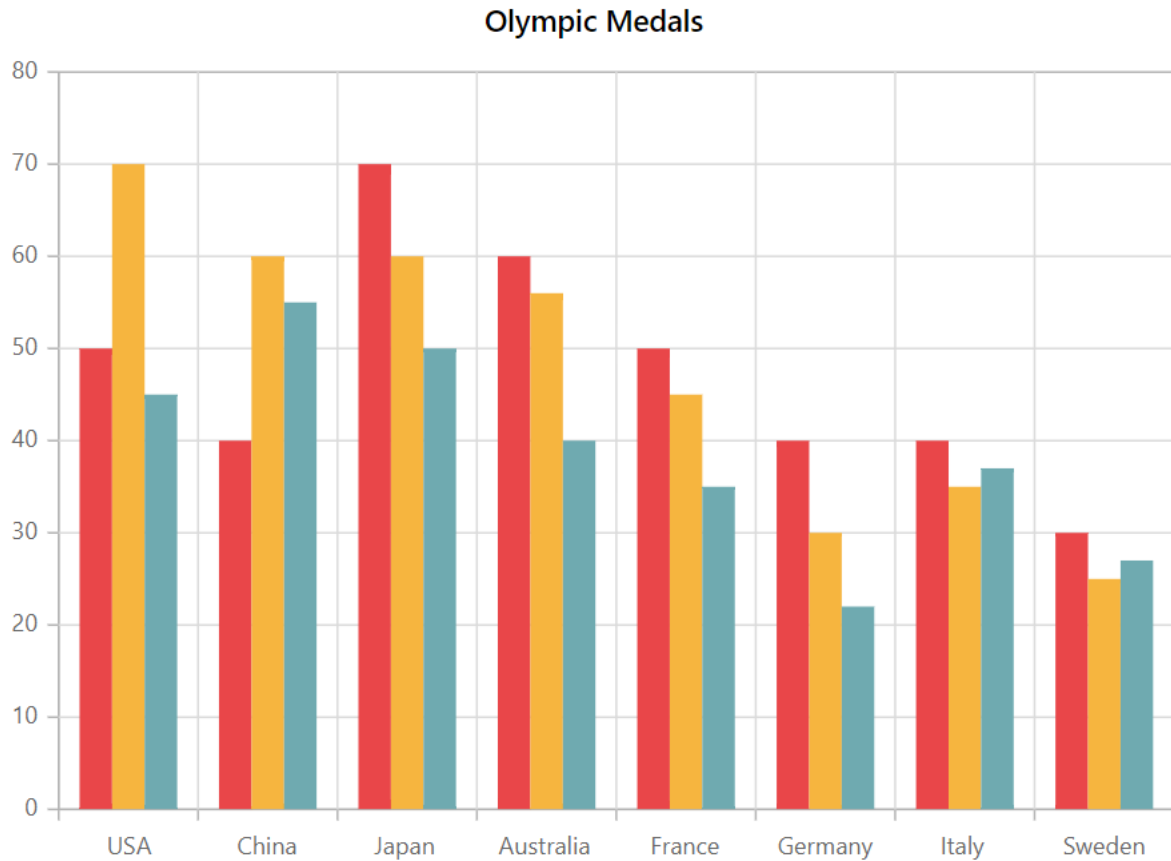
Custom Color Palette

The default color of series or points can be changed by providing a custom color palette to the [Palettes](#) property.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart Title="Olympic Medals" Palettes="@palettes">
  <ChartPrimaryXAxis ValueType="Syncfusion.Blazor.Charts.ValueType.Category">
  </ChartPrimaryXAxis>
  <ChartSeriesCollection>
    <ChartSeries DataSource="@MedalDetails" XName="Country" YName="Gold"
      Type="ChartSeriesType.Column">
    </ChartSeries>
```

```
<ChartSeries DataSource="@MedalDetails" XName="Country" YName="Silver"
Type="ChartSeriesType.Column">
</ChartSeries>
<ChartSeries DataSource="@MedalDetails" XName="Country" YName="Bronze"
Type="ChartSeriesType.Column">
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code{
public class ChartData
{
public string Country { get; set; }
public double Gold { get; set; }
public double Silver { get; set; }
public double Bronze { get; set; }
}
public List<ChartData> MedalDetails = new List<ChartData>
{
new ChartData{ Country= "USA", Gold=50, Silver=70, Bronze=45 },
new ChartData{ Country="China", Gold=40, Silver= 60, Bronze=55 },
new ChartData{ Country= "Japan", Gold=70, Silver= 60, Bronze=50 },
new ChartData{ Country= "Australia", Gold=60, Silver= 56, Bronze=40 },
new ChartData{ Country= "France", Gold=50, Silver= 45, Bronze=35 },
new ChartData{ Country= "Germany", Gold=40, Silver=30, Bronze=22 },
new ChartData{ Country= "Italy", Gold=40, Silver=35, Bronze=37 },
new ChartData{ Country= "Sweden", Gold=30, Silver=25, Bronze=27 }
};
public String[] palettes = new String[] { "#E94649", "#F6B53F", "#6FAAB0" };
}
```



<!-- markdownlint-disable MD036 -->

Chart Customization

<!-- markdownlint-disable MD036 -->

Chart Background and Border

<!-- markdownlint-disable MD013 -->

The chart's background color can be customized using the [Background](#) property and the border color and width can be customized to specified in [ChartBorder](#).

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart Title="Olympic Medals" Background="skyblue">
  <ChartBorder Color="#FF0000" Width="2"></ChartBorder>
  <ChartPrimaryXAxis ValueType="Syncfusion.Blazor.Charts.ValueType.Category"
  />
  <ChartSeriesCollection>
    <ChartSeries DataSource="@MedalDetails" XName="Country" YName="Gold"
    Type="ChartSeriesType.Column">
  </ChartSeries>
  </ChartSeriesCollection>
</SfChart>
@code{
public class ChartData
{

```



```

public string Country { get; set; }
public double Gold { get; set; }
}
public List<ChartData> MedalDetails = new List<ChartData>
{
    new ChartData{ Country= "USA", Gold=50 },
    new ChartData{ Country="China", Gold=40 },
    new ChartData{ Country= "Japan", Gold=70 },
    new ChartData{ Country= "Australia", Gold=60},
    new ChartData{ Country= "France", Gold=50 },
    new ChartData{ Country= "Germany", Gold=40 },
    new ChartData{ Country= "Italy", Gold=40 },
    new ChartData{ Country= "Sweden", Gold=30 }
};
}

```

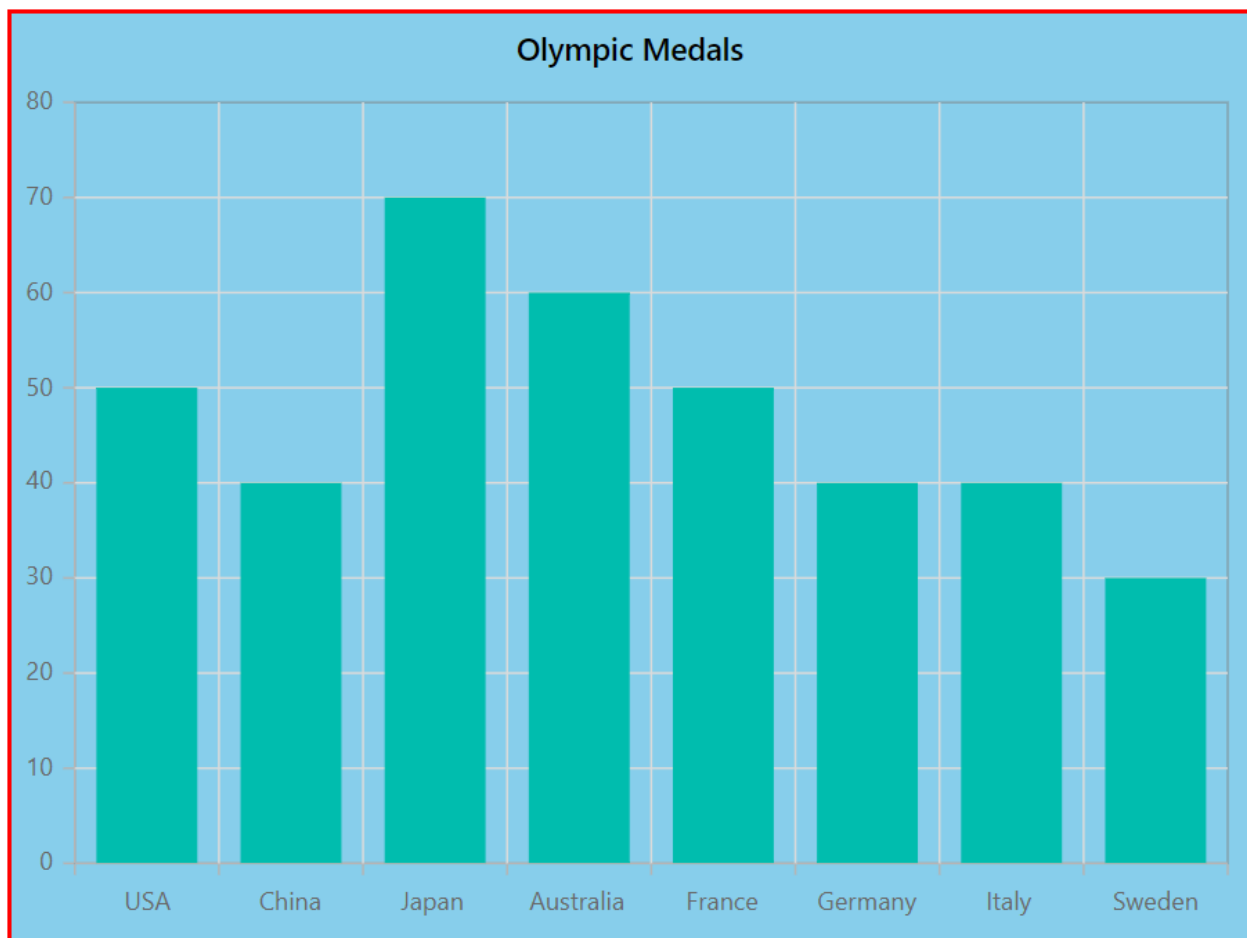


Chart Margin

The chart's margin from its container can be customized using the [ChartMargin](#).

ASPX-CS

```

@using Syncfusion.Blazor.Charts
<SfChart Title="Olympic Medals" Background="skyblue">
  <ChartBorder Color="#FF0000" Width="2"></ChartBorder>
  <ChartMargin Left="60" Right="60" Top="60" Bottom="60"></ChartMargin>

```

```
<ChartPrimaryXAxis ValueType="Syncfusion.Blazor.Charts.ValueType.Category">
</ChartPrimaryXAxis>
<ChartSeriesCollection>
<ChartSeries DataSource="@MedalDetails" XName="Country" YName="Gold"
Type="ChartSeriesType.Column">
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code{
public class ChartData
{
public string Country { get; set; }
public double Gold { get; set; }
}
public List<ChartData> MedalDetails = new List<ChartData>
{
new ChartData{ Country= "USA", Gold=50 },
new ChartData{ Country="China", Gold=40 },
new ChartData{ Country= "Japan", Gold=70 },
new ChartData{ Country= "Australia", Gold=60},
new ChartData{ Country= "France", Gold=50 },
new ChartData{ Country= "Germany", Gold=40 },
new ChartData{ Country= "Italy", Gold=40 },
new ChartData{ Country= "Sweden", Gold=30 }
};
}
```

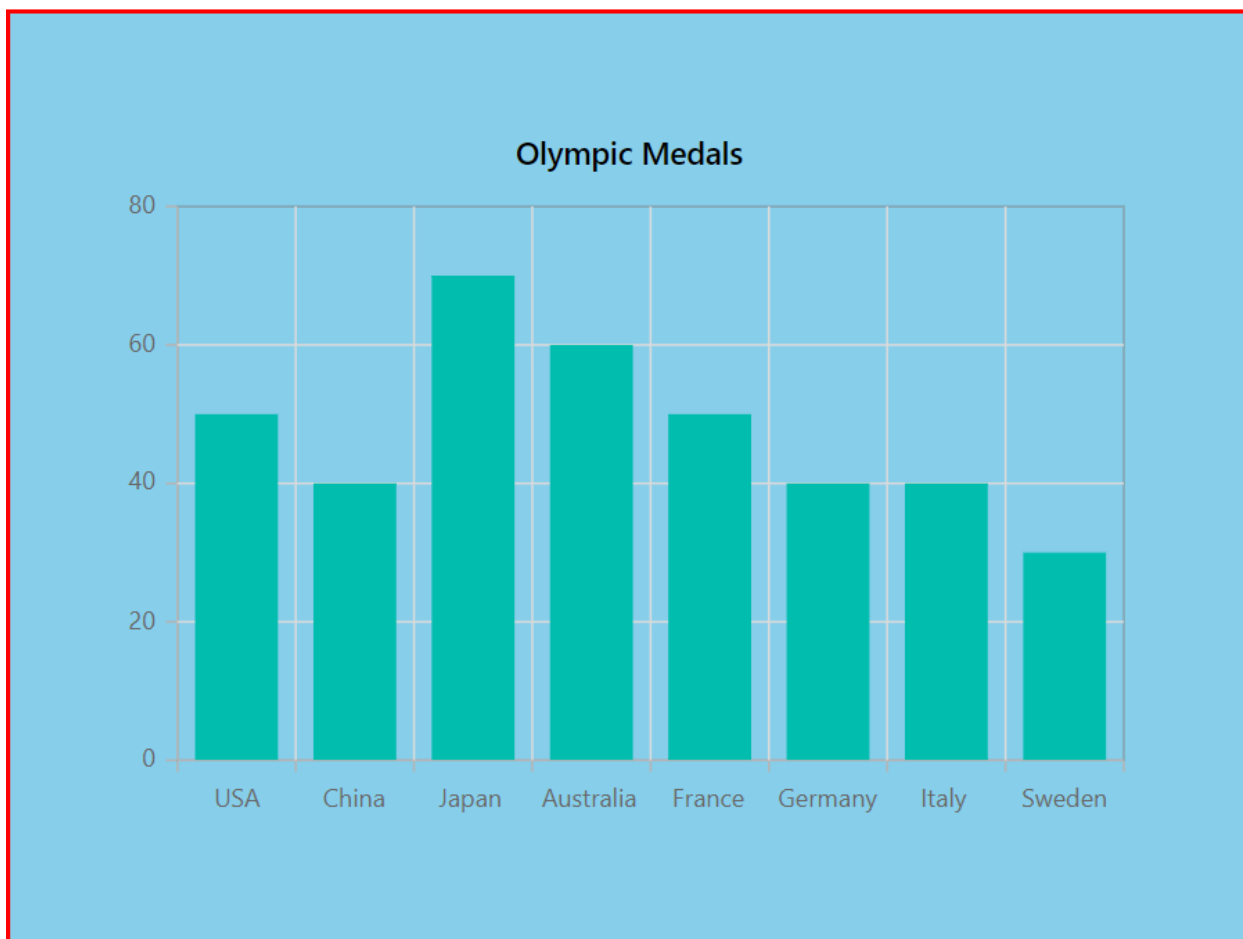


Chart Area Background and Border

The chart area's background color can be customized using the [Background](#) property and the border color and width can be customized to specified in [ChartAreaBorder](#) of [ChartArea](#).

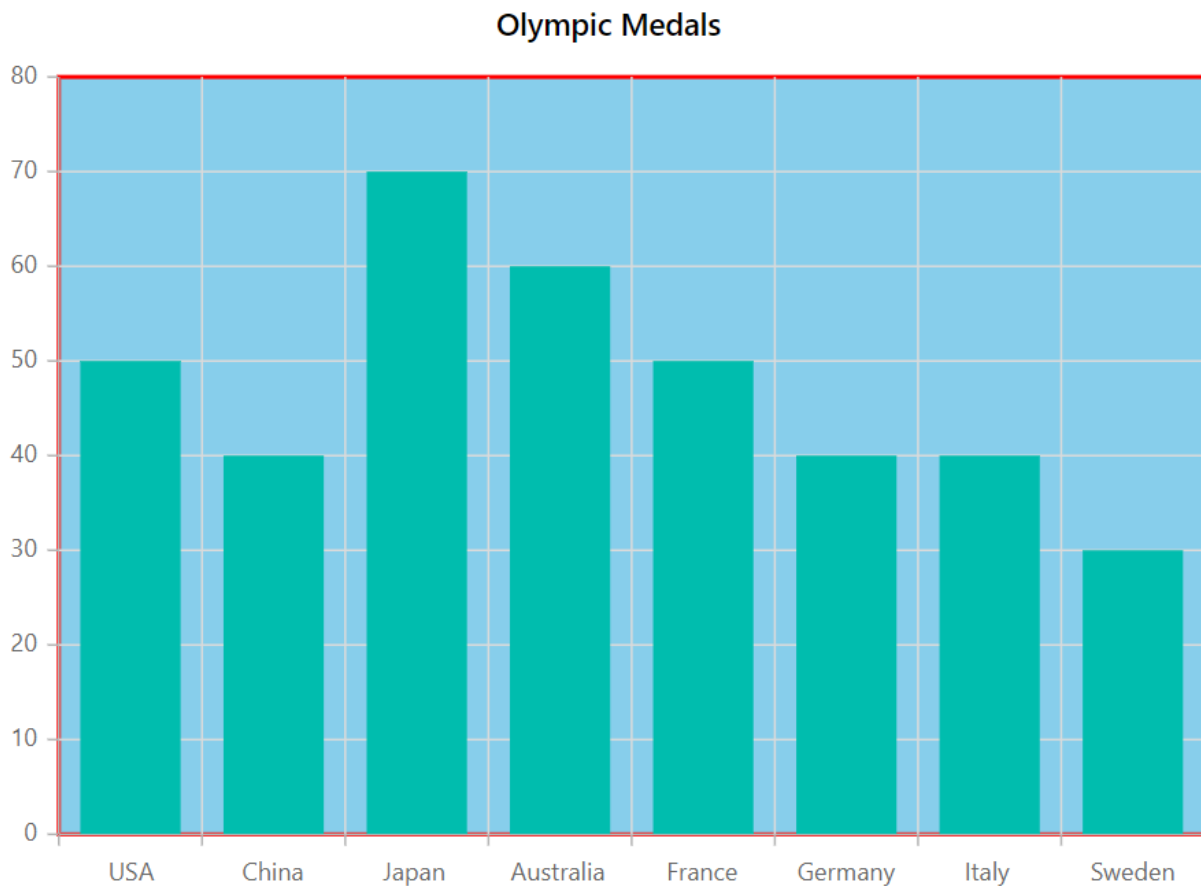
ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart Title="Olympic Medals" >
  <ChartArea Background="skyblue">
    <ChartAreaBorder Color="#FF0000" Width="2" />
  </ChartArea>
  <ChartPrimaryXAxis ValueType="Syncfusion.Blazor.Charts.ValueType.Category" />
  <ChartSeriesCollection>
    <ChartSeries DataSource="@MedalDetails" XName="Country" YName="Gold"
      Type="ChartSeriesType.Column" />
  </ChartSeriesCollection>
</SfChart>

@code{
public class ChartData
{
    public string Country { get; set; }
    public double Gold { get; set; }
}

public List<ChartData> MedalDetails = new List<ChartData>
```

```
{  
new ChartData{ Country= "USA", Gold=50 },  
new ChartData{ Country="China", Gold=40 },  
new ChartData{ Country= "Japan", Gold=70 },  
new ChartData{ Country= "Australia", Gold=60},  
new ChartData{ Country= "France", Gold=50 },  
new ChartData{ Country= "Germany", Gold=40 },  
new ChartData{ Country= "Italy", Gold=40 },  
new ChartData{ Country= "Sweden", Gold=30 }  
};  
}
```



Animation

The [Animation](#) property allows to customize animation for a certain series. The [Enable](#) property can be used to enable or disable the series animation. The duration of the animation is specified by [Duration](#) property, and [Delay](#) property allows to start the animation at a specific moment.

ASPX-CS

```
@using Syncfusion.Blazor.Charts  
<SfChart Title="Olympic Medals">  
<ChartPrimaryXAxis ValueType="Syncfusion.Blazor.Charts.ValueType.Category"/>  
<ChartSeriesCollection>  
<ChartSeries DataSource="@MedalDetails" Name="Gold" XName="Country"  
Width="2" Opacity="1" YName="Gold" Type="ChartSeriesType.Column">
```

```

<ChartSeriesAnimation Enable="true" Duration="2000"
Delay="200"></ChartSeriesAnimation>
<ChartSeriesBorder Width="3" Color="red"></ChartSeriesBorder>
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code{
public class ChartData
{
public string Country { get; set; }
public double Gold { get; set; }
public double Silver { get; set; }
public double Bronze { get; set; }
}
public List<ChartData> MedalDetails = new List<ChartData>
{
new ChartData{ Country= "USA", Gold=50, Silver=70, Bronze=45 },
new ChartData{ Country="China", Gold=40, Silver= 60, Bronze=55 },
new ChartData{ Country= "Japan", Gold=70, Silver= 60, Bronze=50 },
new ChartData{ Country= "Australia", Gold=60, Silver= 56, Bronze=40 },
new ChartData{ Country= "France", Gold=50, Silver= 45, Bronze=35 },
new ChartData{ Country= "Germany", Gold=40, Silver=30, Bronze=22 },
new ChartData{ Country= "Italy", Gold=40, Silver=35, Bronze=37 },
new ChartData{ Country= "Sweden", Gold=30, Silver=25, Bronze=27 }
};
}

```

Chart Title

The [Title](#) property can be used to give the chart a title in-order to provide information about the data displayed.

ASPX-CS

```

@using Syncfusion.Blazor.Charts
<SfChart Title="Olympic Medals">
<ChartTitleStyle Size="23px" Color="red" FontFamily="Arial"
FontWeight="regular" FontStyle="italic"></ChartTitleStyle>
<ChartPrimaryXAxis ValueTypes="Syncfusion.Blazor.Charts.ValueType.Category">
</ChartPrimaryXAxis>
<ChartSeriesCollection>
<ChartSeries DataSource="@MedalDetails" XName="Country" YName="Gold"
Type="ChartSeriesType.Column">
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code{
public class ChartData
{
public string Country { get; set; }
public double Gold {get; set; }
}
public List<ChartData> MedalDetails = new List<ChartData>
{
new ChartData{ Country= "USA", Gold=50 },
new ChartData{ Country="China", Gold=40 },
new ChartData{ Country= "Japan", Gold=70 },

```

```
new ChartData{ Country= "Australia", Gold=60},
new ChartData{ Country= "France", Gold=50 },
new ChartData{ Country= "Germany", Gold=40 },
new ChartData{ Country= "Italy", Gold=40 },
new ChartData{ Country= "Sweden", Gold=30 }
};
}
```

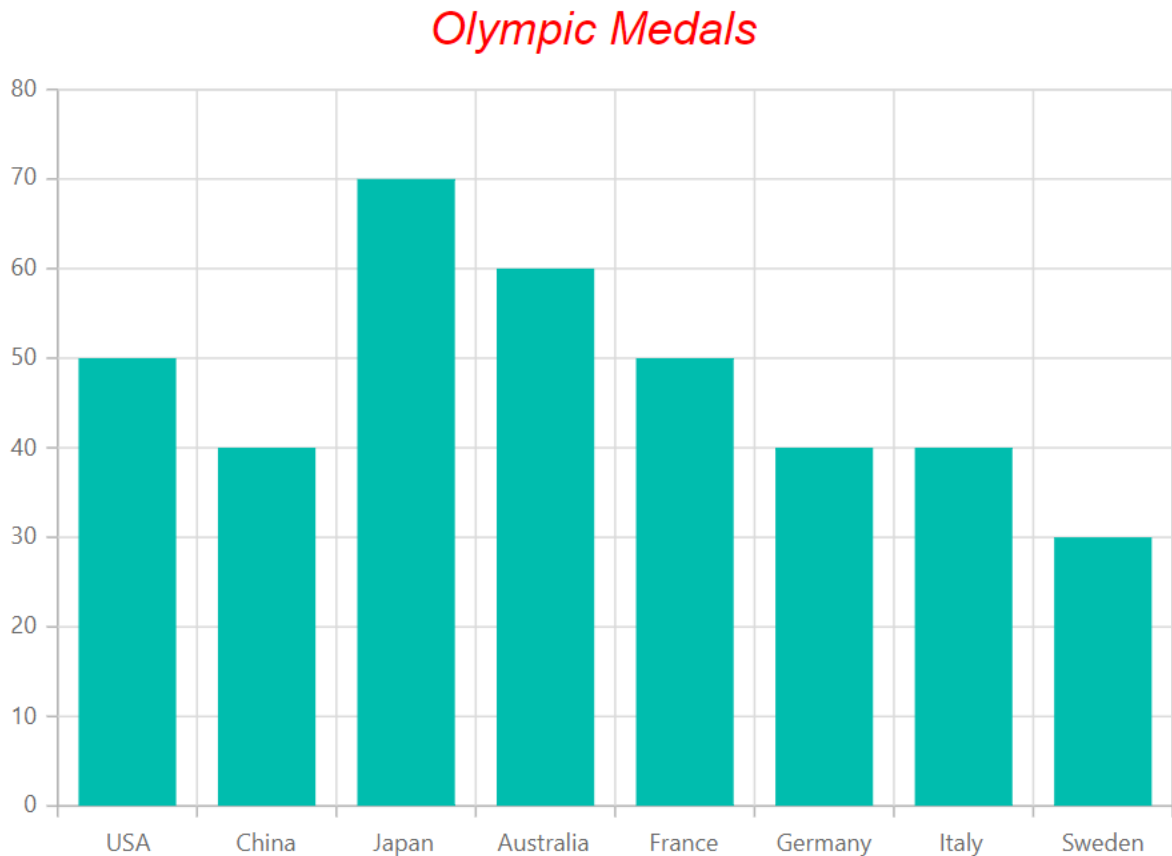


Chart Subtitle

The [SubTitle](#) property can be used to give the chart a subtitle in-order to provide additional information about the data displayed.

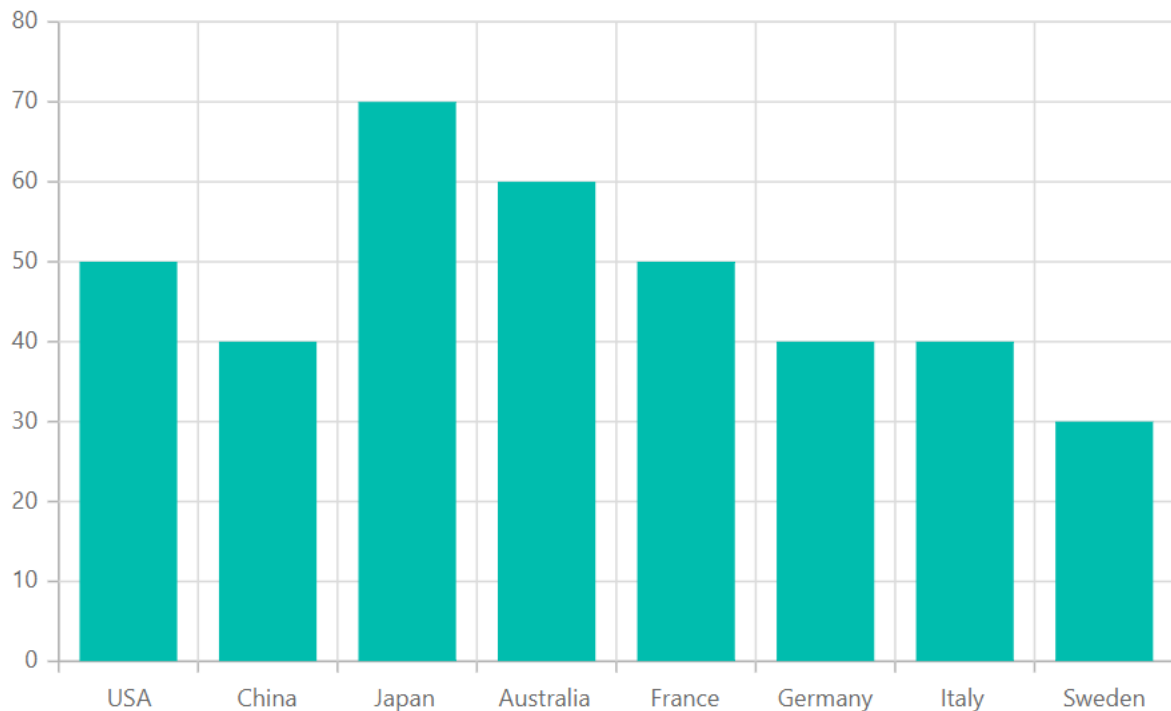
ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart Title="Olympic Medals" SubTitle="Medals">
  <ChartSubTitleStyle FontFamily="Arial" FontStyle="italic"
    FontWeight="regular" Size="18px" Color="red"></ChartSubTitleStyle>
  <ChartTitleStyle FontFamily="Arial" FontStyle="italic" FontWeight="regular"
    Size="23px" Color="red"></ChartTitleStyle>
  <ChartPrimaryXAxis ValueType="Syncfusion.Blazor.Charts.ValueType.Category">
  </ChartPrimaryXAxis>
  <ChartSeriesCollection>
  <ChartSeries DataSource="@MedalDetails" XName="Country" YName="Gold"
    Type="ChartSeriesType.Column">
  </ChartSeries>
```

```
</ChartSeriesCollection>
</SfChart>
@code{
public class ChartData
{
public string Country { get; set; }
public double Gold { get; set; }
}
public List<ChartData> MedalDetails = new List<ChartData>
{
new ChartData{ Country= "USA", Gold=50 },
new ChartData{ Country="China", Gold=40 },
new ChartData{ Country= "Japan", Gold=70 },
new ChartData{ Country= "Australia", Gold=60},
new ChartData{ Country= "France", Gold=50 },
new ChartData{ Country= "Germany", Gold=40 },
new ChartData{ Country= "Italy", Gold=40 },
new ChartData{ Country= "Sweden", Gold=30 }
};
}
```

Olympic Medals

Medals



Refer to our [Blazor Charts](#) feature tour page for its groundbreaking feature representations and also explore our [Blazor Chart Example](#) to know various chart types and how to represent time-dependent data, showing trends at equal intervals.

See Also

- [Data Label](#)
- [Tooltip](#)

Legend in Blazor Charts Component

The [legend](#) provides information on the series shown in the chart.

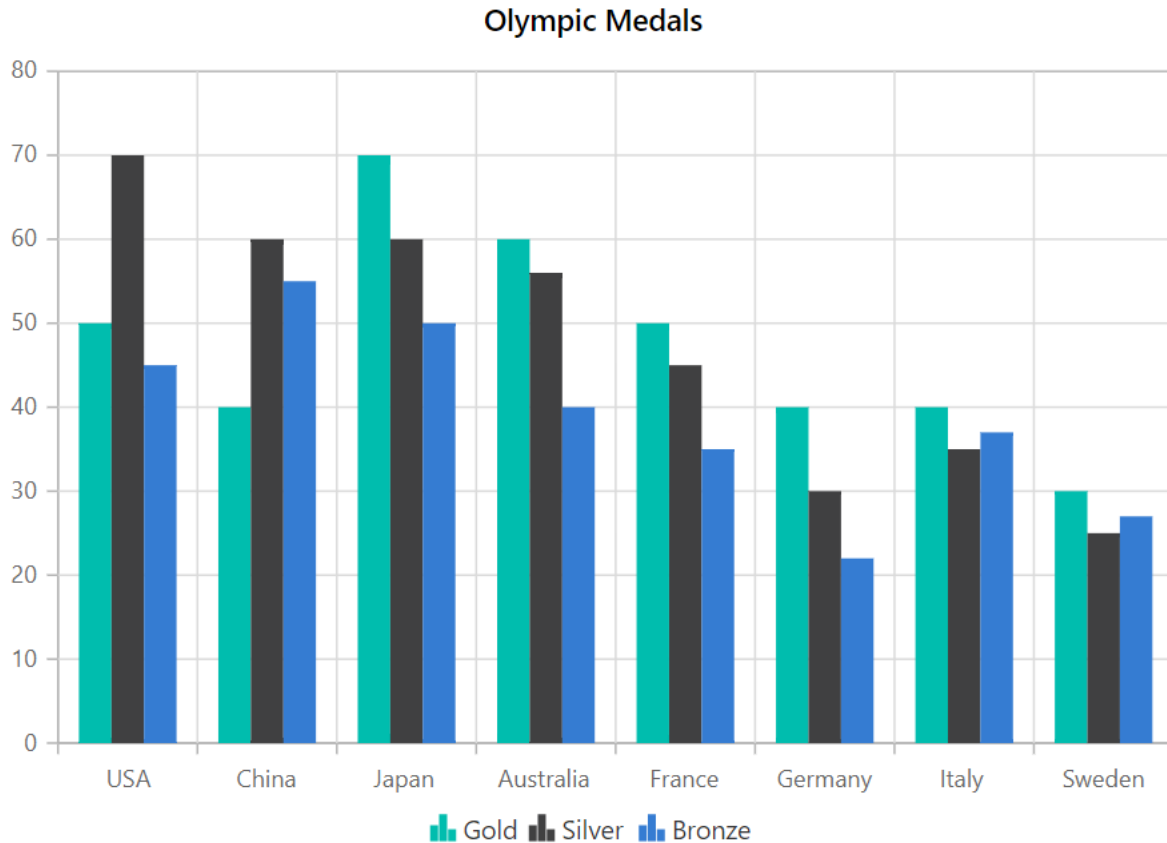
Enable Legend

To display the legend for the chart, set the [Visible](#) property in [ChartLegendSettings](#) to **true**.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart Title="Olympic Medals">
  <ChartPrimaryXAxis ValueType="Syncfusion.Blazor.Charts.ValueType.Category"/>
  <ChartSeriesCollection>
    <ChartSeries DataSource="@MedalDetails" Name="Gold" XName="Country"
      Width="2" Opacity="1" YName="Gold" Type="ChartSeriesType.Column"/>
    <ChartSeries DataSource="@MedalDetails" Name="Silver" XName="Country"
      Width="2" Opacity="1" YName="Silver" Type="ChartSeriesType.Column"/>
    <ChartSeries DataSource="@MedalDetails" Name="Bronze" XName="Country"
      Width="2" Opacity="1" YName="Bronze" Type="ChartSeriesType.Column"/>
  </ChartSeriesCollection>
  <ChartLegendSettings Visible="true"/>
</SfChart>

@code{
  public class ChartData
  {
    public string Country { get; set; }
    public double Gold { get; set; }
    public double Silver { get; set; }
    public double Bronze { get; set; }
  }
  public List<ChartData> MedalDetails = new List<ChartData>
  {
    new ChartData{ Country= "USA", Gold=50, Silver=70, Bronze=45 },
    new ChartData{ Country="China", Gold=40, Silver= 60, Bronze=55 },
    new ChartData{ Country= "Japan", Gold=70, Silver= 60, Bronze=50 },
    new ChartData{ Country= "Australia", Gold=60, Silver= 56, Bronze=40 },
    new ChartData{ Country= "France", Gold=50, Silver= 45, Bronze=35 },
    new ChartData{ Country= "Germany", Gold=40, Silver=30, Bronze=22 },
    new ChartData{ Country= "Italy", Gold=40, Silver=35, Bronze=37 },
    new ChartData{ Country= "Sweden", Gold=30, Silver=25, Bronze=27 }
  };
}
```

Position and Alignment

<!-- markdownlint-disable MD036 -->

Legend Position

<!-- markdownlint-disable MD036 -->

The legend can be placed at [Left](#), [Right](#), [Top](#), [Bottom](#) or [Custom](#) position of the chart using the [Position](#) property. By default, the legend appears at the bottom of the chart.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart Title="Olympic Medals">
  <ChartPrimaryXAxis Value="Syncfusion.Blazor.Charts.ValueType.Category"/>
  <ChartSeriesCollection>
    <ChartSeries DataSource="@MedalDetails" Name="Gold" XName="Country"
      Width="2" Opacity="1" YName="Gold" Type="ChartSeriesType.Column"/>
    <ChartSeries DataSource="@MedalDetails" Name="Silver" XName="Country"
      Width="2" Opacity="1" YName="Silver" Type="ChartSeriesType.Column"/>
    <ChartSeries DataSource="@MedalDetails" Name="Bronze" XName="Country"
      Width="2" Opacity="1" YName="Bronze" Type="ChartSeriesType.Column"/>
  </ChartSeriesCollection>
  <ChartLegendSettings Visible="true" Position="LegendPosition.Top"/>
</SfChart>

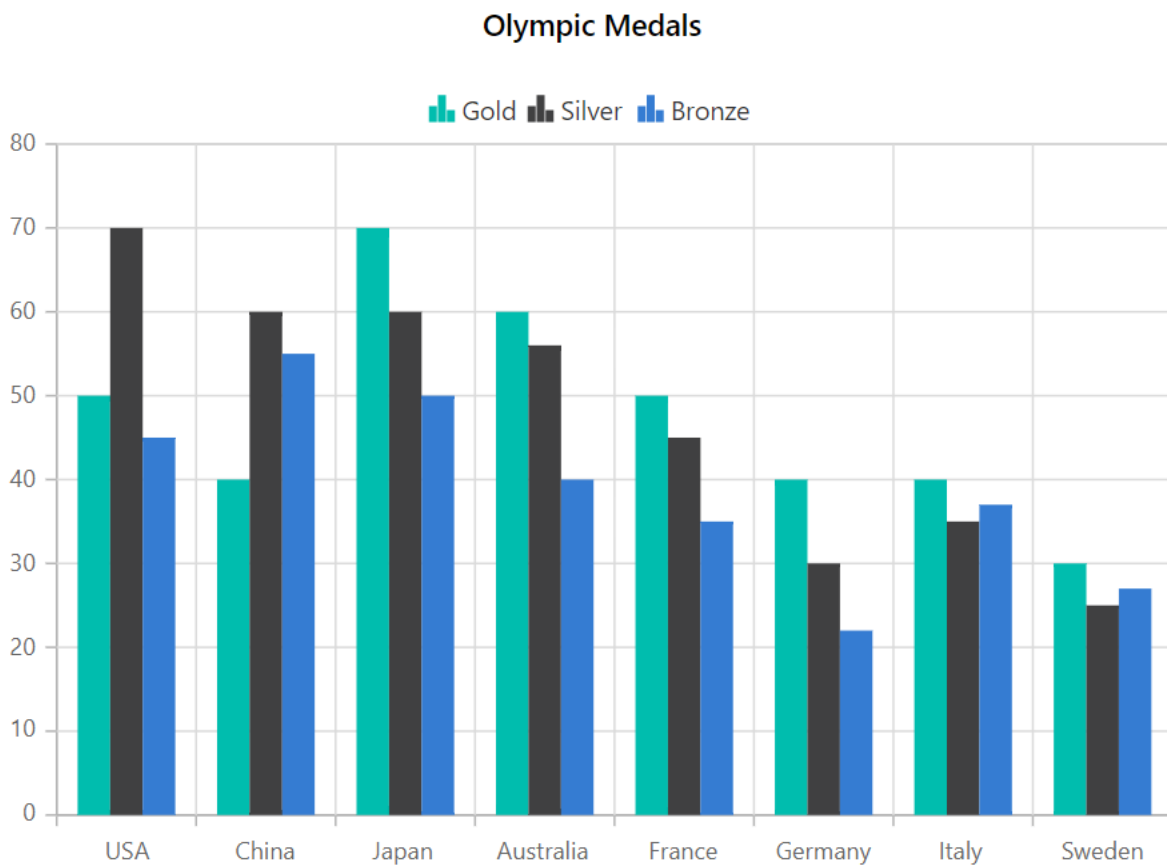
@code{
public class ChartData
{

```

```

public string Country { get; set; }
public double Gold { get; set; }
public double Silver { get; set; }
public double Bronze { get; set; }
}
public List<ChartData> MedalDetails = new List<ChartData>
{
    new ChartData{ Country= "USA", Gold=50, Silver=70, Bronze=45 },
    new ChartData{ Country="China", Gold=40, Silver= 60, Bronze=55 },
    new ChartData{ Country= "Japan", Gold=70, Silver= 60, Bronze=50 },
    new ChartData{ Country= "Australia", Gold=60, Silver= 56, Bronze=40 },
    new ChartData{ Country= "France", Gold=50, Silver= 45, Bronze=35 },
    new ChartData{ Country= "Germany", Gold=40, Silver=30, Bronze=22 },
    new ChartData{ Country= "Italy", Gold=40, Silver=35, Bronze=37 },
    new ChartData{ Country= "Sweden", Gold=30, Silver=25, Bronze=27 }
};
}

```



The [Custom](#) position helps to position the legend anywhere in the chart using x and y coordinates.

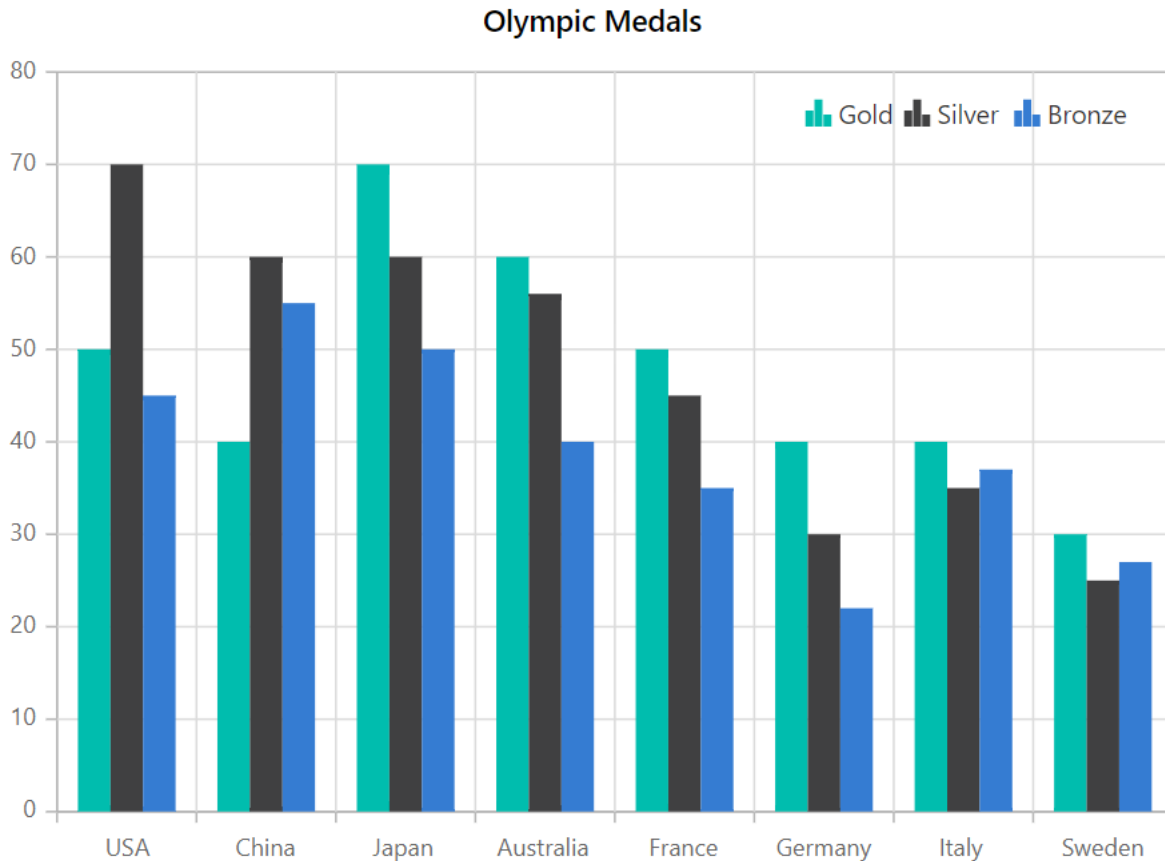
ASPX-CS

```

@using Syncfusion.Blazor.Charts
<SfChart Title="Olympic Medals">
  <ChartPrimaryXAxis ValueType="Syncfusion.Blazor.Charts.ValueType.Category"/>
  <ChartSeriesCollection>

```

```
<ChartSeries DataSource="@MedalDetails" Name="Gold" XName="Country"
Width="2" Opacity="1" YName="Gold" Type="ChartSeriesType.Column">
</ChartSeries>
<ChartSeries DataSource="@MedalDetails" Name="Silver" XName="Country"
Width="2" Opacity="1" YName="Silver" Type="ChartSeriesType.Column">
</ChartSeries>
<ChartSeries DataSource="@MedalDetails" Name="Bronze" XName="Country"
Width="2" Opacity="1" YName="Bronze" Type="ChartSeriesType.Column">
</ChartSeries>
</ChartSeriesCollection>
<ChartLegendSettings Visible="true" Position="LegendPosition.Custom">
<ChartLocation X="200" Y="100"/>
</ChartLegendSettings>
</SfChart>
@code{
public class ChartData
{
public string Country { get; set; }
public double Gold { get; set; }
public double Silver { get; set; }
public double Bronze { get; set; }
}
public List<ChartData> MedalDetails = new List<ChartData>
{
new ChartData{ Country= "USA", Gold=50, Silver=70, Bronze=45 },
new ChartData{ Country="China", Gold=40, Silver= 60, Bronze=55 },
new ChartData{ Country= "Japan", Gold=70, Silver= 60, Bronze=50 },
new ChartData{ Country= "Australia", Gold=60, Silver= 56, Bronze=40 },
new ChartData{ Country= "France", Gold=50, Silver= 45, Bronze=35 },
new ChartData{ Country= "Germany", Gold=40, Silver=30, Bronze=22 },
new ChartData{ Country= "Italy", Gold=40, Silver=35, Bronze=37 },
new ChartData{ Country= "Sweden", Gold=30, Silver=25, Bronze=27 }
};
}
```



<!-- markdownlint-disable MD036 -->

Legend Alignment

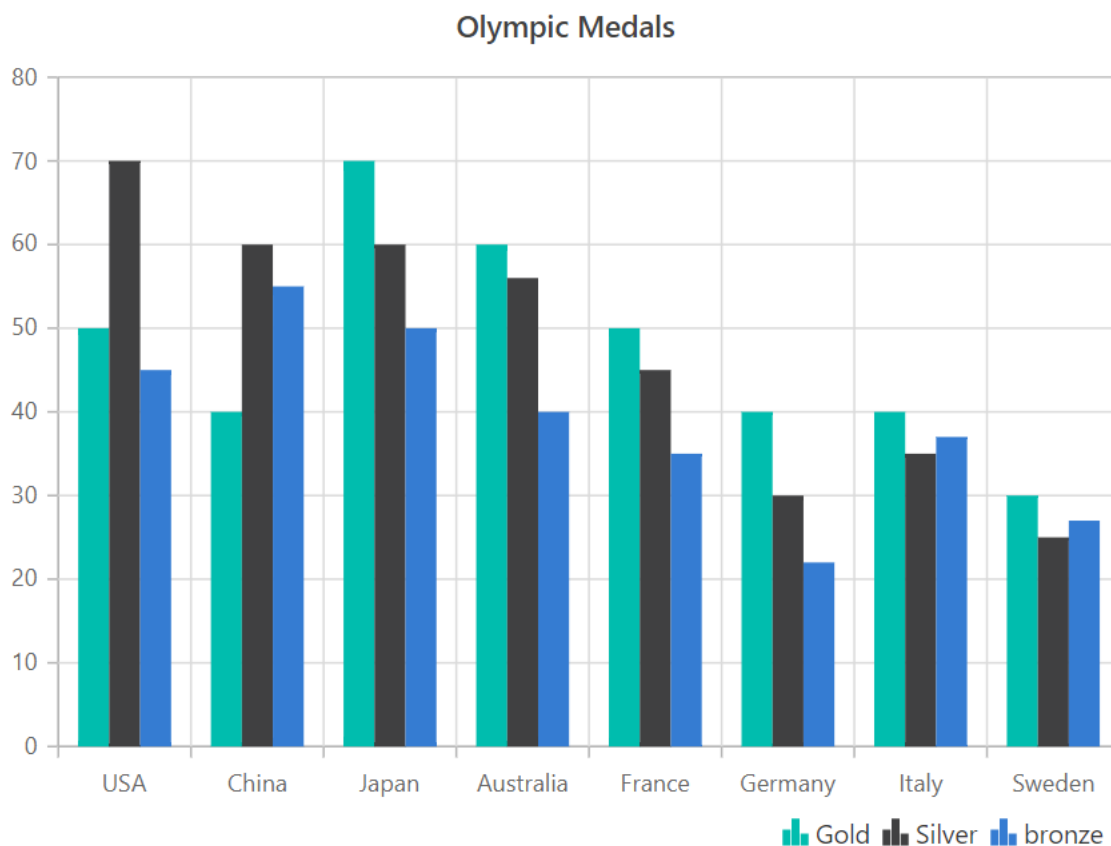
<!-- markdownlint-disable MD036 -->

Using the [Alignment](#) property, place the legend in [Centre](#), [Far](#), or [Near](#) alignment.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart Title="Olympic Medals">
  <ChartPrimaryXAxis ValueType="Syncfusion.Blazor.Charts.ValueType.Category">
  </ChartPrimaryXAxis>
  <ChartSeriesCollection>
    <ChartSeries DataSource="@MedalDetails" Name="Gold" XName="Country"
    Width="2" Opacity="1" YName="Gold" Type="ChartSeriesType.Column">
    </ChartSeries>
    <ChartSeries DataSource="@MedalDetails" Name="Silver" XName="Country"
    Width="2" Opacity="1" YName="Silver" Type="ChartSeriesType.Column">
    </ChartSeries>
    <ChartSeries DataSource="@MedalDetails" Name="Bronze" XName="Country"
    Width="2" Opacity="1" YName="Bronze" Type="ChartSeriesType.Column">
    </ChartSeries>
  </ChartSeriesCollection>
  <ChartLegendSettings Visible="true"
  Alignment="Alignment.Far"></ChartLegendSettings>
</SfChart>
```

```
@code{
public class ChartData
{
    public string Country { get; set; }
    public double Gold { get; set; }
    public double Silver { get; set; }
    public double Bronze { get; set; }
}
public List<ChartData> MedalDetails = new List<ChartData>
{
    new ChartData{ Country= "USA", Gold=50, Silver=70, Bronze=45 },
    new ChartData{ Country="China", Gold=40, Silver= 60, Bronze=55 },
    new ChartData{ Country= "Japan", Gold=70, Silver= 60, Bronze=50 },
    new ChartData{ Country= "Australia", Gold=60, Silver= 56, Bronze=40 },
    new ChartData{ Country= "France", Gold=50, Silver= 45, Bronze=35 },
    new ChartData{ Country= "Germany", Gold=40, Silver=30, Bronze=22 },
    new ChartData{ Country= "Italy", Gold=40, Silver=35, Bronze=37 },
    new ChartData{ Country= "Sweden", Gold=30, Silver=25, Bronze=27 }
};
}
```



Legend Customization

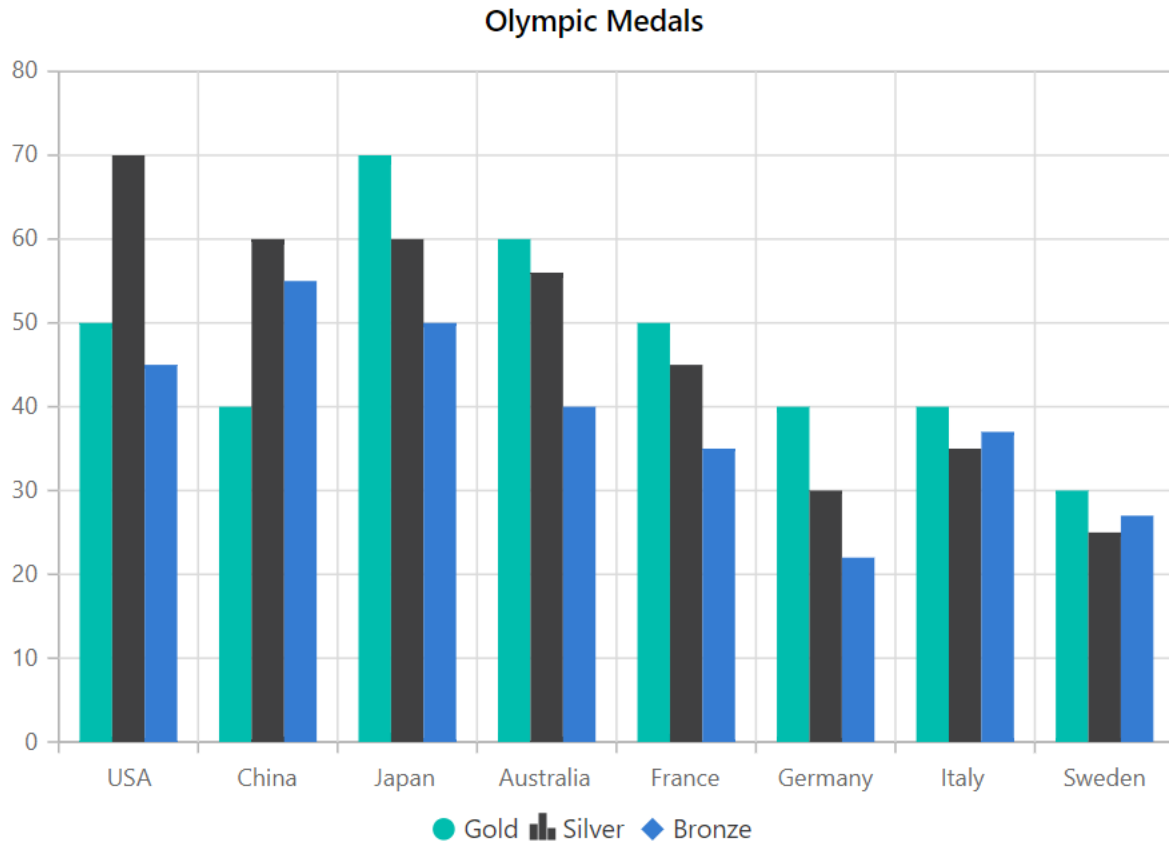
Legend Shape

The [LegendShape](#) property in the [Series](#) can be used to change the shape of the legend icon. The default icon shape for legends is [SeriesType](#).

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart Title="Olympic Medals">
  <ChartPrimaryXAxis ValueType="Syncfusion.Blazor.Charts.ValueType.Category"
  />
  <ChartSeriesCollection>
    <ChartSeries DataSource="@MedalDetails" Name="Gold" XName="Country"
    Width="2" Opacity="1" YName="Gold" Type="ChartSeriesType.Column"
    LegendShape="LegendShape.Circle"/>
    <ChartSeries DataSource="@MedalDetails" Name="Silver" XName="Country"
    Width="2" Opacity="1" YName="Silver" Type="ChartSeriesType.Column"
    LegendShape="LegendShape.SeriesType"/>
    <ChartSeries DataSource="@MedalDetails" Name="Bronze" XName="Country"
    Width="2" Opacity="1" YName="Bronze" Type="ChartSeriesType.Column"
    LegendShape="LegendShape.Diamond"/>
  </ChartSeriesCollection>
  <ChartLegendSettings Visible="true" />
</SfChart>

@code{
public class ChartData
{
public string Country { get; set; }
public double Gold { get; set; }
public double Silver { get; set; }
public double Bronze { get; set; }
}
public List<ChartData> MedalDetails = new List<ChartData>
{
new ChartData{ Country= "USA", Gold=50, Silver=70, Bronze=45 },
new ChartData{ Country="China", Gold=40, Silver= 60, Bronze=55 },
new ChartData{ Country= "Japan", Gold=70, Silver= 60, Bronze=50 },
new ChartData{ Country= "Australia", Gold=60, Silver= 56, Bronze=40 },
new ChartData{ Country= "France", Gold=50, Silver= 45, Bronze=35 },
new ChartData{ Country= "Germany", Gold=40, Silver=30, Bronze=22 },
new ChartData{ Country= "Italy", Gold=40, Silver=35, Bronze=37 },
new ChartData{ Country= "Sweden", Gold=30, Silver=25, Bronze=27 }
};
}
```



Legend Size

When the legend is placed on the top or bottom of the chart, it takes up 20% - 25% of the chart's height, and 20% - 25% of the chart's width when it is positioned on the left or right side of the chart. So, the [Width](#) and [Height](#) properties can be used to adjust the default legend size.

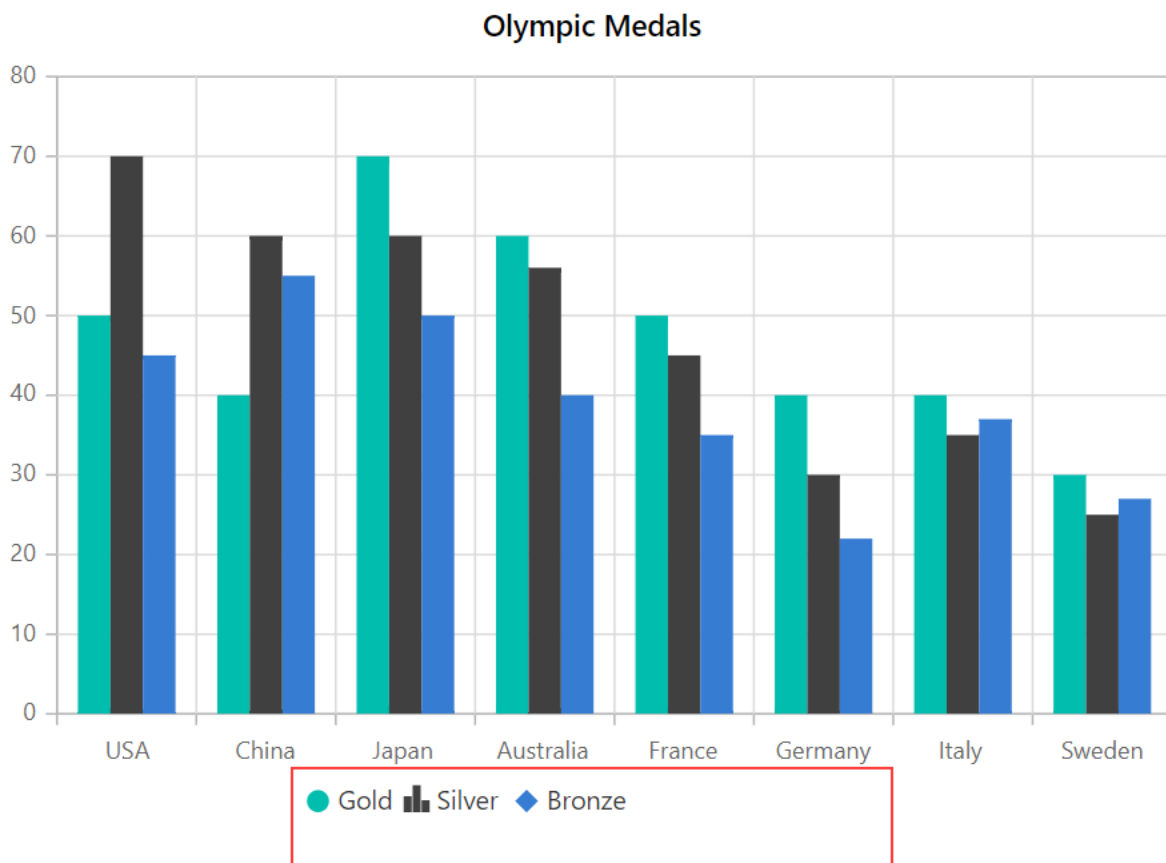
ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart Title="Olympic Medals">
  <ChartPrimaryXAxis ValueTypes="Syncfusion.Blazor.Charts.ValueType.Category"/>
  <ChartSeriesCollection>
    <ChartSeries DataSource="@MedalDetails" Name="Gold" XName="Country"
      Width="2" LegendShape="LegendShape.Circle" Opacity="1" YName="Gold"
      Type="ChartSeriesType.Column">
    </ChartSeries>
    <ChartSeries DataSource="@MedalDetails" Name="Silver" XName="Country"
      Width="2" LegendShape="LegendShape.SeriesType" Opacity="1" YName="Silver"
      Type="ChartSeriesType.Column">
    </ChartSeries>
    <ChartSeries DataSource="@MedalDetails" Name="Bronze" XName="Country"
      Width="2" LegendShape="LegendShape.Diamond" Opacity="1" YName="Bronze"
      Type="ChartSeriesType.Column">
    </ChartSeries>
  </ChartSeriesCollection>
  <ChartLegendSettings Visible="true" Height="50" Width="300" >
  <ChartLegendBorder Color="red" Width="1"/>
</ChartLegendSettings>
```

```

</SfChart>
@code{
public class ChartData
{
    public string Country { get; set; }
    public double Gold { get; set; }
    public double Silver { get; set; }
    public double Bronze { get; set; }
}
public List<ChartData> MedalDetails = new List<ChartData>
{
    new ChartData{ Country= "USA", Gold=50, Silver=70, Bronze=45 },
    new ChartData{ Country="China", Gold=40, Silver= 60, Bronze=55 },
    new ChartData{ Country= "Japan", Gold=70, Silver= 60, Bronze=50 },
    new ChartData{ Country= "Australia", Gold=60, Silver= 56, Bronze=40 },
    new ChartData{ Country= "France", Gold=50, Silver= 45, Bronze=35 },
    new ChartData{ Country= "Germany", Gold=40, Silver=30, Bronze=22 },
    new ChartData{ Country= "Italy", Gold=40, Silver=35, Bronze=37 },
    new ChartData{ Country= "Sweden", Gold=30, Silver=25, Bronze=27 }
};
}

```



Legend Shape Size

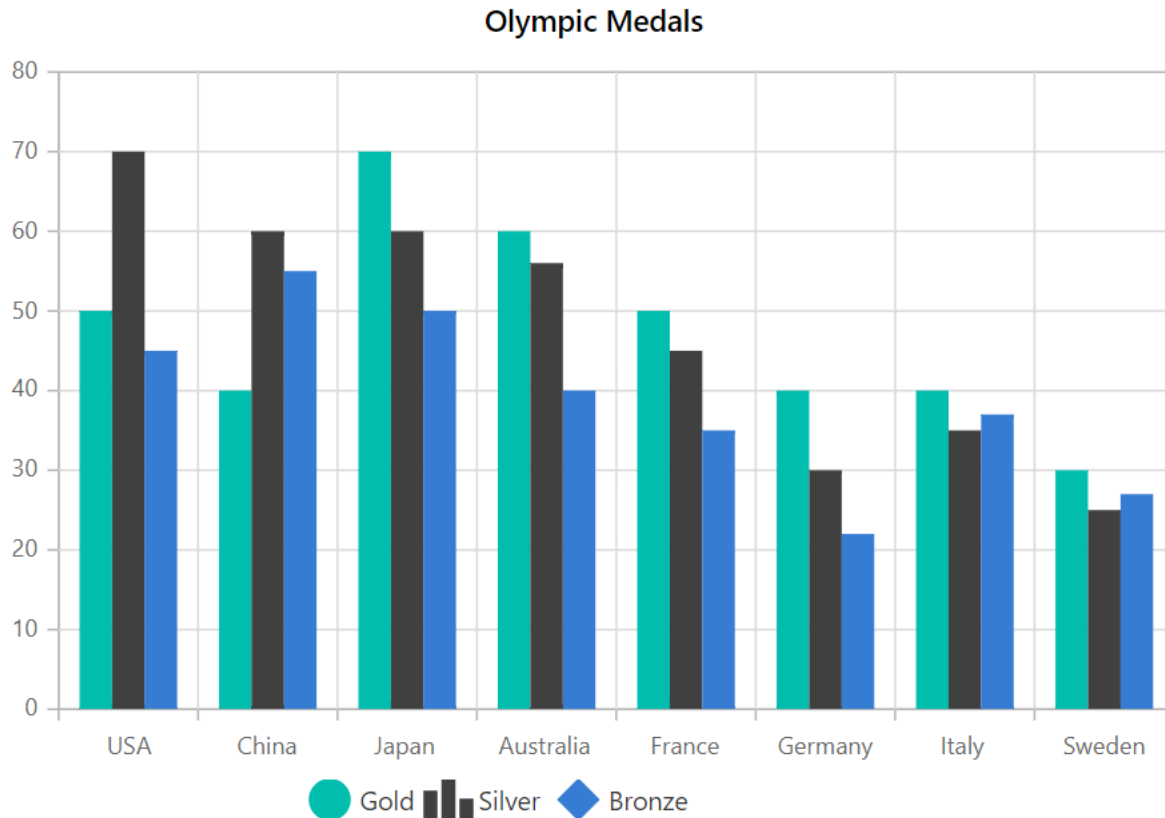
The [ShapeHeight](#) and [ShapeWidth](#) properties can be used to adjust the dimensions of the legend shape.

ASPX-CS


```

@using Syncfusion.Blazor.Charts
<SfChart Title="Olympic Medals">
<ChartPrimaryXAxis ValueType="Syncfusion.Blazor.Charts.ValueType.Category"/>
<ChartSeriesCollection>
<ChartSeries DataSource="@MedalDetails" Name="Gold" XName="Country"
Width="2" LegendShape="LegendShape.Circle" Opacity="1" YName="Gold"
Type="ChartSeriesType.Column">
</ChartSeries>
<ChartSeries DataSource="@MedalDetails" Name="Silver" XName="Country"
Width="2" LegendShape="LegendShape.SeriesType" Opacity="1" YName="Silver"
Type="ChartSeriesType.Column">
</ChartSeries>
<ChartSeries DataSource="@MedalDetails" Name="Bronze" XName="Country"
Width="2" LegendShape="LegendShape.Diamond" Opacity="1" YName="Bronze"
Type="ChartSeriesType.Column">
</ChartSeries>
</ChartSeriesCollection>
<ChartLegendSettings Visible="true" Height="50" Width="300" ShapeHeight="20"
ShapeWidth="20">
</ChartLegendSettings>
</SfChart>
@code{
public class ChartData
{
public string Country { get; set; }
public double Gold { get; set; }
public double Silver { get; set; }
public double Bronze { get; set; }
}
public List<ChartData> MedalDetails = new List<ChartData>
{
new ChartData{ Country= "USA", Gold=50, Silver=70, Bronze=45 },
new ChartData{ Country="China", Gold=40, Silver= 60, Bronze=55 },
new ChartData{ Country= "Japan", Gold=70, Silver= 60, Bronze=50 },
new ChartData{ Country= "Australia", Gold=60, Silver= 56, Bronze=40 },
new ChartData{ Country= "France", Gold=50, Silver= 45, Bronze=35 },
new ChartData{ Country= "Germany", Gold=40, Silver=30, Bronze=22 },
new ChartData{ Country= "Italy", Gold=40, Silver=35, Bronze=37 },
new ChartData{ Country= "Sweden", Gold=30, Silver=25, Bronze=27 }
};
}

```



Legend Paging

When the legend items exceed legend bounds, paging will be enabled by default. End user can view each legend item using the navigation buttons to navigate between pages.

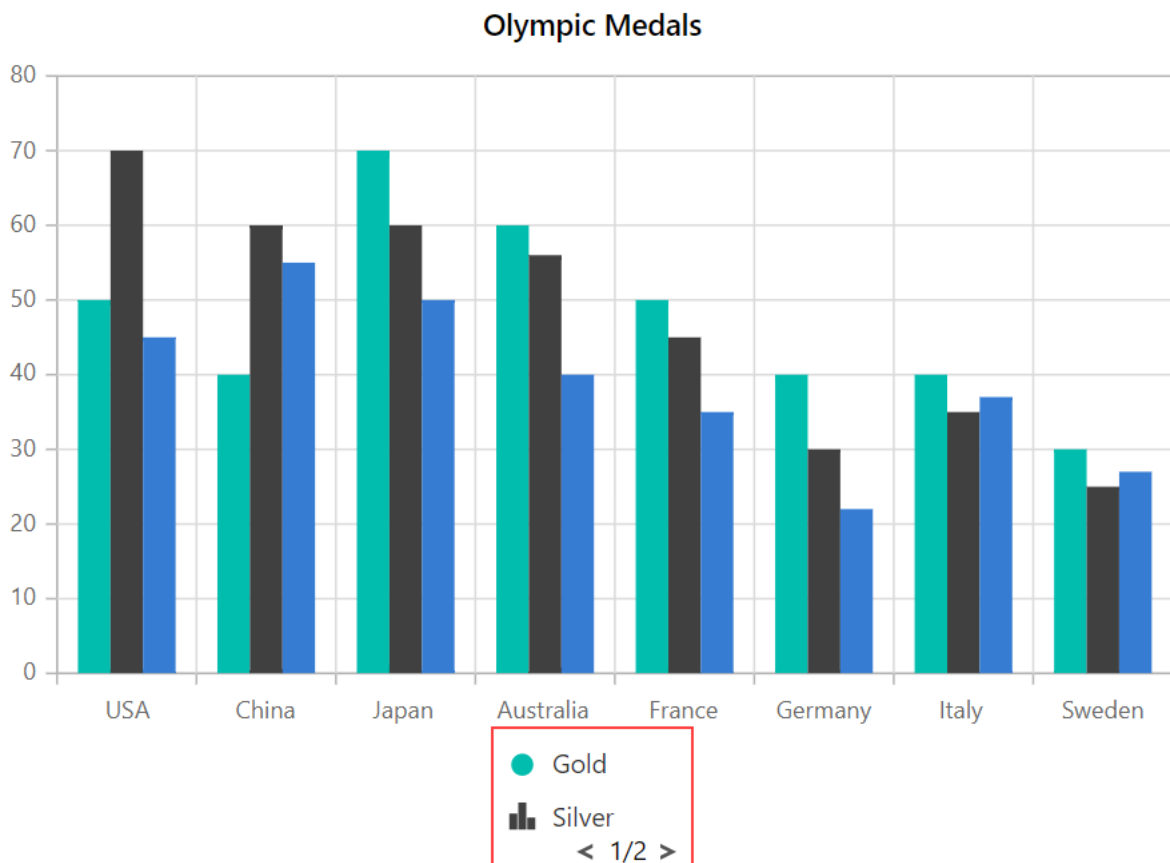
ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart Title="Olympic Medals">
  <ChartPrimaryXAxis ValueType="Syncfusion.Blazor.Charts.ValueType.Category">
  </ChartPrimaryXAxis>
  <ChartSeriesCollection>
    <ChartSeries DataSource="@MedalDetails" Name="Gold" XName="Country"
      Width="2" LegendShape="LegendShape.Circle" Opacity="1" YName="Gold"
      Type="ChartSeriesType.Column">
    </ChartSeries>
    <ChartSeries DataSource="@MedalDetails" Name="Silver" XName="Country"
      Width="2" LegendShape="LegendShape.SeriesType" Opacity="1" YName="Silver"
      Type="ChartSeriesType.Column">
    </ChartSeries>
    <ChartSeries DataSource="@MedalDetails" Name="Bronze" XName="Country"
      Width="2" LegendShape="LegendShape.Diamond" Opacity="1" YName="Bronze"
      Type="ChartSeriesType.Column">
    </ChartSeries>
  </ChartSeriesCollection>
  <ChartLegendSettings Visible="true" Width="100" Height="70" Padding="10"
    ShapePadding="10" >
  <ChartLegendBorder Color="red" Width="1"/>
  </ChartLegendSettings>
```

```

</SfChart>
@code{
public class ChartData
{
    public string Country { get; set; }
    public double Gold { get; set; }
    public double Silver { get; set; }
    public double Bronze { get; set; }
}
public List<ChartData> MedalDetails = new List<ChartData>
{
    new ChartData{ Country= "USA", Gold=50, Silver=70, Bronze=45 },
    new ChartData{ Country="China", Gold=40, Silver= 60, Bronze=55 },
    new ChartData{ Country= "Japan", Gold=70, Silver= 60, Bronze=50 },
    new ChartData{ Country= "Australia", Gold=60, Silver= 56, Bronze=40 },
    new ChartData{ Country= "France", Gold=50, Silver= 45, Bronze=35 },
    new ChartData{ Country= "Germany", Gold=40, Silver=30, Bronze=22 },
    new ChartData{ Country= "Italy", Gold=40, Silver=35, Bronze=37 },
    new ChartData{ Country= "Sweden", Gold=30, Silver=25, Bronze=27 }
};
}

```



Series selection based on legend

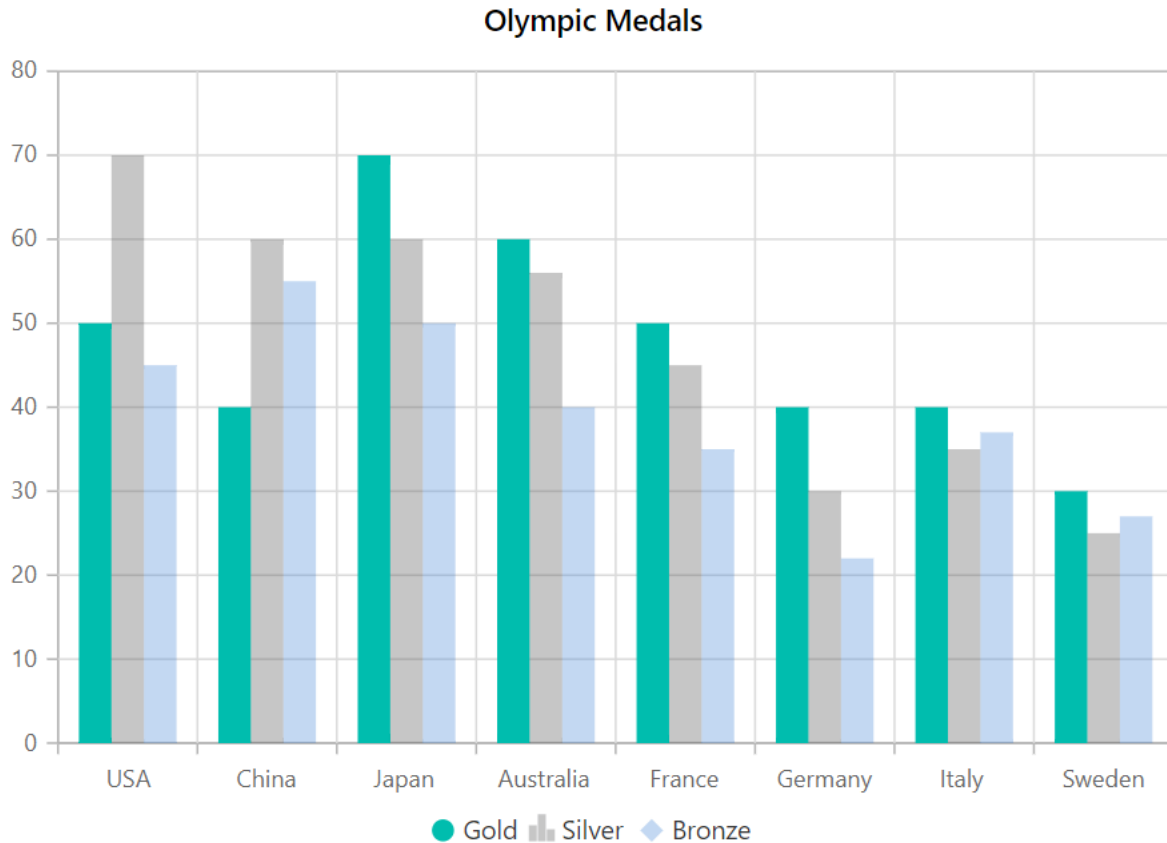
By default, when click on the legend item, the appropriate series visibility is collapsed. On the other hand, [ToggleVisibility](#) property is used to disable such functionality.

ASPX-CS

```

@using Syncfusion.Blazor.Charts
<SfChart Title="Olympic Medals" SelectionMode="SelectionMode.Series">
<ChartPrimaryXAxis ValueTypes="Syncfusion.Blazor.Charts.ValueType.Category"/>
<ChartSeriesCollection>
<ChartSeries DataSource="@MedalDetails" Name="Gold" XName="Country"
Width="2" LegendShape="LegendShape.Circle" Opacity="1" YName="Gold"
Type="ChartSeriesType.Column">
</ChartSeries>
<ChartSeries DataSource="@MedalDetails" Name="Silver" XName="Country"
Width="2" LegendShape="LegendShape.SeriesType" Opacity="1" YName="Silver"
Type="ChartSeriesType.Column">
</ChartSeries>
<ChartSeries DataSource="@MedalDetails" Name="Bronze" XName="Country"
Width="2" LegendShape="LegendShape.Diamond" Opacity="1" YName="Bronze"
Type="ChartSeriesType.Column">
</ChartSeries>
</ChartSeriesCollection>
<ChartLegendSettings Visible="true" ToggleVisibility="false"/>
</SfChart>
@code{
public class ChartData
{
public string Country { get; set; }
public double Gold { get; set; }
public double Silver { get; set; }
public double Bronze { get; set; }
}
public List<ChartData> MedalDetails = new List<ChartData>
{
new ChartData{ Country= "USA", Gold=50, Silver=70, Bronze=45 },
new ChartData{ Country="China", Gold=40, Silver= 60, Bronze=55 },
new ChartData{ Country= "Japan", Gold=70, Silver= 60, Bronze=50 },
new ChartData{ Country= "Australia", Gold=60, Silver= 56, Bronze=40 },
new ChartData{ Country= "France", Gold=50, Silver= 45, Bronze=35 },
new ChartData{ Country= "Germany", Gold=40, Silver=30, Bronze=22 },
new ChartData{ Country= "Italy", Gold=40, Silver=35, Bronze=37 },
new ChartData{ Country= "Sweden", Gold=30, Silver=25, Bronze=27 }
};
}

```



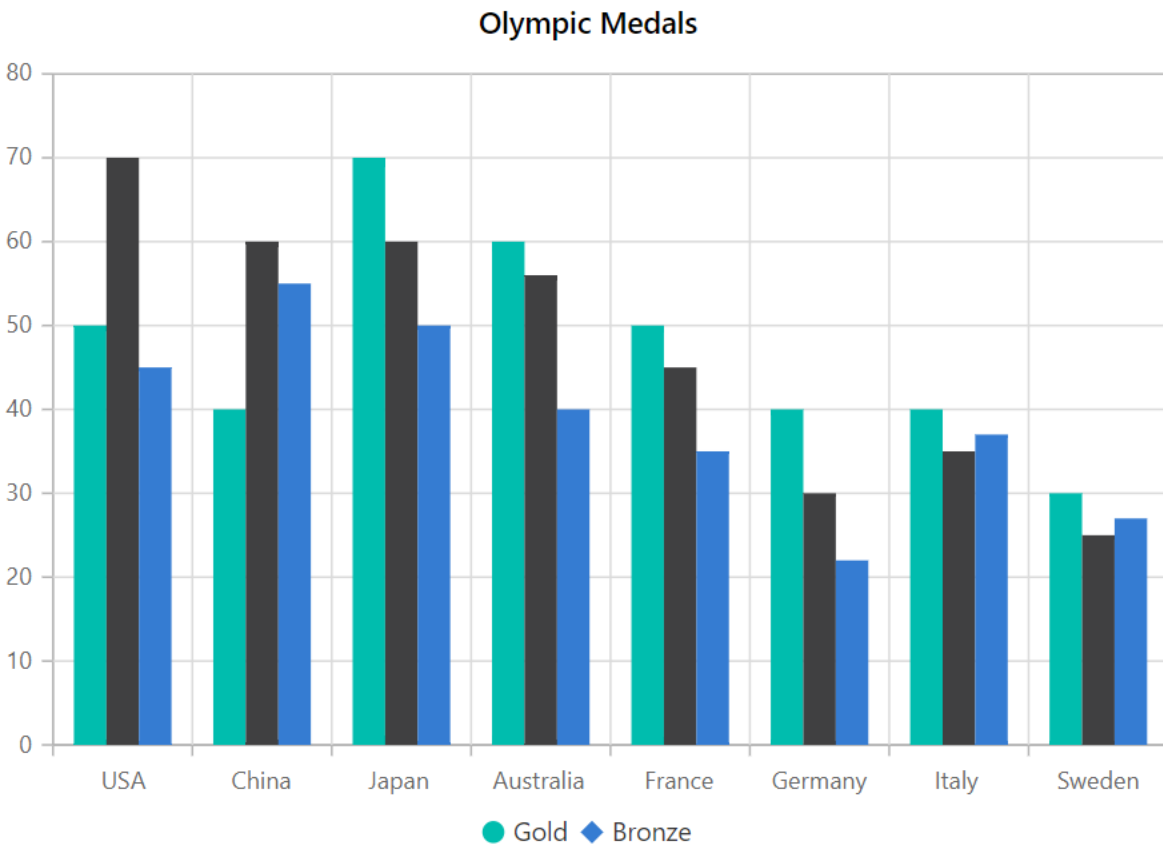
Hiding legend item

The series [Name](#) will be displayed as the legend text by default. One can skip the legend for particular series by providing an empty string to the series [Name](#) property.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart Title="Olympic Medals">
  <ChartPrimaryXAxis Value="Syncfusion.Blazor.Charts.ValueType.Category">
  </ChartPrimaryXAxis>
  <ChartSeriesCollection>
    <ChartSeries DataSource="@MedalDetails" Name="Gold" XName="Country"
      Width="2" LegendShape="LegendShape.Circle" Opacity="1" YName="Gold"
      Type="ChartSeriesType.Column">
    </ChartSeries>
    <ChartSeries DataSource="@MedalDetails" Name="" XName="Country" Width="2"
      LegendShape="LegendShape.SeriesType" Opacity="1" YName="Silver"
      Type="ChartSeriesType.Column">
    </ChartSeries>
    <ChartSeries DataSource="@MedalDetails" Name="Bronze" XName="Country"
      Width="2" LegendShape="LegendShape.Diamond" Opacity="1" YName="Bronze"
      Type="ChartSeriesType.Column">
    </ChartSeries>
  </ChartSeriesCollection>
  <ChartLegendSettings Visible="true" ToggleVisibility="true">
  </ChartLegendSettings>
</SfChart>
@code{
```

```
public class ChartData
{
    public string Country { get; set; }
    public double Gold { get; set; }
    public double Silver { get; set; }
    public double Bronze { get; set; }
}
public List<ChartData> MedalDetails = new List<ChartData>
{
    new ChartData{ Country= "USA", Gold=50, Silver=70, Bronze=45 },
    new ChartData{ Country="China", Gold=40, Silver= 60, Bronze=55 },
    new ChartData{ Country= "Japan", Gold=70, Silver= 60, Bronze=50 },
    new ChartData{ Country= "Australia", Gold=60, Silver= 56, Bronze=40 },
    new ChartData{ Country= "France", Gold=50, Silver= 45, Bronze=35 },
    new ChartData{ Country= "Germany", Gold=40, Silver=30, Bronze=22 },
    new ChartData{ Country= "Italy", Gold=40, Silver=35, Bronze=37 },
    new ChartData{ Country= "Sweden", Gold=30, Silver=25, Bronze=27 }
};
```



Refer to our [Blazor Charts](#) feature tour page for its groundbreaking feature representations and also explore our [Blazor Chart Example](#) to know various chart types and how to represent time-dependent data, showing trends at equal intervals.

See Also

- [Data label](#)
- [Marker](#)

Tooltip in Blazor Charts Component

<!-- markdownlint-disable MD036 -->

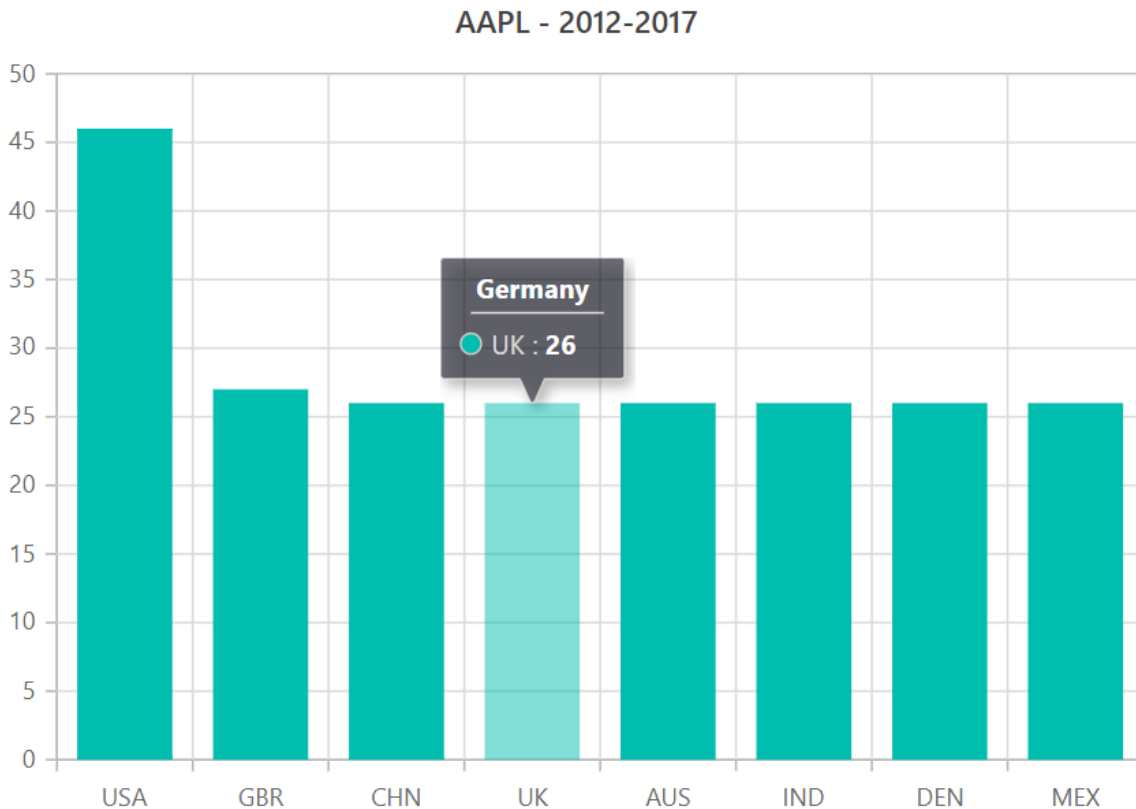
When the mouse is moved over a point on the chart, the tooltip will provide information about that point.

Enable Tooltip

When space constraints prevent displaying information using data labels, the tooltip comes in useful. The [Enable](#) property in [ChartTooltipSettings](#) can be set to **true** to enable the tooltip.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart Title="Product Sales">
<ChartPrimaryXAxis ValueType="Syncfusion.Blazor.Charts.ValueType.Category"
/>
<ChartPrimaryYAxis LabelFormat="{value}M" >
</ChartPrimaryYAxis>
<ChartTooltipSettings Enable="true"></ChartTooltipSettings>
<ChartSeriesCollection>
<ChartSeries DataSource="@SalesReports" Name="Text" XName="X" YName="Y"
Type="ChartSeriesType.Column">
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code{
public class Data
{
public string X { get; set; }
public double Y { get; set; }
public string Text { get; set; }
}
public List<Data> SalesReports = new List<Data>
{
new Data{ X= "Jan", Y= 3, Text= "January" },
new Data{ X= "Feb", Y= 3.5, Text= "February" },
new Data{ X= "Mar", Y= 7, Text= "March" },
new Data{ X= "Apr", Y= 13.5, Text= "April" }
};
}
```



Tooltip Format

<!-- markdownlint-disable MD013 -->

By default, the tooltip displays information in points for the x and y values. In addition, further information can be displayed in the tooltip. For example, the format **`$series.name $point.x`** displays series name and point x-value in the tooltip.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart Title="Product Sales">
  <ChartPrimaryXAxis ValueType="Syncfusion.Blazor.Charts.ValueType.Category"
  />
  <ChartPrimaryYAxis LabelFormat="{value}M"></ChartPrimaryYAxis>
  <ChartTooltipSettings Enable="true" Header="Sales" Format="<b>${series.name}
  : ${point.y}</b>"></ChartTooltipSettings>
  <ChartSeriesCollection>
    <ChartSeries DataSource="@SalesReports" XName="X" YName="Y"
    Type="ChartSeriesType.Column">
  </ChartSeries>
  </ChartSeriesCollection>
</SfChart>

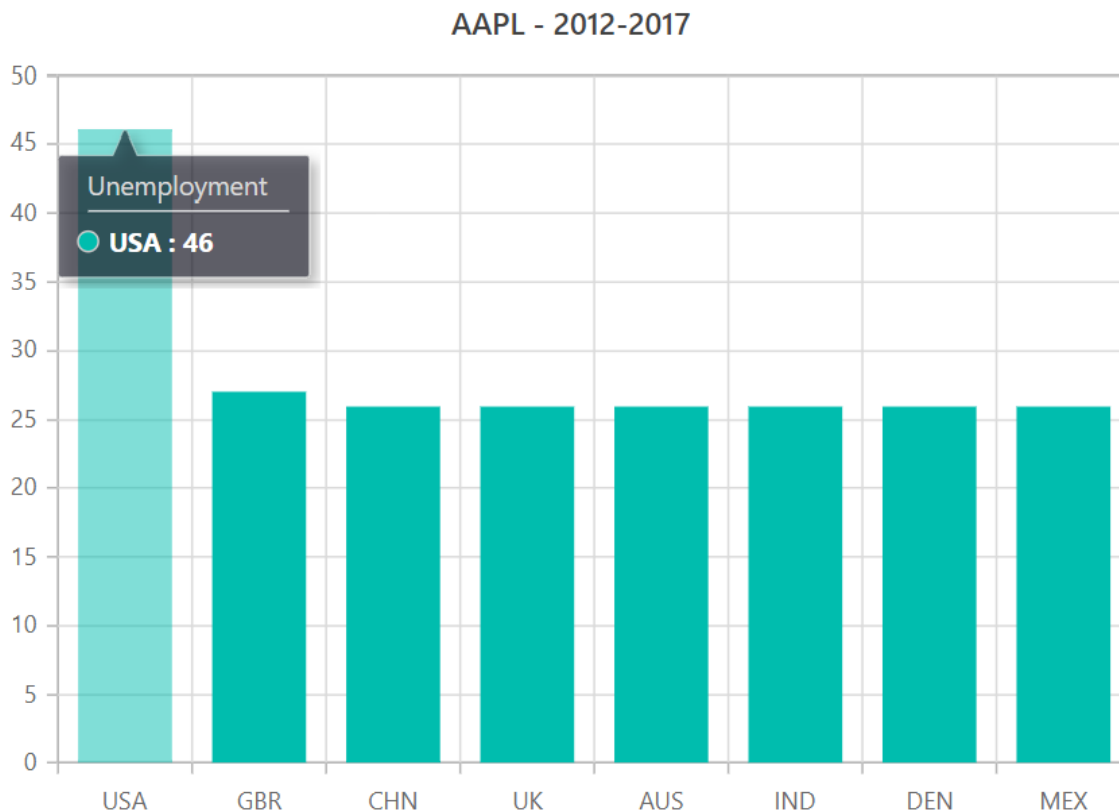
@code{
public class Data
{
    public string X { get; set; }
    public double Y { get; set; }
    public string Text { get; set; }
}
```



```

}
public List<Data> SalesReports = new List<Data>
{
    new Data{ X= "Jan", Y= 3, Text= "January" },
    new Data{ X= "Feb", Y= 3.5, Text= "February" },
    new Data{ X= "Mar", Y= 7, Text= "March" },
    new Data{ X= "Apr", Y= 13.5, Text= "April" }
};
}

```



<!-- markdownlint-disable MD013 -->

Tooltip Customization

The [Fill](#) and [Border](#) properties are used to customize the background color and the border of the tooltip respectively. The [ChartTooltipTextStyle](#) is used to customize the tooltip text.

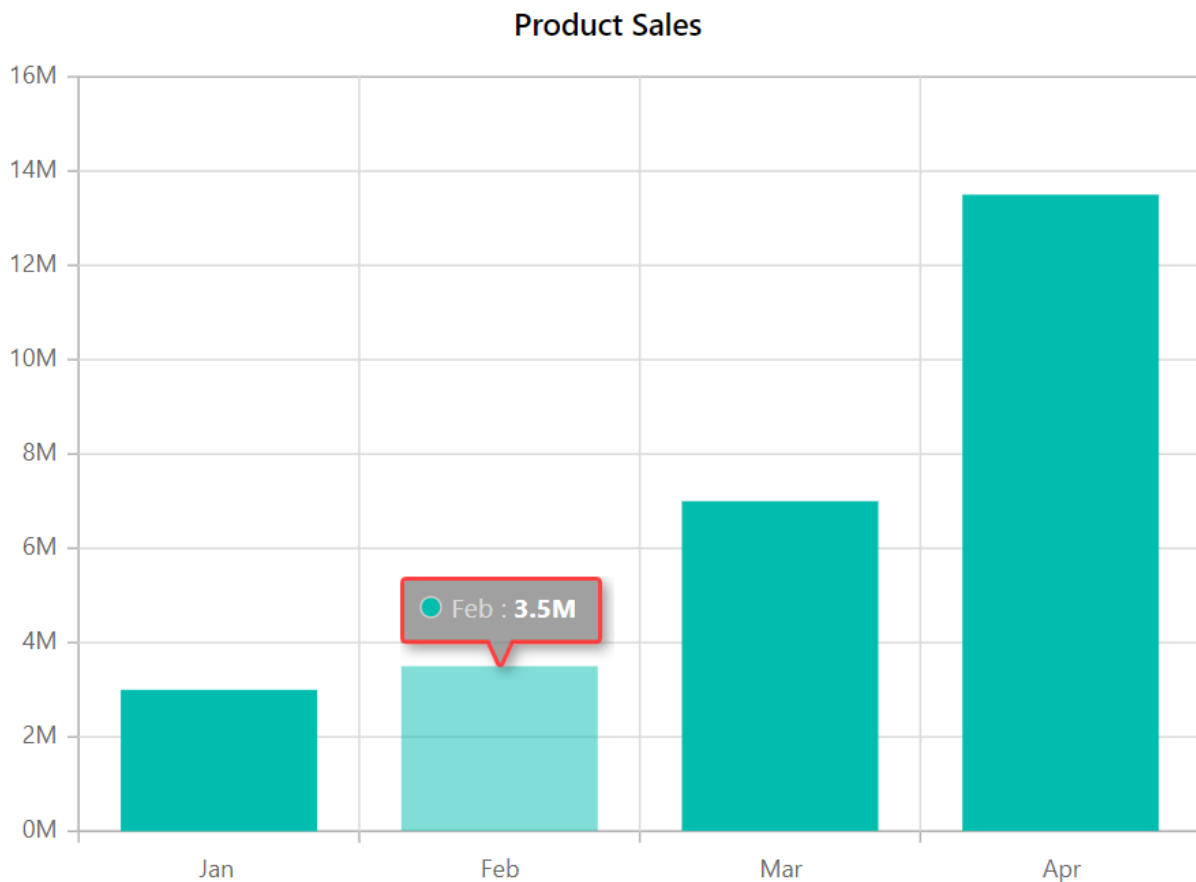
ASPX-CS

```

@using Syncfusion.Blazor.Charts
<SfChart Title="Product Sales">
  <ChartPrimaryXAxis ValueTypes="Syncfusion.Blazor.Charts.ValueType.Category"
  />
  <ChartPrimaryYAxis LabelFormat="{value}M" >
  </ChartPrimaryYAxis>
  <ChartTooltipSettings Enable="true" Fill="gray">
  <ChartTooltipBorder Color="#FF0000" Width="2"></ChartTooltipBorder>
  </ChartTooltipSettings>
  <ChartSeriesCollection>

```

```
<ChartSeries DataSource="@SalesReports" XName="X" YName="Y"
Type="ChartSeriesType.Column">
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code{
public class Data
{
public string X { get; set; }
public double Y { get; set; }
public string Text { get; set; }
}
public List<Data> SalesReports = new List<Data>
{
new Data{ X= "Jan", Y= 3, Text= "January" },
new Data{ X= "Feb", Y= 3.5, Text= "February" },
new Data{ X= "Mar", Y= 7, Text= "March" },
new Data{ X= "Apr", Y= 13.5, Text= "April" }
};
}
```



Refer to our [Blazor Charts](#) feature tour page for its groundbreaking feature representations and also explore our [Blazor Chart Example](#) to know various chart types and how to represent time-dependent data, showing trends at equal intervals.

See Also

- [Data label](#)
- [Marker](#)

Zooming in Blazor Charts Component

Enable Zooming

The chart can be zoomed in three different ways.

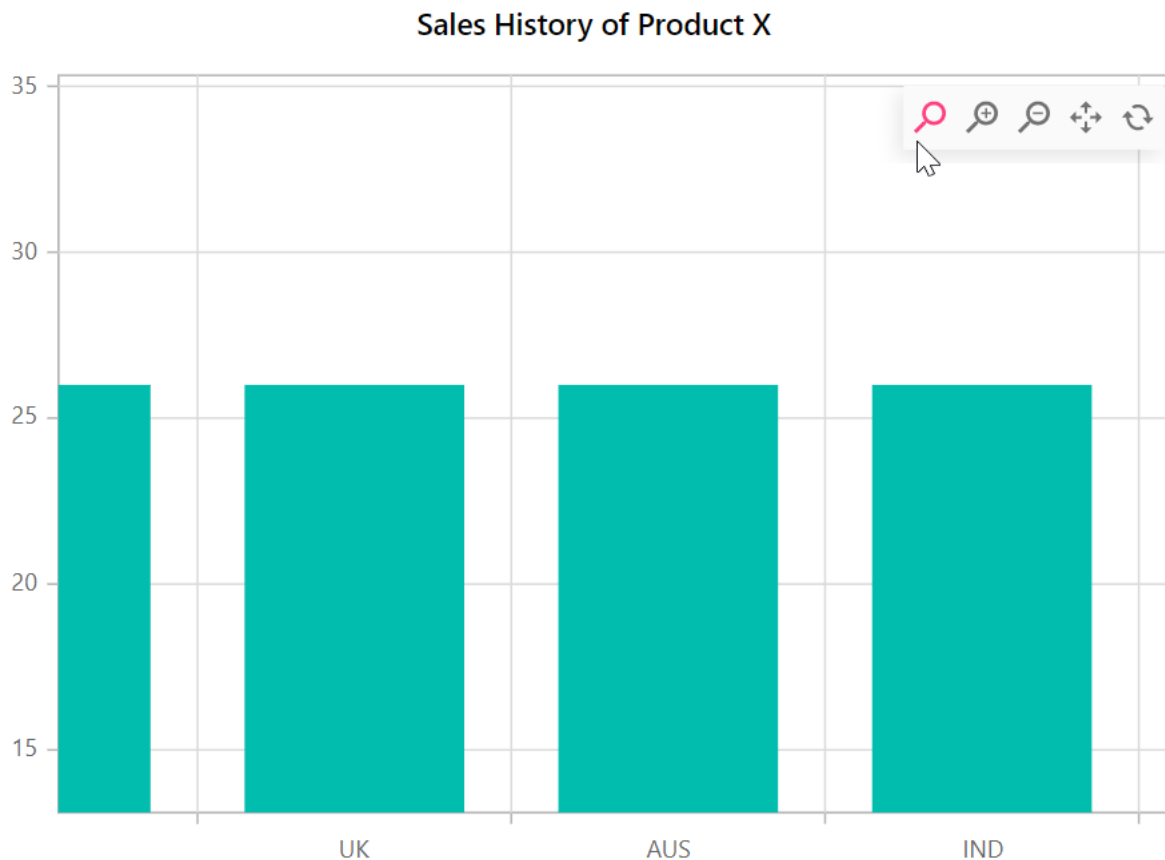
- Selection - By setting [EnableSelectionZooming](#) property to **true** in [ChartZoomSettings](#), the chart can be zoomed using the rubber band selection.
- Mouse Wheel - By setting [EnableMouseWheelZooming](#) property to **true** in [ChartZoomSettings](#), the chart can be zoomed-in and zoomed-out by scrolling the mouse wheel.
- Pinch - By setting [EnablePinchZooming](#) property to **true** in [ChartZoomSettings](#), the chart can be zoomed through pinch gesture in touch enabled devices.

Pinch zooming is only usable in browsers that support multi-touch gestures.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart Title="Sales History of Product X">
  <ChartPrimaryXAxis
    ValueType="Syncfusion.Blazor.Charts.ValueType.Category"></ChartPrimaryXAxis>
  <ChartZoomSettings EnableMouseWheelZooming="true" EnablePinchZooming="true"
    EnableSelectionZooming="true"></ChartZoomSettings>
  <ChartSeriesCollection>
    <ChartSeries DataSource="@SalesReports" XName="X" YName="YValue"
      Type="ChartSeriesType.Column"></ChartSeries>
  </ChartSeriesCollection>
</SfChart>

@code{
public class ChartData
{
    public string X { get; set; }
    public double YValue { get; set; }
}
public List<ChartData> SalesReports = new List<ChartData>
{
    new ChartData { X= "USA", YValue= 46 },
    new ChartData { X= "GBR", YValue= 27 },
    new ChartData { X= "CHN", YValue= 26 },
    new ChartData { X= "UK", YValue= 26 },
    new ChartData { X= "AUS", YValue= 26 },
    new ChartData { X= "IND", YValue= 26 },
    new ChartData { X= "DEN", YValue= 26 },
    new ChartData { X= "MEX", YValue= 26 },
};
}
```



A zooming toolbar will show after zooming the chart, featuring options for **Zoom**, **Zoom In**, **Zoom Out**, **Pan**, and **Reset**. The **Pan** option allows you to pan the chart, while the **Reset** option allows you to reset the zoomed chart.

Modes

The [Mode](#) property in [ChartZoomSettings](#) determines whether the chart can scale along the horizontal or vertical axes. The default value of the mode is XY (both axis).

There are three types of modes.

- X - Allows us to zoom the chart horizontally.
- Y - Allows us to zoom the chart vertically.
- XY - Allows us to zoom the chart both vertically and horizontally.

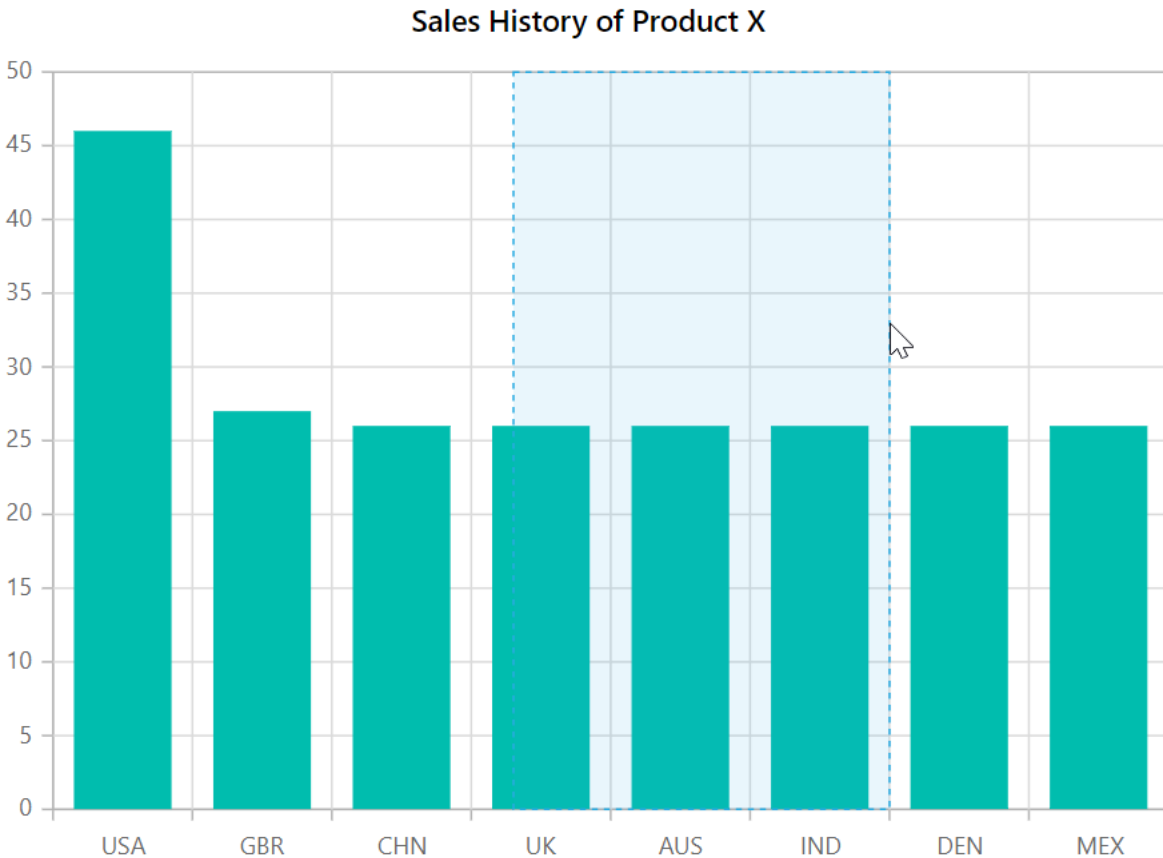
ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart Title="Sales History of Product X">
  <ChartPrimaryXAxis
    ValueType="Syncfusion.Blazor.Charts.ValueType.Category"></ChartPrimaryXAxis>
  <ChartZoomSettings EnableSelectionZooming="true"
    Mode="ZoomMode.X"></ChartZoomSettings>
  <ChartSeriesCollection>
    <ChartSeries DataSource="@SalesReports" XName="X" YName="YValue"
      Type="ChartSeriesType.Column"></ChartSeries>
  </ChartSeriesCollection>
```

```

</SfChart>
@code{
public class ChartData
{
public string X { get; set; }
public double YValue { get; set; }
}
public List<ChartData> SalesReports = new List<ChartData>
{
new ChartData { X= "USA", YValue= 46 },
new ChartData { X= "GBR", YValue= 27 },
new ChartData { X= "CHN", YValue= 26 },
new ChartData { X= "UK", YValue= 26 },
new ChartData { X= "AUS", YValue= 26 },
new ChartData { X= "IND", YValue= 26 },
new ChartData { X= "DEN", YValue= 26 },
new ChartData { X= "MEX", YValue= 26 },
};
}

```



Toolbar

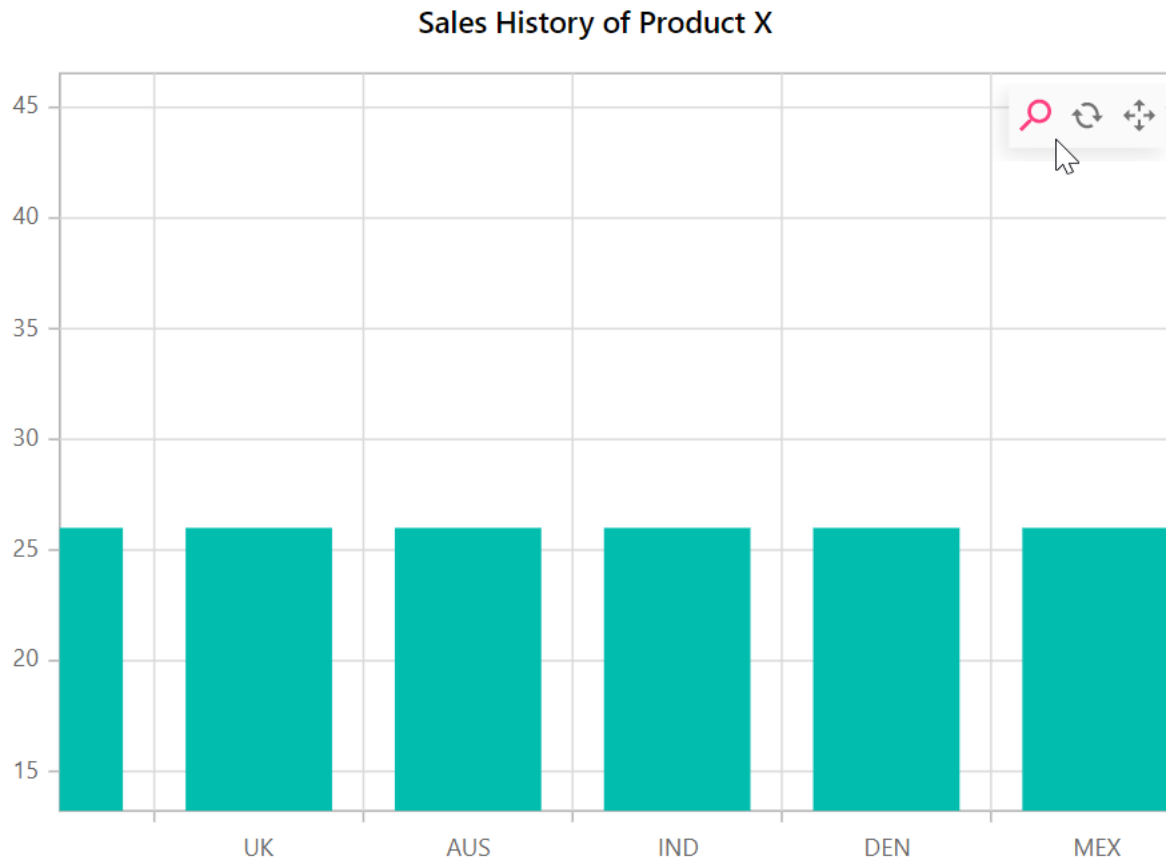
By default, zoom in, zoom out, pan, and reset buttons are available in the toolbar for zoomed charts. The [ToolbarItems](#) property specifies which tools should be displayed in the toolbar.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
```

```
<SfChart Title="Sales History of Product X">
  <ChartPrimaryXAxis
    ValueType="Syncfusion.Blazor.Charts.ValueType.Category"></ChartPrimaryXAxis>
  <ChartZoomSettings EnableSelectionZooming="true"
    EnableMouseWheelZooming="true"
    EnablePinchZooming="true" ToolbarItems="@ToolbarItem">
  </ChartZoomSettings>
  <ChartSeriesCollection>
    <ChartSeries DataSource="@SalesReports" XName="X" YName="YValue"
      Type="ChartSeriesType.Column"></ChartSeries>
  </ChartSeriesCollection>
</SfChart>

@code{
public List<ToolbarItems> ToolbarItem = new List<ToolbarItems>() {
    ToolbarItems.Zoom, ToolbarItems.Reset, ToolbarItems.Pan };
public class ChartData
{
    public string X { get; set; }
    public double YValue { get; set; }
}
public List<ChartData> SalesReports = new List<ChartData>
{
    new ChartData { X= "USA", YValue= 46 },
    new ChartData { X= "GBR", YValue= 27 },
    new ChartData { X= "CHN", YValue= 26 },
    new ChartData { X= "UK", YValue= 26 },
    new ChartData { X= "AUS", YValue= 26 },
    new ChartData { X= "IND", YValue= 26 },
    new ChartData { X= "DEN", YValue= 26 },
    new ChartData { X= "MEX", YValue= 26 },
};
}
```



Enable Pan

By using the [EnablePan](#) property, one can pan the zoomed chart without the help of toolbar items.

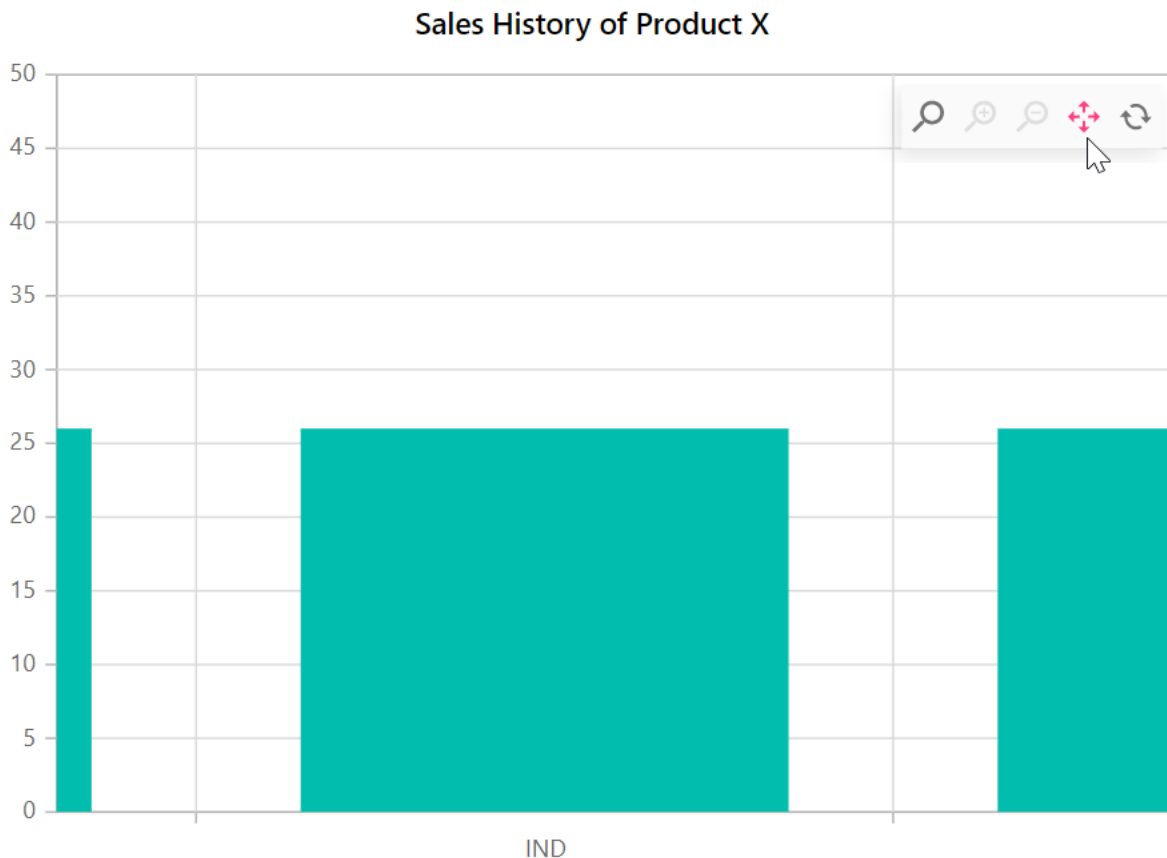
ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart Title="Sales History of Product X">
  <ChartPrimaryXAxis ValueType="Syncfusion.Blazor.Charts.ValueType.Category"
    ZoomFactor="0.2" ZoomPosition="0.6"></ChartPrimaryXAxis>
  <ChartZoomSettings EnableSelectionZooming="true"
    EnablePan="true"></ChartZoomSettings>
  <ChartSeriesCollection>
    <ChartSeries DataSource="@SalesReports" XName="X" YName="YValue"
      Type="ChartSeriesType.Column"></ChartSeries>
  </ChartSeriesCollection>
</SfChart>

@code{
public class ChartData
{
    public string X { get; set; }
    public double YValue { get; set; }
}

public List<ChartData> SalesReports = new List<ChartData>
{
    new ChartData { X= "USA", YValue= 46 },
    new ChartData { X= "GBR", YValue= 27 },
    new ChartData { X= "CHN", YValue= 26 },
}
```

```
new ChartData { X= "UK", YValue= 26 },
new ChartData { X= "AUS", YValue= 26 },
new ChartData { X= "IND", YValue= 26 },
new ChartData { X= "DEN", YValue= 26 },
new ChartData { X= "MEX", YValue= 26 },
};
}
```



Enable Scrollbar

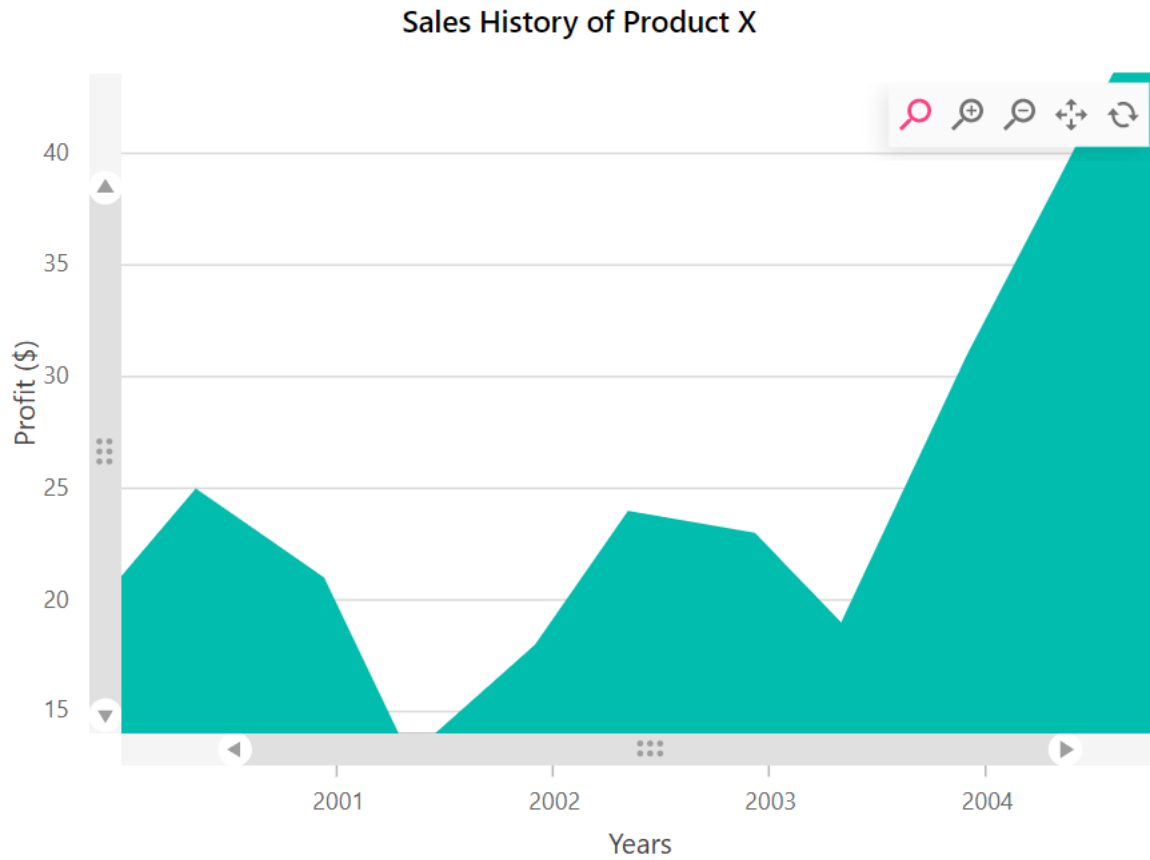
The [EnableScrollbar](#) property can be used to add a scrollbar to a zoomed chart. The chart can be panned or zoomed using this scrollbar.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart Title="Sales History of Product X">
  <ChartPrimaryXAxis Title="Years"
    ValueType="Syncfusion.Blazor.Charts.ValueType.DateTime"
    Skeleton="yMMM" EdgeLabelPlacement="EdgeLabelPlacement.Shift">
    <ChartAxisMajorGridLines Width="0"></ChartAxisMajorGridLines>
  </ChartPrimaryXAxis>
  <ChartPrimaryYAxis Title="Profit ($)" RangePadding="ChartRangePadding.None">
    <ChartAxisLineStyle Width="0"></ChartAxisLineStyle>
    <ChartAxisMajorTickLines Width="0"></ChartAxisMajorTickLines>
  </ChartPrimaryYAxis>
  <ChartLegendSettings Visible="false"></ChartLegendSettings>
```



```
<ChartZoomSettings EnableMouseWheelZooming="true" EnableScrollbar="true"
EnablePinchZooming="true"
EnableSelectionZooming="true"></ChartZoomSettings>
<ChartSeriesCollection>
<ChartSeries DataSource="@SalesReports" Name="Warmest" XName="XValue"
Width="2" Opacity="1"
YName="YValue" Type="ChartSeriesType.Area">
</ChartSeries>
</ChartSeriesCollection>
<ChartArea>
<ChartAreaBorder Width="0"></ChartAreaBorder>
</ChartArea>
</SfChart>
@code {
public class ChartData
{
public DateTime XValue { get; set; }
public double YValue { get; set; }
}
public List<ChartData> SalesReports = new List<ChartData>
{
new ChartData { XValue = new DateTime(2000, 02, 11), YValue = 14 },
new ChartData { XValue = new DateTime(2000, 09, 04), YValue = 20 },
new ChartData { XValue = new DateTime(2001, 02, 11), YValue = 25 },
new ChartData { XValue = new DateTime(2001, 09, 16), YValue = 21 },
new ChartData { XValue = new DateTime(2002, 02, 07), YValue = 13 },
new ChartData { XValue = new DateTime(2002, 09, 07), YValue = 18 },
new ChartData { XValue = new DateTime(2003, 02, 11), YValue = 24 },
new ChartData { XValue = new DateTime(2003, 09, 14), YValue = 23 },
new ChartData { XValue = new DateTime(2004, 02, 06), YValue = 19 },
new ChartData { XValue = new DateTime(2004, 09, 06), YValue = 31 },
new ChartData { XValue = new DateTime(2005, 02, 11), YValue = 39 },
new ChartData { XValue = new DateTime(2005, 09, 11), YValue = 50 },
new ChartData { XValue = new DateTime(2006, 02, 11), YValue = 24 },
};
}
```



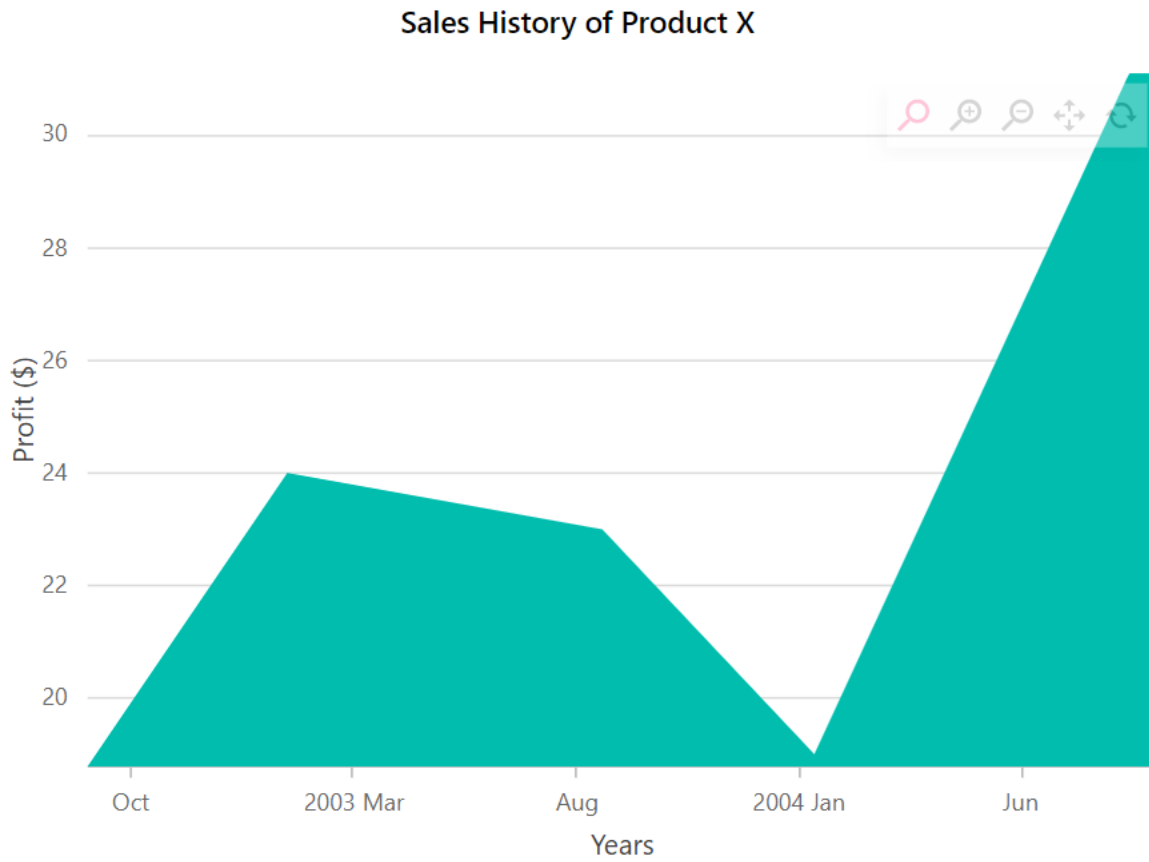
Auto interval on zooming

The axis interval will be calculated automatically with respect to the zoomed range, if the [EnableAutoIntervalOnZooming](#) property is set to **true**.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart Title="Sales History of Product X">
  <ChartPrimaryXAxis Title="Years"
    ValueType="Syncfusion.Blazor.Charts.ValueType.DateTime"
    Skeleton="yMMM" EdgeLabelPlacement="EdgeLabelPlacement.Shift"
    EnableAutoIntervalOnZooming="true">
    <ChartAxisMajorGridLines Width="0"></ChartAxisMajorGridLines>
  </ChartPrimaryXAxis>
  <ChartPrimaryYAxis Title="Profit ($)" RangePadding="ChartRangePadding.None">
    <ChartAxisLineStyle Width="0"></ChartAxisLineStyle>
    <ChartAxisMajorTickLines Width="0"></ChartAxisMajorTickLines>
  </ChartPrimaryYAxis>
  <ChartLegendSettings Visible="false"></ChartLegendSettings>
  <ChartZoomSettings EnableMouseWheelZooming="true" EnablePinchZooming="true"
    EnableSelectionZooming="true"></ChartZoomSettings>
  <ChartSeriesCollection>
    <ChartSeries DataSource="@SalesReports" Name="Warmest" XName="XValue"
      Width="2" Opacity="1"
      YName="YValue" Type="ChartSeriesType.Area">
    </ChartSeries>
  </ChartSeriesCollection>
```

```
<ChartArea>
<ChartAreaBorder Width="0"></ChartAreaBorder>
</ChartArea>
</SfChart>
@code {
public class ChartData
{
public DateTime XValue { get; set; }
public double YValue { get; set; }
}
public List<ChartData> SalesReports = new List<ChartData>
{
new ChartData { XValue = new DateTime(2000, 02, 11), YValue = 14 },
new ChartData { XValue = new DateTime(2000, 09, 04), YValue = 20 },
new ChartData { XValue = new DateTime(2001, 02, 11), YValue = 25 },
new ChartData { XValue = new DateTime(2001, 09, 16), YValue = 21 },
new ChartData { XValue = new DateTime(2002, 02, 07), YValue = 13 },
new ChartData { XValue = new DateTime(2002, 09, 07), YValue = 18 },
new ChartData { XValue = new DateTime(2003, 02, 11), YValue = 24 },
new ChartData { XValue = new DateTime(2003, 09, 14), YValue = 23 },
new ChartData { XValue = new DateTime(2004, 02, 06), YValue = 19 },
new ChartData { XValue = new DateTime(2004, 09, 06), YValue = 31 },
new ChartData { XValue = new DateTime(2005, 02, 11), YValue = 39 },
new ChartData { XValue = new DateTime(2005, 09, 11), YValue = 50 },
new ChartData { XValue = new DateTime(2006, 02, 11), YValue = 24 },
};
}
```



Refer to our [Blazor Charts](#) feature tour page for its groundbreaking feature representations and also explore our [Blazor Chart Example](#) to know various chart types and how to represent time-dependent data, showing trends at equal intervals.

See Also

- [Data label](#)
- [Legend](#)
- [Marker](#)

Data Editing in Blazor Charts Component

Data editing allows the rendered points to be dragged and dropped at run-time. End user can adjust the points position or value based on its y-value. To enable data editing, set the [Enable](#) property in the [ChartDataEditSettings](#) to **true**. One can also use the [Fill](#) property to specify the color, and the [MinY](#) and [MaxY](#) properties to determine data editing's minimum and maximum range.

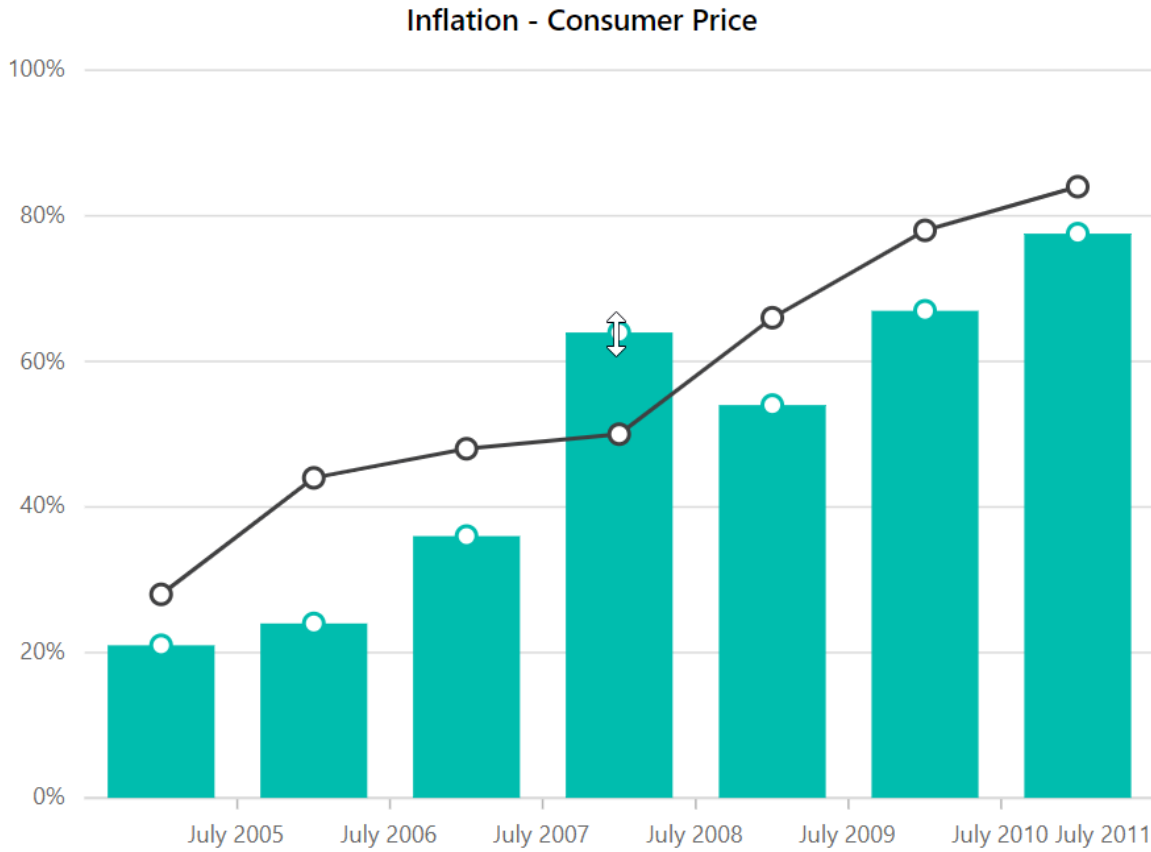
ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart Title="Inflation - Consumer Price">
<ChartArea><ChartAreaBorder Width="0"></ChartAreaBorder></ChartArea>
<ChartPrimaryXAxis ValueType="Syncfusion.Blazor.Charts.ValueType.DateTime"
LabelFormat="y"
IntervalType="IntervalType.Years"
EdgeLabelPlacement="EdgeLabelPlacement.Shift">
<ChartAxisMajorGridLines Width="0"></ChartAxisMajorGridLines>
```

```

</ChartPrimaryXAxis>
<ChartPrimaryYAxis LabelFormat="{value}%"
RangePadding="ChartRangePadding.None" Minimum="0" Maximum="100"
Interval="20">
<ChartAxisLineStyle Width="0"></ChartAxisLineStyle>
<ChartAxisMajorTickLines Width="0"></ChartAxisMajorTickLines>
</ChartPrimaryYAxis>
<ChartTooltipSettings Enable="true"></ChartTooltipSettings>
<ChartSeriesCollection>
<ChartSeries DataSource="@ConsumerDetails" XName="XValue" Width="2"
Opacity="1" YName="YValue" Type="ChartSeriesType.Column">
<ChartMarker Visible="true" Width="10" Height="10">
</ChartMarker>
<ChartDataEditSettings Enable="true"></ChartDataEditSettings>
</ChartSeries>
<ChartSeries DataSource="@ConsumerDetails" XName="XValue" Width="2"
Opacity="1" YName="YValue1" Type="ChartSeriesType.Line">
<ChartMarker Visible="true" Width="10" Height="10">
</ChartMarker>
<ChartDataEditSettings Enable="true"></ChartDataEditSettings>
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code{
public class ChartData
{
public DateTime XValue { get; set; }
public double YValue { get; set; }
public double YValue1 { get; set; }
}
public List<ChartData> ConsumerDetails = new List<ChartData>
{
new ChartData { XValue = new DateTime(2005, 01, 01), YValue = 21, YValue1 =
28 },
new ChartData { XValue = new DateTime(2006, 01, 01), YValue = 24, YValue1 =
44 },
new ChartData { XValue = new DateTime(2007, 01, 01), YValue = 36, YValue1 =
48 },
new ChartData { XValue = new DateTime(2008, 01, 01), YValue = 38, YValue1 =
50 },
new ChartData { XValue = new DateTime(2009, 01, 01), YValue = 54, YValue1 =
66 },
new ChartData { XValue = new DateTime(2010, 01, 01), YValue = 57, YValue1 =
78 },
new ChartData { XValue = new DateTime(2011, 01, 01), YValue = 70, YValue1 =
84 },
};
}

```



Refer to our [Blazor Charts](#) feature tour page for its groundbreaking feature representations and also explore our [Blazor Chart Example](#) to know various chart types and how to represent time-dependent data, showing trends at equal intervals.

See Also

- [Tooltip](#)
- [Legend](#)
- [Marker](#)

Crosshair and Trackball in Blazor Charts Component

Inspect or target any data point on mouse move or touch with the help of crosshair. A thin horizontal line and vertical line indicate the data point with the information displayed in an interactive tooltip. The crosshair can be enabled using the [Enable](#) property in the [ChartCrosshairSettings](#).

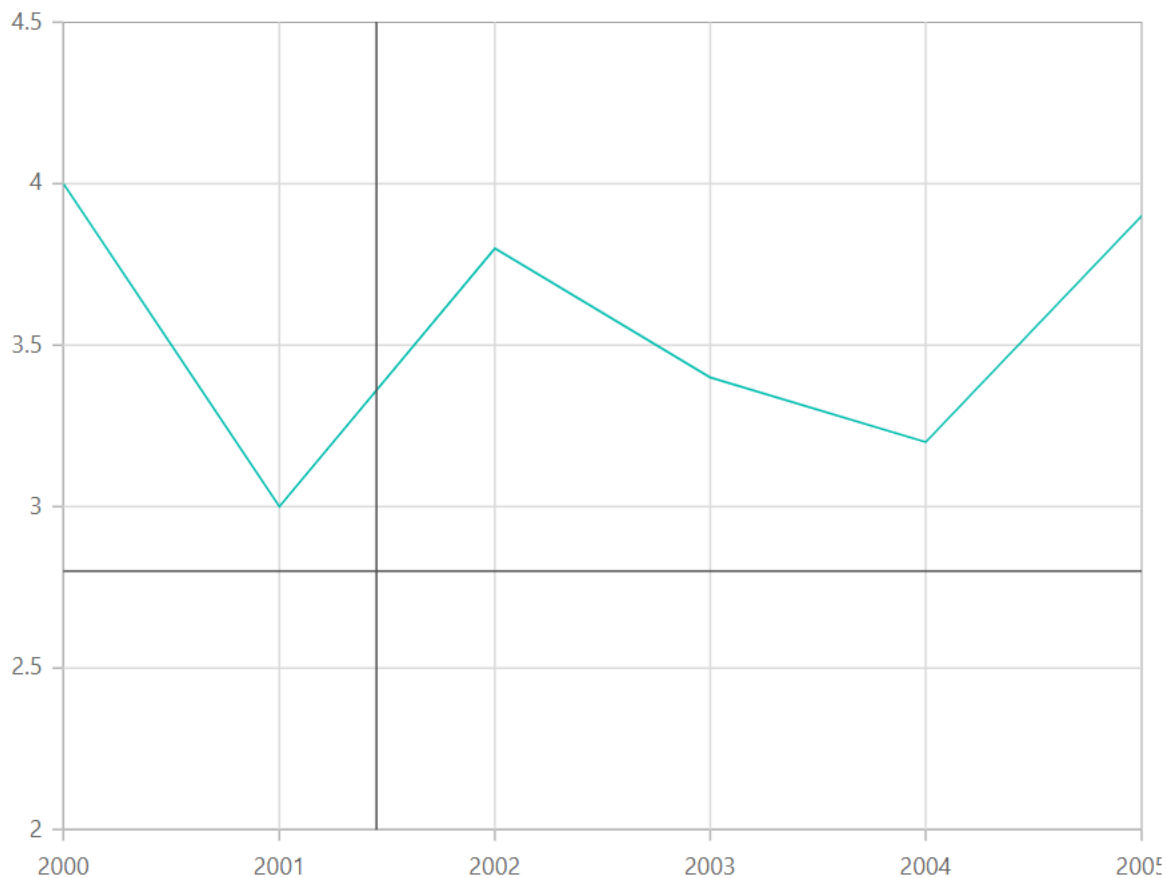
ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart>
<ChartPrimaryXAxis
Value="Type" YName="Y" DataSource="@SalesDetails"
Type="ChartSeriesType.Line">
<ChartCrosshairSettings Enable="true"></ChartCrosshairSettings>
<ChartSeriesCollection>
<ChartSeries XName="X" YName="Y" DataSource="@SalesDetails"
Type="ChartSeriesType.Line">
```

```

</ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code{
public class ChartData
{
public DateTime X { get; set; }
public double Y { get; set; }
}
public List<ChartData> SalesDetails = new List<ChartData>
{
new ChartData { X = new DateTime(2000, 01, 01), Y = 4 },
new ChartData { X = new DateTime(2001, 01, 01), Y = 3.0 },
new ChartData { X = new DateTime(2002, 01, 01), Y = 3.8 },
new ChartData { X = new DateTime(2003, 01, 01), Y = 3.4 },
new ChartData { X = new DateTime(2004, 01, 01), Y = 3.2 },
new ChartData { X = new DateTime(2005, 01, 01), Y = 3.9 },
};
}

```

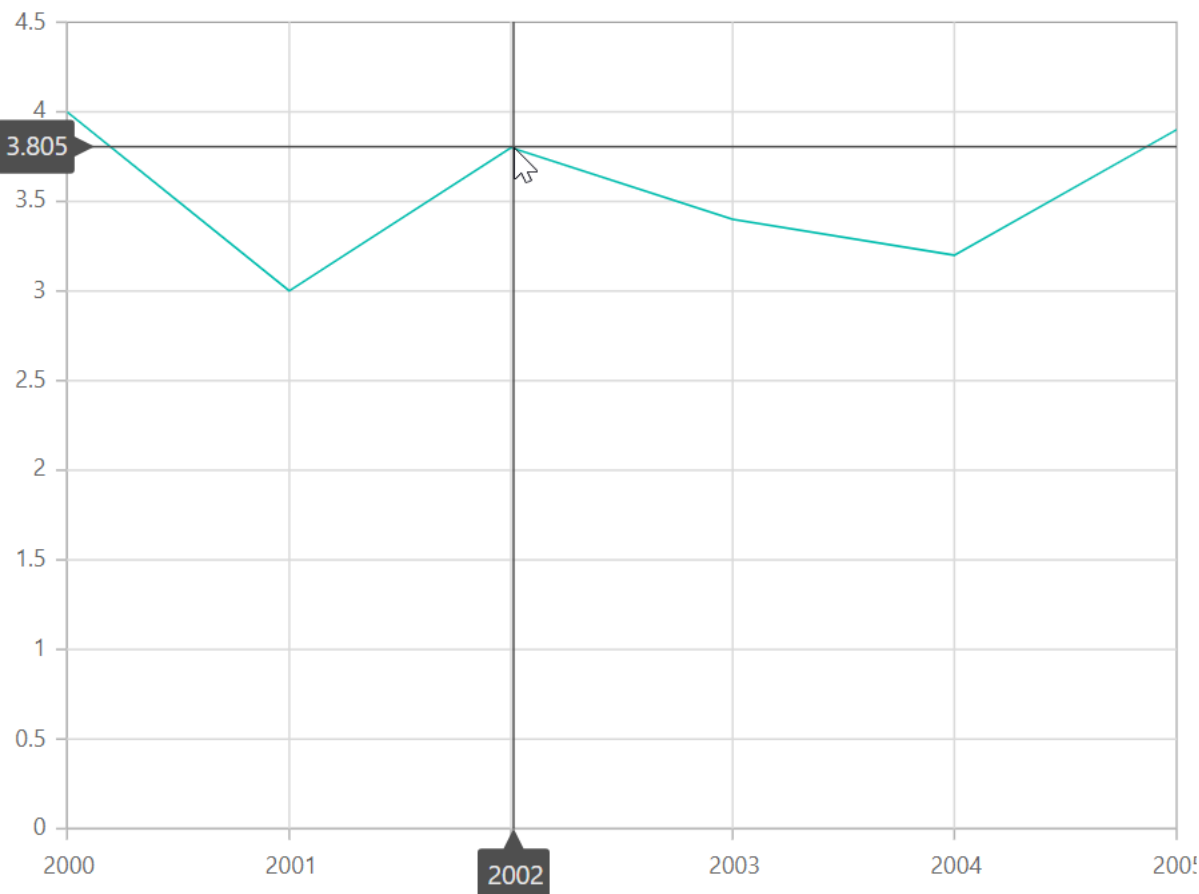


Enable Tooltip

The [Enable](#) property of [ChartAxisCrosshairTooltip](#) in the corresponding axis can be set to **true**, to enable the tooltip for that axis.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart>
<ChartPrimaryXAxis ValueType="Syncfusion.Blazor.Charts.ValueType.DateTime">
<ChartAxisCrosshairTooltip Enable="true"></ChartAxisCrosshairTooltip>
</ChartPrimaryXAxis>
<ChartCrosshairSettings Enable="true"></ChartCrosshairSettings>
<ChartPrimaryYAxis>
<ChartAxisCrosshairTooltip Enable="true"></ChartAxisCrosshairTooltip>
</ChartPrimaryYAxis>
<ChartSeriesCollection>
<ChartSeries DataSource="@SalesDetails" XName="X" YName="Y"
Type="ChartSeriesType.Line">
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code{
public class ChartData
{
public DateTime X { get; set; }
public double Y { get; set; }
}
public List<ChartData> SalesDetails = new List<ChartData>
{
new ChartData { X = new DateTime(2000, 01, 01), Y = 4 },
new ChartData { X = new DateTime(2001, 01, 01), Y = 3.0 },
new ChartData { X = new DateTime(2002, 01, 01), Y = 3.8 },
new ChartData { X = new DateTime(2003, 01, 01), Y = 3.4 },
new ChartData { X = new DateTime(2004, 01, 01), Y = 3.2 },
new ChartData { X = new DateTime(2005, 01, 01), Y = 3.9 },
};
}
```

Crosshair Customization

The [Fill](#) and [ChartCrosshairTextStyle](#) of the [ChartAxisCrosshairTooltip](#) is used to customize the background color and text style of the crosshair tooltip. The color and width of the crosshair line can be customized using the [ChartCrosshairLine](#) of [ChartCrosshairSettings](#).

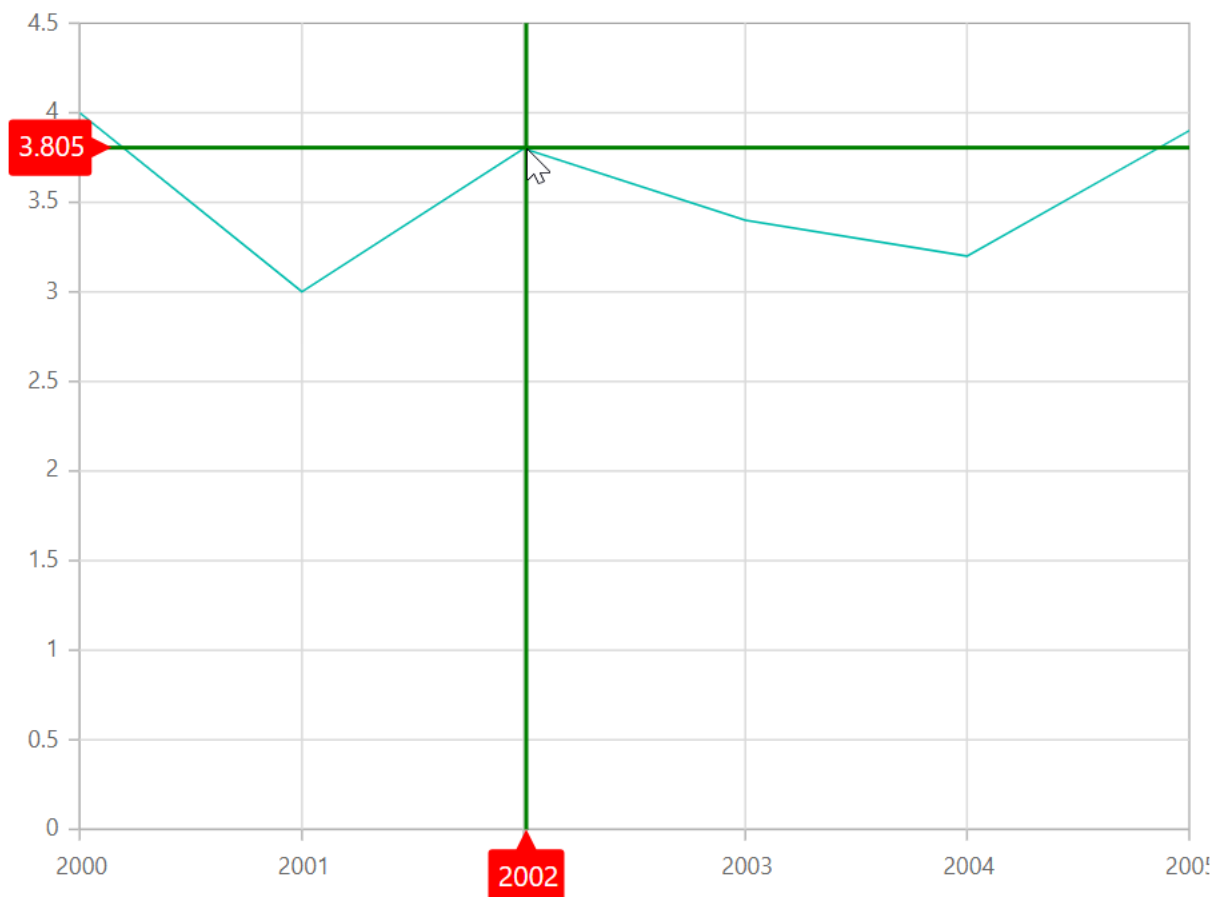
ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart>
<ChartPrimaryXAxis ValueTypes="Syncfusion.Blazor.Charts.ValueType.DateTime">
<ChartAxisCrosshairTooltip Enable="true" Fill="red">
<ChartCrosshairTextStyle Size="14px" Color="white">
</ChartCrosshairTextStyle>
</ChartAxisCrosshairTooltip>
</ChartPrimaryXAxis>
<ChartPrimaryYAxis>
<ChartAxisCrosshairTooltip Enable="true" Fill="red">
<ChartCrosshairTextStyle Size="14px" Color="white">
</ChartCrosshairTextStyle>
</ChartAxisCrosshairTooltip>
</ChartPrimaryYAxis>
<ChartCrosshairSettings Enable="true">
<ChartCrosshairLine Width="2" Color="green"></ChartCrosshairLine>
</ChartCrosshairSettings>
<ChartSeriesCollection>
```

```

<ChartSeries DataSource="@SalesDetails" XName="X" YName="Y"
Type="ChartSeriesType.Line">
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code{
public class ChartData
{
public DateTime X { get; set; }
public double Y { get; set; }
}
public List<ChartData> SalesDetails = new List<ChartData>
{
new ChartData { X = new DateTime(2000, 01, 01), Y = 4 },
new ChartData { X = new DateTime(2001, 01, 01), Y = 3.0 },
new ChartData { X = new DateTime(2002, 01, 01), Y = 3.8 },
new ChartData { X = new DateTime(2003, 01, 01), Y = 3.4 },
new ChartData { X = new DateTime(2004, 01, 01), Y = 3.2 },
new ChartData { X = new DateTime(2005, 01, 01), Y = 3.9 },
};
}

```



Trackball

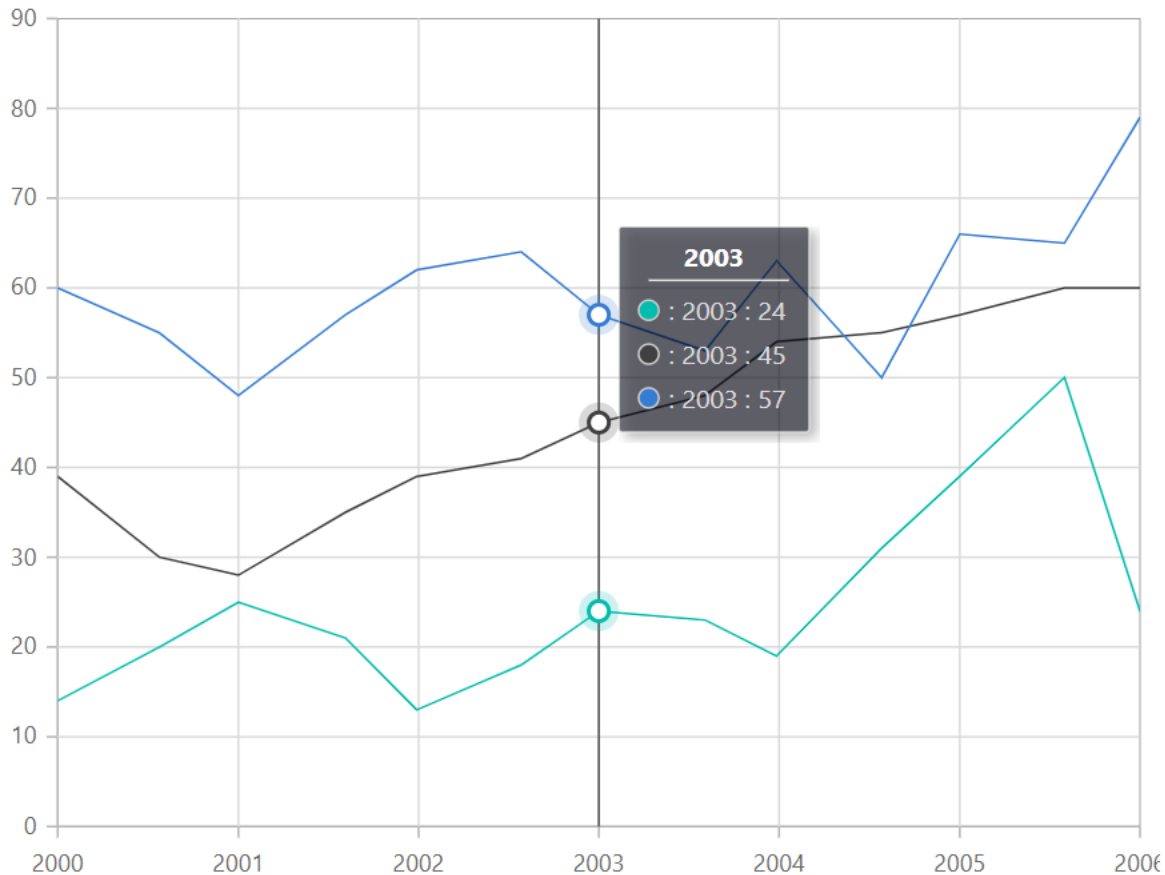
The trackball is used to track the data point that is closest to the mouse or touch position. The closest point is indicated by a trackball marker, and the information about the point is displayed via a trackball

tooltip. Trackball can be enabled by setting the [Enable](#) property of the [ChartCrosshairSettings](#) to **true** and [Shared](#) property in [Tooltip](#) to **true** in the chart.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart>
<ChartPrimaryXAxis ValueType="Syncfusion.Blazor.Charts.ValueType.DateTime">
</ChartPrimaryXAxis>
<ChartCrosshairSettings Enable="true"
LineType="LineType.Vertical"></ChartCrosshairSettings>
<ChartTooltipSettings Enable="true" Shared="true" Format="{series.name} :
${point.x} : ${point.y}"></ChartTooltipSettings>
<ChartSeriesCollection>
<ChartSeries DataSource="@SalesDetails" XName="XValue" YName="YValue"
Type="ChartSeriesType.Line">
</ChartSeries>
<ChartSeries DataSource="@SalesDetails" XName="XValue" YName="YValue1"
Type="ChartSeriesType.Line">
</ChartSeries>
<ChartSeries DataSource="@SalesDetails" XName="XValue" YName="YValue2"
Type="ChartSeriesType.Line">
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code{
public class ChartData
{
public DateTime XValue { get; set; }
public double YValue { get; set; }
public double YValue1 { get; set; }
public double YValue2 { get; set; }
}
public List<ChartData> SalesDetails = new List<ChartData>
{
new ChartData { XValue = new DateTime(2000, 2, 11), YValue = 14, YValue1 =
39, YValue2 = 60 },
new ChartData { XValue = new DateTime(2000, 9, 4), YValue = 20, YValue1 =
30, YValue2 = 55 },
new ChartData { XValue = new DateTime(2001, 2, 11), YValue = 25, YValue1 =
28, YValue2 = 48 },
new ChartData { XValue = new DateTime(2001, 9, 16), YValue = 21, YValue1 =
35, YValue2 = 57 },
new ChartData { XValue = new DateTime(2002, 2, 7), YValue = 13, YValue1 =
39, YValue2 = 62 },
new ChartData { XValue = new DateTime(2002, 9, 7), YValue = 18, YValue1 =
41, YValue2 = 64 },
new ChartData { XValue = new DateTime(2003, 2, 11), YValue = 24, YValue1 =
45, YValue2 = 57 },
new ChartData { XValue = new DateTime(2003, 9, 14), YValue = 23, YValue1 =
48, YValue2 = 53 },
new ChartData { XValue = new DateTime(2004, 2, 6), YValue = 19, YValue1 =
54, YValue2 = 63 },
new ChartData { XValue = new DateTime(2004, 9, 6), YValue = 31, YValue1 =
55, YValue2 = 50 },
new ChartData { XValue = new DateTime(2005, 2, 11), YValue = 39, YValue1 =
57, YValue2 = 66 },
```

```
new ChartData { XValue = new DateTime(2005, 9, 11), YValue = 50, YValue1 = 60, YValue2 = 65 },
new ChartData { XValue = new DateTime(2006, 2, 11), YValue = 24, YValue1 = 60, YValue2 = 79 },
};
}
```



Refer to our [Blazor Charts](#) feature tour page for its groundbreaking feature representations and also explore our [Blazor Chart Example](#) to know various chart types and how to represent time-dependent data, showing trends at equal intervals.

See Also

- [Data Label](#)
- [Tooltip](#)
- [Marker](#)

<!-- markdownlint-disable MD036 -->

Selection in Blazor Charts Component

The chart provides selection support for the series and its data points on mouse or touch action.

When clicked on the data points, the corresponding series legend will also be selected.

The chart offers a variety of selection mode for selecting the data. They are,

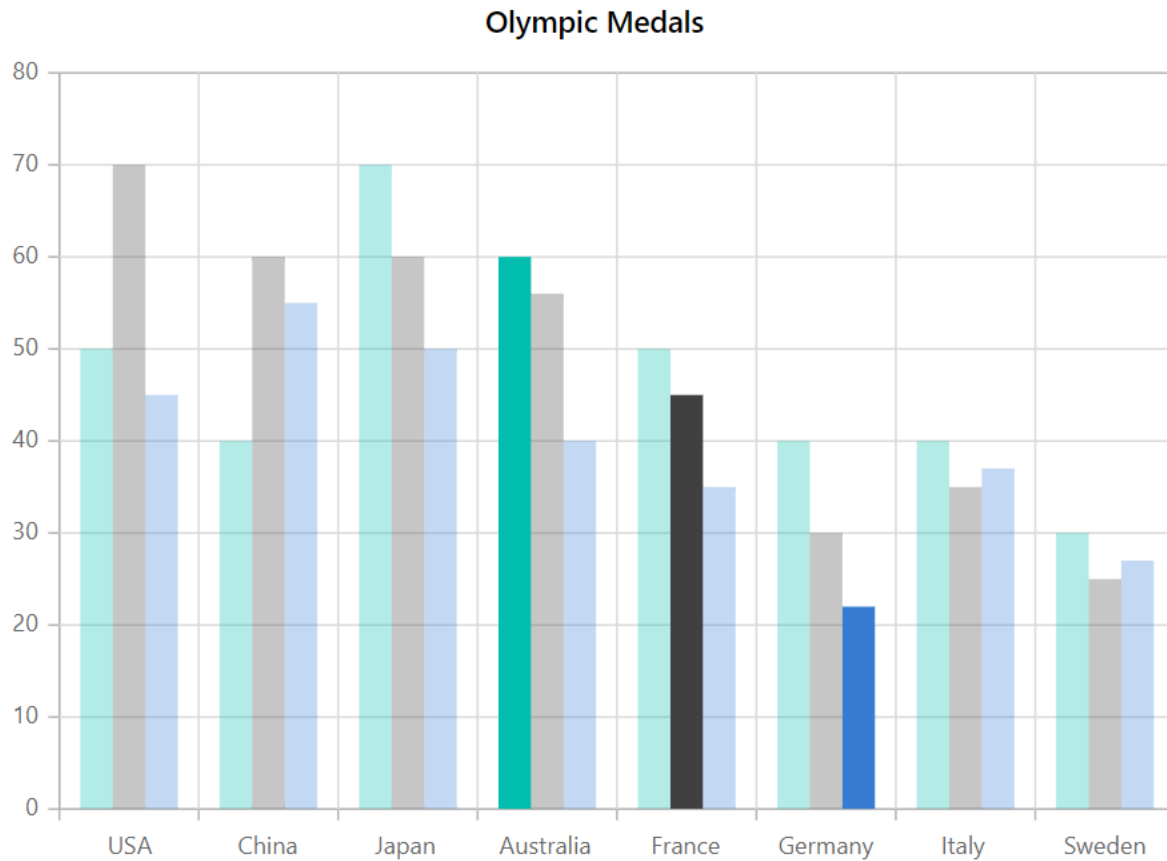
- None
- Point
- Series
- Cluster
- DragXY
- DragX
- DragY

Point

When the [SelectionMode](#) property is set to **Point**, it allows to select a single point.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart Title="Olympic Medals" SelectionMode="SelectionMode.Point">
<ChartPrimaryXAxis ValueType="Syncfusion.Blazor.Charts.ValueType.Category">
</ChartPrimaryXAxis>
<ChartSeriesCollection>
<ChartSeries DataSource="@MedalDetails" XName="Country" YName="Gold"
Type="ChartSeriesType.Column">
</ChartSeries>
<ChartSeries DataSource="@MedalDetails" XName="Country" YName="Silver"
Type="ChartSeriesType.Column">
</ChartSeries>
<ChartSeries DataSource="@MedalDetails" XName="Country" YName="Bronze"
Type="ChartSeriesType.Column">
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code{
public class ChartData
{
public string Country { get; set; }
public double Gold { get; set; }
public double Silver { get; set; }
public double Bronze { get; set; }
}
public List<ChartData> MedalDetails = new List<ChartData>
{
new ChartData{ Country= "USA", Gold=50, Silver=70, Bronze=45 },
new ChartData{ Country="China", Gold=40, Silver= 60, Bronze=55 },
new ChartData{ Country= "Japan", Gold=70, Silver= 60, Bronze=50 },
new ChartData{ Country= "Australia", Gold=60, Silver= 56, Bronze=40 },
new ChartData{ Country= "France", Gold=50, Silver= 45, Bronze=35 },
new ChartData{ Country= "Germany", Gold=40, Silver=30, Bronze=22 },
new ChartData{ Country= "Italy", Gold=40, Silver=35, Bronze=37 },
new ChartData{ Country= "Sweden", Gold=30, Silver=25, Bronze=27 }
};
}
```



Series

When the [SelectionMode](#) property is set to **Series**, it allows to select a series.

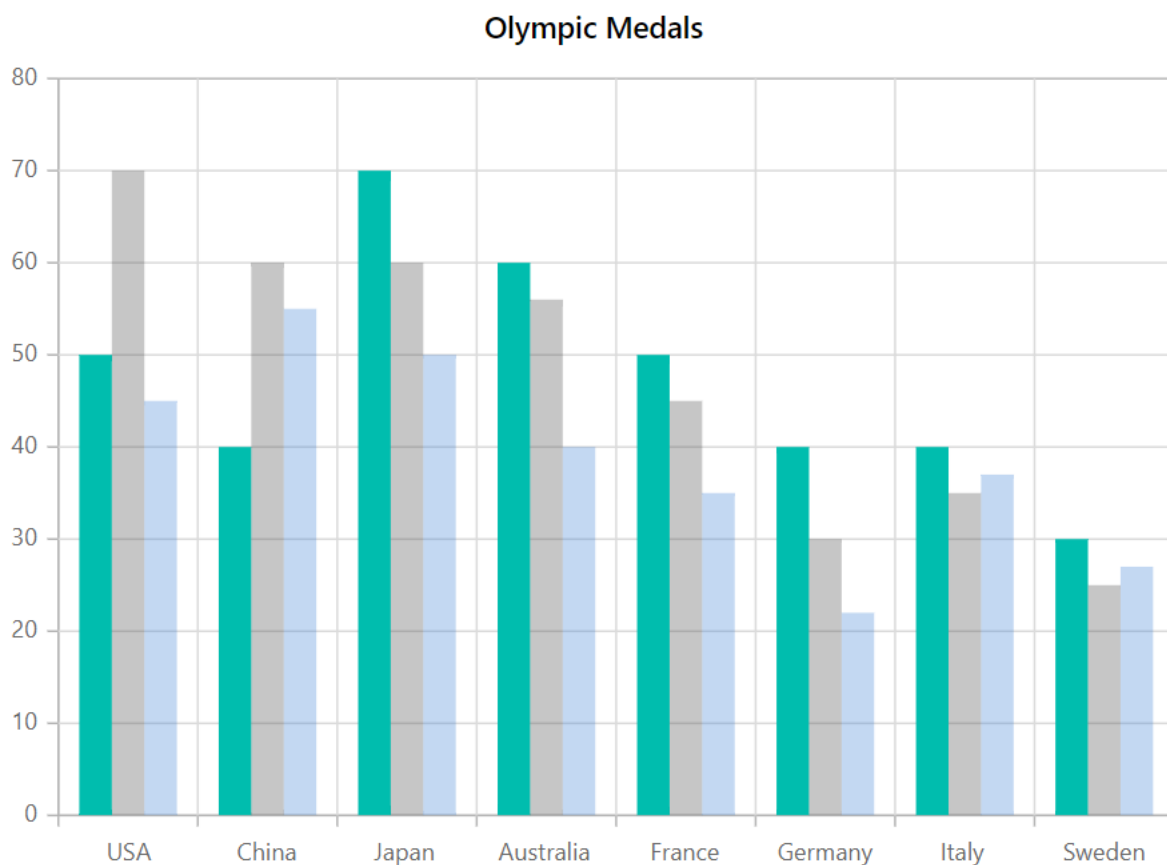
ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart Title="Olympic Medals" SelectionMode="SelectionMode.Series">
  <ChartPrimaryXAxis ValueTypes="Syncfusion.Blazor.Charts.ValueType.Category">
  </ChartPrimaryXAxis>
  <ChartSeriesCollection>
    <ChartSeries DataSource="@MedalDetails" XName="Country" YName="Gold"
    Type="ChartSeriesType.Column">
    </ChartSeries>
    <ChartSeries DataSource="@MedalDetails" XName="Country" YName="Silver"
    Type="ChartSeriesType.Column">
    </ChartSeries>
    <ChartSeries DataSource="@MedalDetails" XName="Country" YName="Bronze"
    Type="ChartSeriesType.Column">
    </ChartSeries>
  </ChartSeriesCollection>
</SfChart>
@code{
public class ChartData
{
    public string Country { get; set; }
    public double Gold { get; set; }
    public double Silver { get; set; }
}
```

```

public double Bronze { get; set; }
}
public List<ChartData> MedalDetails = new List<ChartData>
{
    new ChartData{ Country= "USA", Gold=50, Silver=70, Bronze=45 },
    new ChartData{ Country="China", Gold=40, Silver= 60, Bronze=55 },
    new ChartData{ Country= "Japan", Gold=70, Silver= 60, Bronze=50 },
    new ChartData{ Country= "Australia", Gold=60, Silver= 56, Bronze=40 },
    new ChartData{ Country= "France", Gold=50, Silver= 45, Bronze=35 },
    new ChartData{ Country= "Germany", Gold=40, Silver=30, Bronze=22 },
    new ChartData{ Country= "Italy", Gold=40, Silver=35, Bronze=37 },
    new ChartData{ Country= "Sweden", Gold=30, Silver=25, Bronze=27 }
};
}

```



Cluster

By setting the [SelectionMode](#) property to **Cluster**, one can select the points in all series that correspond to the same index.

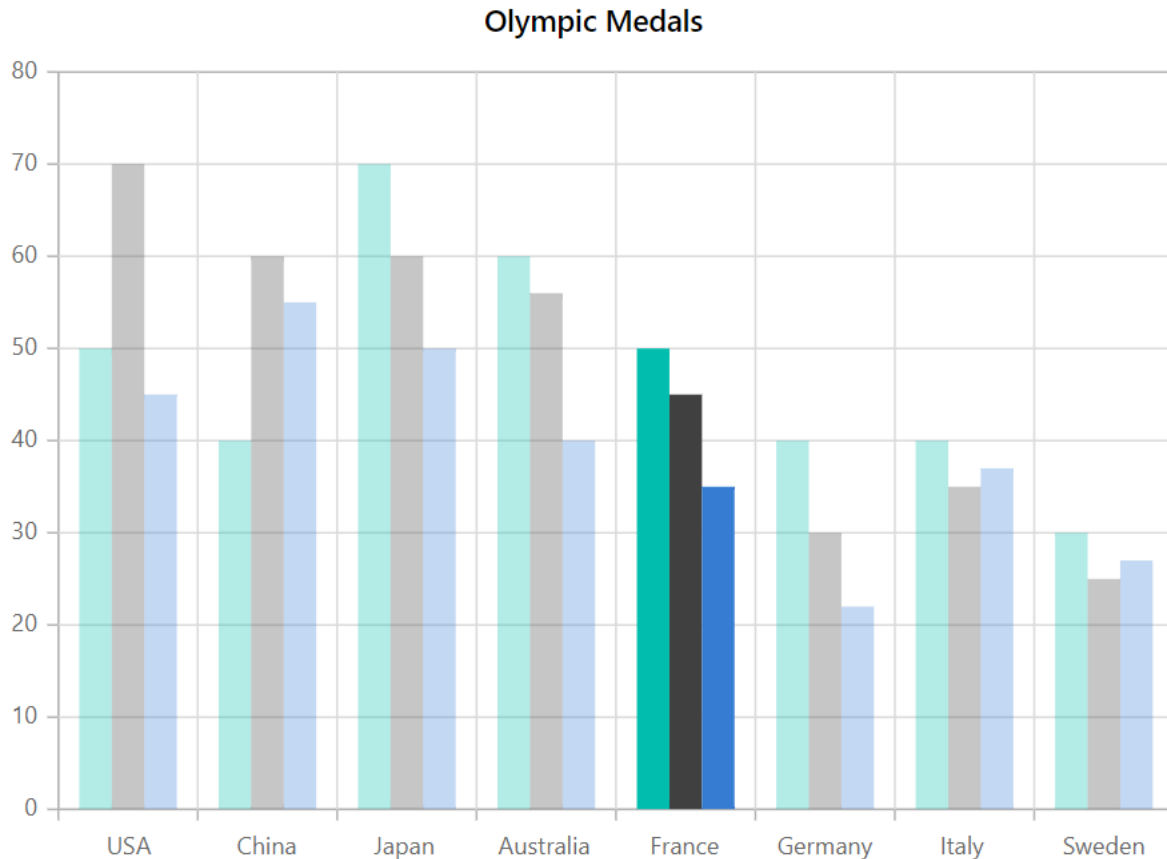
ASPX-CS

```

@using Syncfusion.Blazor.Charts
<SfChart Title="Olympic Medals" SelectionMode="SelectionMode.Cluster">
  <ChartPrimaryXAxis ValueType="Syncfusion.Blazor.Charts.ValueType.Category">
  </ChartPrimaryXAxis>
  <ChartSeriesCollection>

```

```
<ChartSeries DataSource="@MedalDetails" XName="Country" YName="Gold"
Type="ChartSeriesType.Column">
</ChartSeries>
<ChartSeries DataSource="@MedalDetails" XName="Country" YName="Silver"
Type="ChartSeriesType.Column">
</ChartSeries>
<ChartSeries DataSource="@MedalDetails" XName="Country" YName="Bronze"
Type="ChartSeriesType.Column">
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code{
public class ChartData
{
public string Country { get; set; }
public double Gold { get; set; }
public double Silver { get; set; }
public double Bronze { get; set; }
}
public List<ChartData> MedalDetails = new List<ChartData>
{
new ChartData{ Country= "USA", Gold=50, Silver=70, Bronze=45 },
new ChartData{ Country="China", Gold=40, Silver= 60, Bronze=55 },
new ChartData{ Country= "Japan", Gold=70, Silver= 60, Bronze=50 },
new ChartData{ Country= "Australia", Gold=60, Silver= 56, Bronze=40 },
new ChartData{ Country= "France", Gold=50, Silver= 45, Bronze=35 },
new ChartData{ Country= "Germany", Gold=40, Silver=30, Bronze=22 },
new ChartData{ Country= "Italy", Gold=40, Silver=35, Bronze=37 },
new ChartData{ Country= "Sweden", Gold=30, Silver=25, Bronze=27 }
};
}
```

Rectangular Selection

DragXY, DragX and DragY

Set [SelectionMode](#) to **DragXY** to retrieve a group of data under a specific region.

- DragXY - Allows to select data with respect to horizontal and vertical axis.
- DragX - Allows to select data with respect to horizontal axis.
- DragY - Allows to select data with respect to vertical axis.

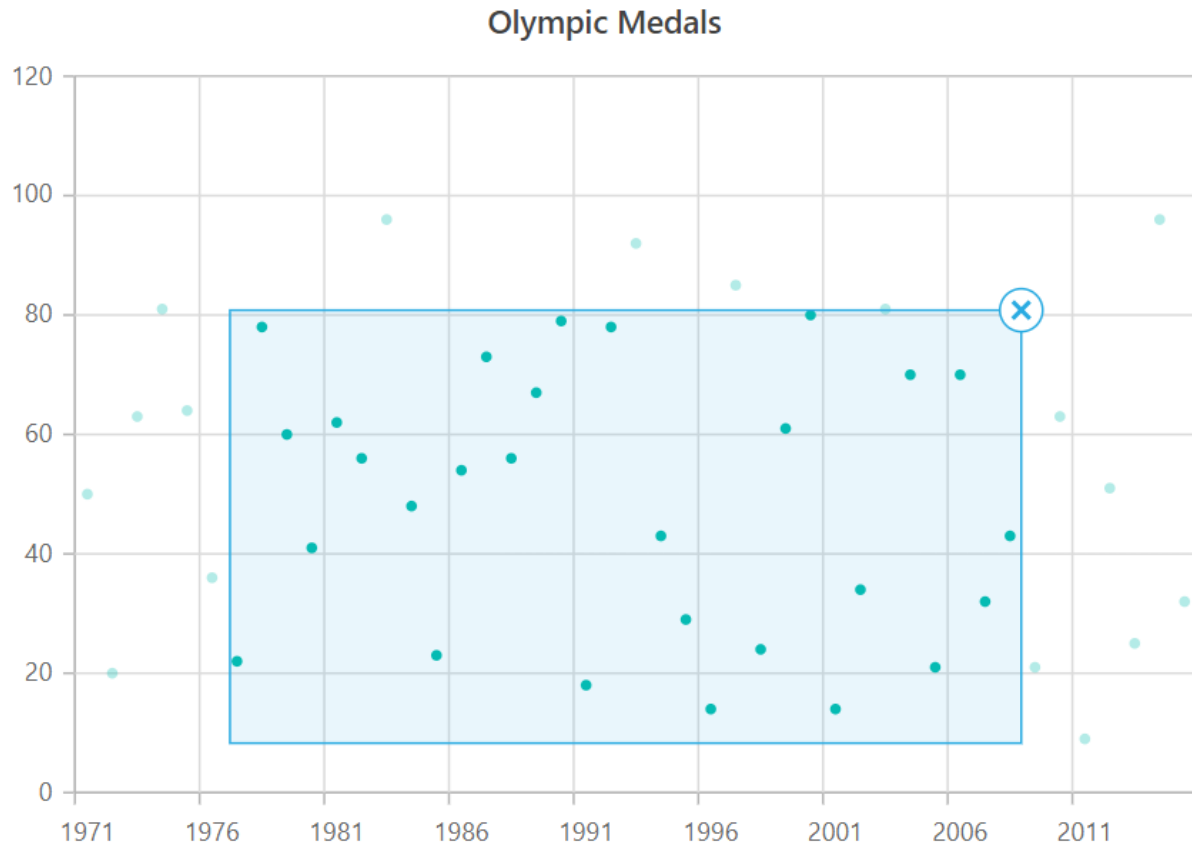
In the drag complete event, the selected data will be returned as an array collection.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart Title="Olympic Medals" SelectionMode="SelectionMode.DragXY">
  <ChartPrimaryXAxis ValueTypes="Syncfusion.Blazor.Charts.ValueType.Category">
  </ChartPrimaryXAxis>
  <ChartSeriesCollection>
    <ChartSeries DataSource="@MedalDetails" XName="XValue" Opacity="1"
      YName="YValue" Type="ChartSeriesType.Scatter">
    </ChartSeries>
  </ChartSeriesCollection>
</SfChart>
@code{
public class ChartData
{

```

```
public double XValue { get; set; }
public double YValue { get; set; }
};
public List<ChartData> MedalDetails = new List<ChartData>
{
    new ChartData { XValue = 1971, YValue = 50 },
    new ChartData { XValue = 1972, YValue = 20 },
    new ChartData { XValue = 1973, YValue = 63 },
    new ChartData { XValue = 1974, YValue = 81 },
    new ChartData { XValue = 1975, YValue = 64 },
    new ChartData { XValue = 1976, YValue = 36 },
    new ChartData { XValue = 1977, YValue = 22 },
    new ChartData { XValue = 1978, YValue = 78 },
    new ChartData { XValue = 1979, YValue = 60 },
    new ChartData { XValue = 1980, YValue = 41 },
    new ChartData { XValue = 1981, YValue = 62 },
    new ChartData { XValue = 1982, YValue = 56 },
    new ChartData { XValue = 1983, YValue = 96 },
    new ChartData { XValue = 1984, YValue = 48 },
    new ChartData { XValue = 1985, YValue = 23 },
    new ChartData { XValue = 1986, YValue = 54 },
    new ChartData { XValue = 1987, YValue = 73 },
    new ChartData { XValue = 1988, YValue = 56 },
    new ChartData { XValue = 1989, YValue = 67 },
    new ChartData { XValue = 1990, YValue = 79 },
    new ChartData { XValue = 1991, YValue = 18 },
    new ChartData { XValue = 1992, YValue = 78 },
    new ChartData { XValue = 1993, YValue = 92 },
    new ChartData { XValue = 1994, YValue = 43 },
    new ChartData { XValue = 1995, YValue = 29 },
    new ChartData { XValue = 1996, YValue = 14 },
    new ChartData { XValue = 1997, YValue = 85 },
    new ChartData { XValue = 1998, YValue = 24 },
    new ChartData { XValue = 1999, YValue = 61 },
    new ChartData { XValue = 2000, YValue = 80 },
    new ChartData { XValue = 2001, YValue = 14 },
    new ChartData { XValue = 2002, YValue = 34 },
    new ChartData { XValue = 2003, YValue = 81 },
    new ChartData { XValue = 2004, YValue = 70 },
    new ChartData { XValue = 2005, YValue = 21 },
    new ChartData { XValue = 2006, YValue = 70 },
    new ChartData { XValue = 2007, YValue = 32 },
    new ChartData { XValue = 2008, YValue = 43 },
    new ChartData { XValue = 2009, YValue = 21 },
    new ChartData { XValue = 2010, YValue = 63 },
    new ChartData { XValue = 2011, YValue = 9 },
    new ChartData { XValue = 2012, YValue = 51 },
    new ChartData { XValue = 2013, YValue = 25 },
    new ChartData { XValue = 2014, YValue = 96 },
    new ChartData { XValue = 2015, YValue = 32 }
};
}
```



Multiple Selection

Multiple points or series can be selected by setting the [IsMultiSelect](#) property to **true**.

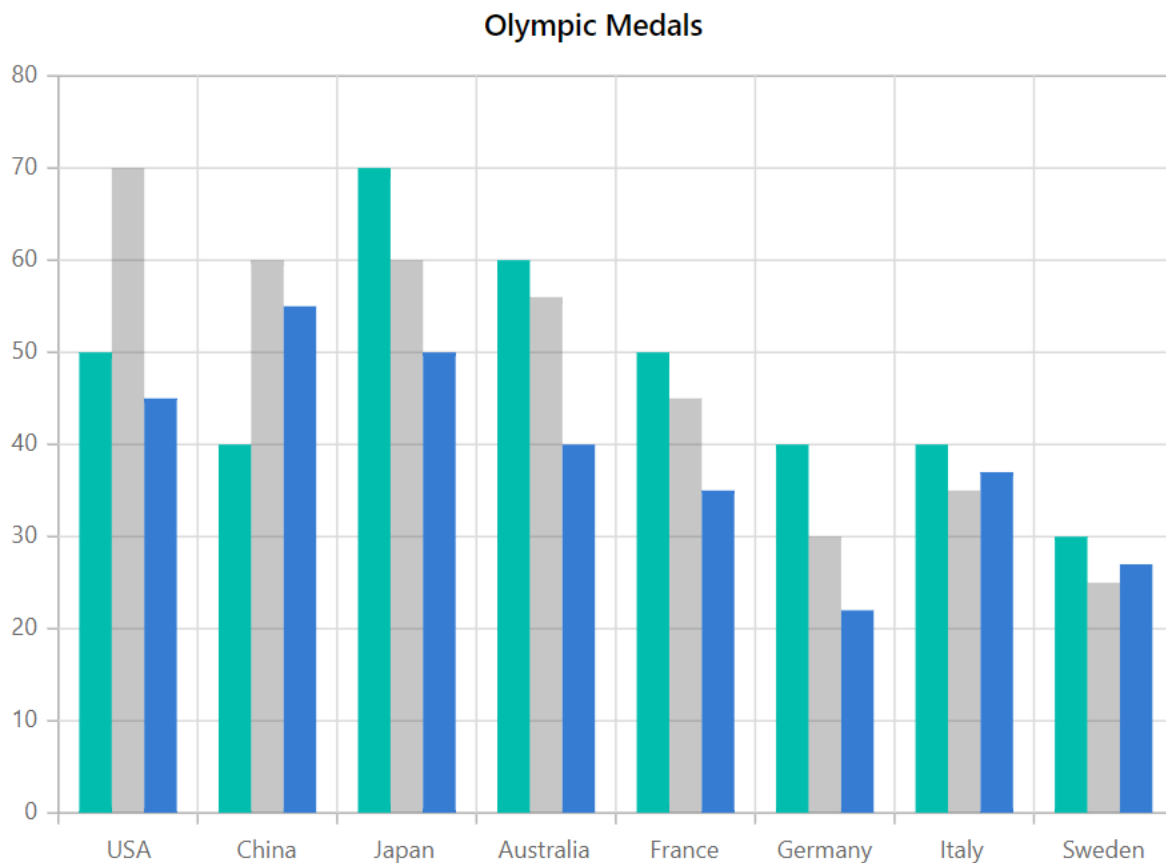
ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart Title="Olympic Medals" SelectionMode="SelectionMode.Series"
IsMultiSelect="true">
<ChartPrimaryXAxis ValueType="Syncfusion.Blazor.Charts.ValueType.Category">
</ChartPrimaryXAxis>
<ChartSeriesCollection>
<ChartSeries DataSource="@MedalDetails" XName="Country" YName="Gold"
Type="ChartSeriesType.Column">
</ChartSeries>
<ChartSeries DataSource="@MedalDetails" XName="Country" YName="Silver"
Type="ChartSeriesType.Column">
</ChartSeries>
<ChartSeries DataSource="@MedalDetails" XName="Country" YName="Bronze"
Type="ChartSeriesType.Column">
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code{
public class ChartData
{
public string Country { get; set; }
public double Gold { get; set; }
}
```

```

public double Silver { get; set; }
public double Bronze { get; set; }
}
public List<ChartData> MedalDetails = new List<ChartData>
{
    new ChartData{ Country= "USA", Gold=50, Silver=70, Bronze=45 },
    new ChartData{ Country="China", Gold=40, Silver= 60, Bronze=55 },
    new ChartData{ Country= "Japan", Gold=70, Silver= 60, Bronze=50 },
    new ChartData{ Country= "Australia", Gold=60, Silver= 56, Bronze=40 },
    new ChartData{ Country= "France", Gold=50, Silver= 45, Bronze=35 },
    new ChartData{ Country= "Germany", Gold=40, Silver=30, Bronze=22 },
    new ChartData{ Country= "Italy", Gold=40, Silver=35, Bronze=37 },
    new ChartData{ Country= "Sweden", Gold=30, Silver=25, Bronze=27 }
};
}

```



Selection via code-behind

A point or series can be selected programmatically on a chart using the [SelectedDataIndexes](#) property.

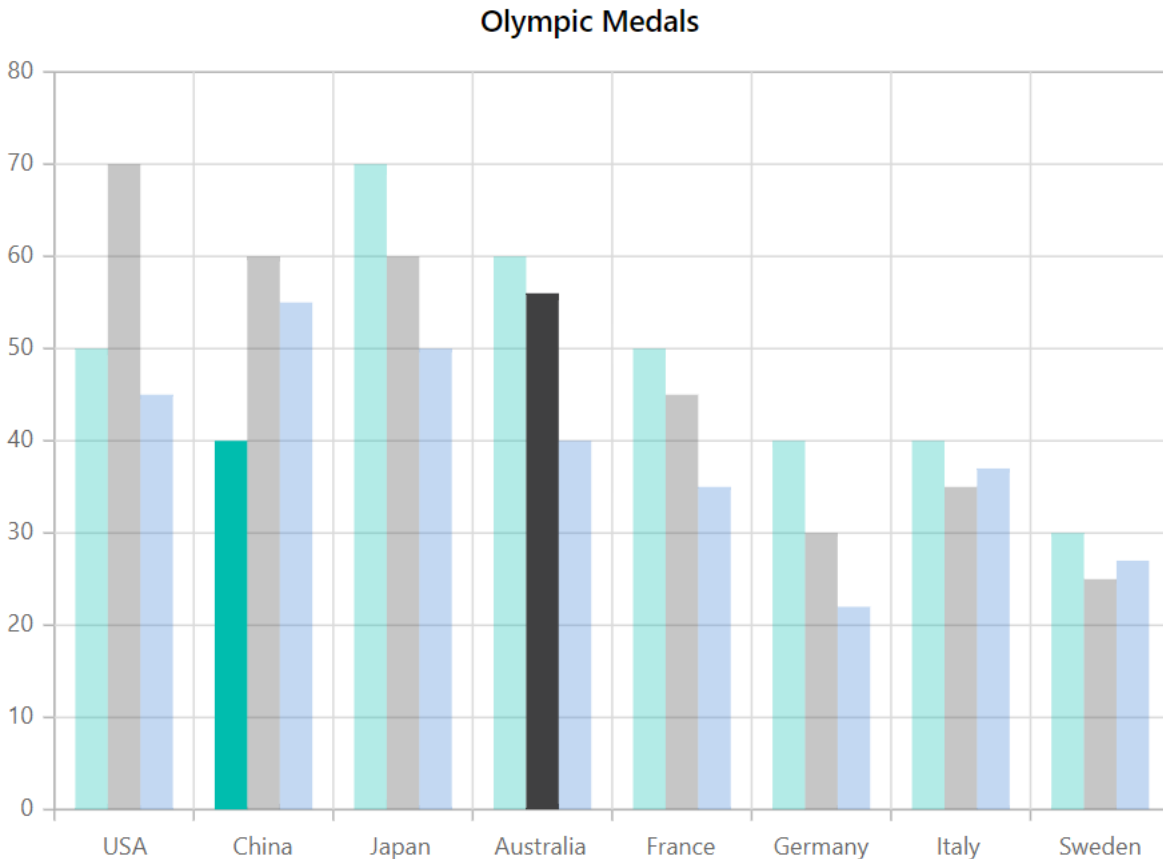
ASPX-CS

```

@using Syncfusion.Blazor.Charts
<SfChart Title="Olympic Medals" SelectionMode="SelectionMode.Point"
IsMultiSelect="true">
<ChartPrimaryXAxis ValueTypes="Syncfusion.Blazor.Charts.ValueType.Category">
</ChartPrimaryXAxis>

```

```
<ChartSelectedDataIndexes>
<ChartSelectedDataIndex Series="0" Point="1">
</ChartSelectedDataIndex>
<ChartSelectedDataIndex Series="1" Point="3">
</ChartSelectedDataIndex>
</ChartSelectedDataIndexes>
<ChartSeriesCollection>
<ChartSeries DataSource="@MedalDetails" XName="Country" YName="Gold"
Type="ChartSeriesType.Column">
</ChartSeries>
<ChartSeries DataSource="@MedalDetails" XName="Country" YName="Silver"
Type="ChartSeriesType.Column">
</ChartSeries>
<ChartSeries DataSource="@MedalDetails" XName="Country" YName="Bronze"
Type="ChartSeriesType.Column">
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code{
public class ChartData
{
public string Country { get; set; }
public double Gold { get; set; }
public double Silver { get; set; }
public double Bronze { get; set; }
}
public List<ChartData> MedalDetails = new List<ChartData>
{
new ChartData{ Country= "USA", Gold=50, Silver=70, Bronze=45 },
new ChartData{ Country="China", Gold=40, Silver= 60, Bronze=55 },
new ChartData{ Country= "Japan", Gold=70, Silver= 60, Bronze=50 },
new ChartData{ Country= "Australia", Gold=60, Silver= 56, Bronze=40 },
new ChartData{ Country= "France", Gold=50, Silver= 45, Bronze=35 },
new ChartData{ Country= "Germany", Gold=40, Silver=30, Bronze=22 },
new ChartData{ Country= "Italy", Gold=40, Silver=35, Bronze=37 },
new ChartData{ Country= "Sweden", Gold=30, Silver=25, Bronze=27 }
};
}
```



Legend Selection

A point or series can be selected through legend using the [ToggleVisibility](#) property.

ASPX-CS

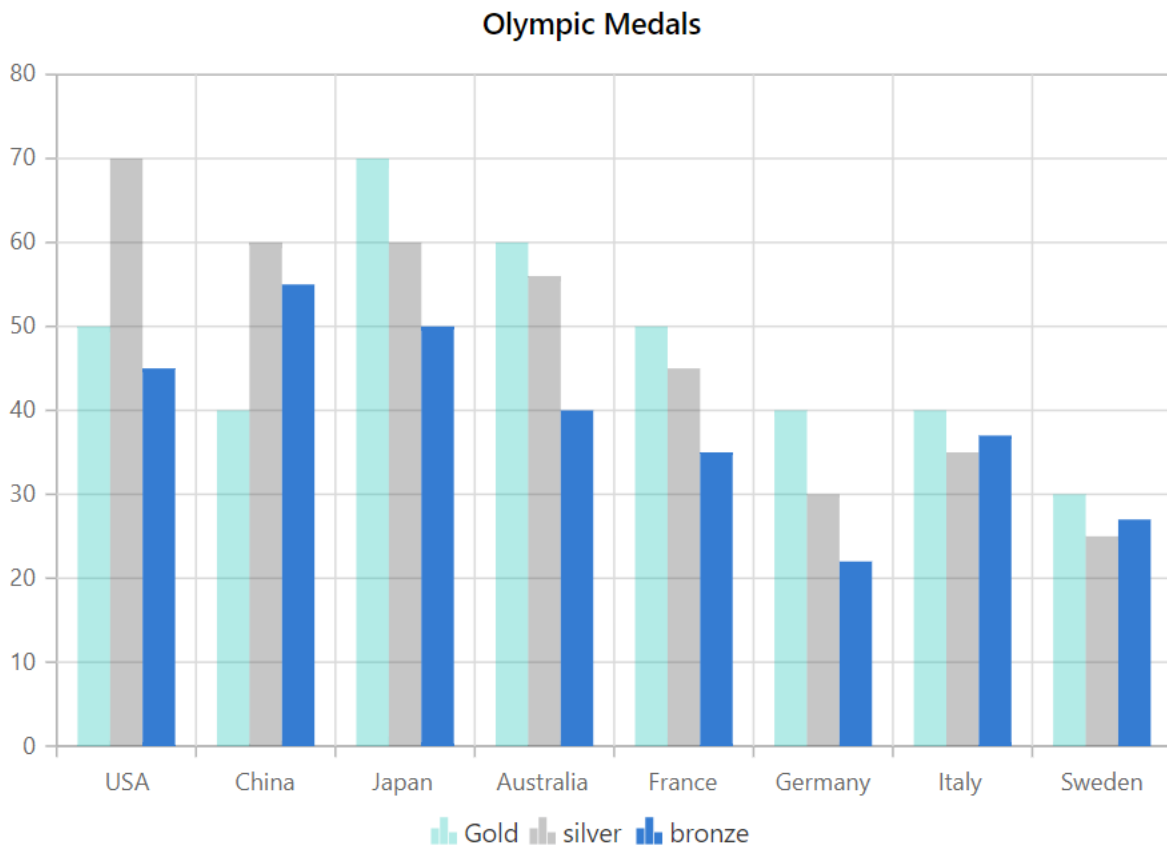
```
@using Syncfusion.Blazor.Charts
<SfChart Title="Olympic Medals" SelectionMode="SelectionMode.Point">
  <ChartPrimaryXAxis ValueTypes="Syncfusion.Blazor.Charts.ValueType.Category">
  </ChartPrimaryXAxis>
  <ChartSeriesCollection>
    <ChartSeries DataSource="@MedalDetails" Name="Gold" XName="Country"
      YName="Gold" Type="ChartSeriesType.Column">
    </ChartSeries>
    <ChartSeries DataSource="@MedalDetails" Name="silver" XName="Country"
      YName="Silver" Type="ChartSeriesType.Column">
    </ChartSeries>
    <ChartSeries DataSource="@MedalDetails" Name="bronze" XName="Country"
      YName="Bronze" Type="ChartSeriesType.Column">
    </ChartSeries>
  </ChartSeriesCollection>
  <ChartLegendSettings Visible="true" ToggleVisibility="false">
  </ChartLegendSettings>
</SfChart>

@code{
public class ChartData
{
    public string Country { get; set; }
}
```

```

public double Gold { get; set; }
public double Silver { get; set; }
public double Bronze { get; set; }
}
public List<ChartData> MedalDetails = new List<ChartData>
{
    new ChartData{ Country= "USA", Gold=50, Silver=70, Bronze=45 },
    new ChartData{ Country="China", Gold=40, Silver= 60, Bronze=55 },
    new ChartData{ Country= "Japan", Gold=70, Silver= 60, Bronze=50 },
    new ChartData{ Country= "Australia", Gold=60, Silver= 56, Bronze=40 },
    new ChartData{ Country= "France", Gold=50, Silver= 45, Bronze=35 },
    new ChartData{ Country= "Germany", Gold=40, Silver=30, Bronze=22 },
    new ChartData{ Country= "Italy", Gold=40, Silver=35, Bronze=37 },
    new ChartData{ Country= "Sweden", Gold=30, Silver=25, Bronze=27 }
};
}

```



Selection Customization

The custom style can be applied to selected points or series using the [SelectionStyle](#) property.

ASPX-CS

```

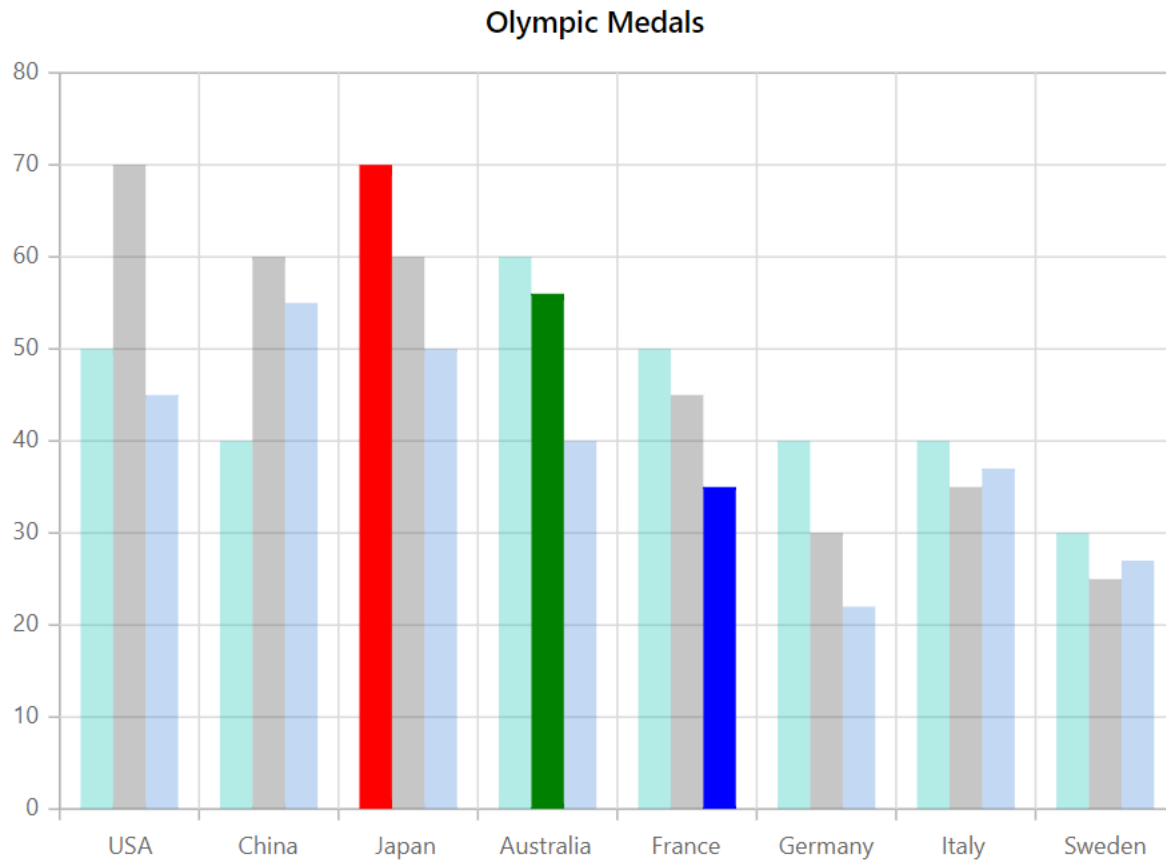
@using Syncfusion.Blazor.Charts
<SfChart Title="Olympic Medals" SelectionMode="SelectionMode.Point">
  <ChartPrimaryXAxis ValueType="Syncfusion.Blazor.Charts.ValueType.Category">
  </ChartPrimaryXAxis>
  <ChartLegendSettings Visible="true" ToggleVisibility="true">

```

```

</ChartLegendSettings>
<ChartSeriesCollection>
<ChartSeries DataSource="@MedalDetails" XName="Country" YName="Gold"
Type="ChartSeriesType.Column" SelectionStyle="chartSelection1">
</ChartSeries>
<ChartSeries DataSource="@MedalDetails" XName="Country" YName="Silver"
Type="ChartSeriesType.Column" SelectionStyle="chartSelection2">
</ChartSeries>
<ChartSeries DataSource="@MedalDetails" XName="Country" YName="Bronze"
Type="ChartSeriesType.Column" SelectionStyle="chartSelection3">
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
<style>
.chartSelection1 {
fill: red
}
.chartSelection2 {
fill: green
}
.chartSelection3 {
fill: blue
}
</style>
@code{
public class ChartData
{
public string Country { get; set; }
public double Gold { get; set; }
public double Silver { get; set; }
public double Bronze { get; set; }
}
public List<ChartData> MedalDetails = new List<ChartData>
{
new ChartData{ Country= "USA", Gold=50, Silver=70, Bronze=45 },
new ChartData{ Country="China", Gold=40, Silver= 60, Bronze=55 },
new ChartData{ Country= "Japan", Gold=70, Silver= 60, Bronze=50 },
new ChartData{ Country= "Australia", Gold=60, Silver= 56, Bronze=40 },
new ChartData{ Country= "France", Gold=50, Silver= 45, Bronze=35 },
new ChartData{ Country= "Germany", Gold=40, Silver=30, Bronze=22 },
new ChartData{ Country= "Italy", Gold=40, Silver=35, Bronze=37 },
new ChartData{ Country= "Sweden", Gold=30, Silver=25, Bronze=27 }
};
}

```

Refer to our [Blazor Charts](#) feature tour page for its groundbreaking feature representations and also explore our [Blazor Chart Example](#) to know various chart types and how to represent time-dependent data, showing trends at equal intervals.

See Also

- [Data label](#)
- [Tooltip](#)
- [Marker](#)

Print and Export in Blazor Charts Component

Print

The `PrintAsync` method can be used to print a rendered chart directly from the browser.

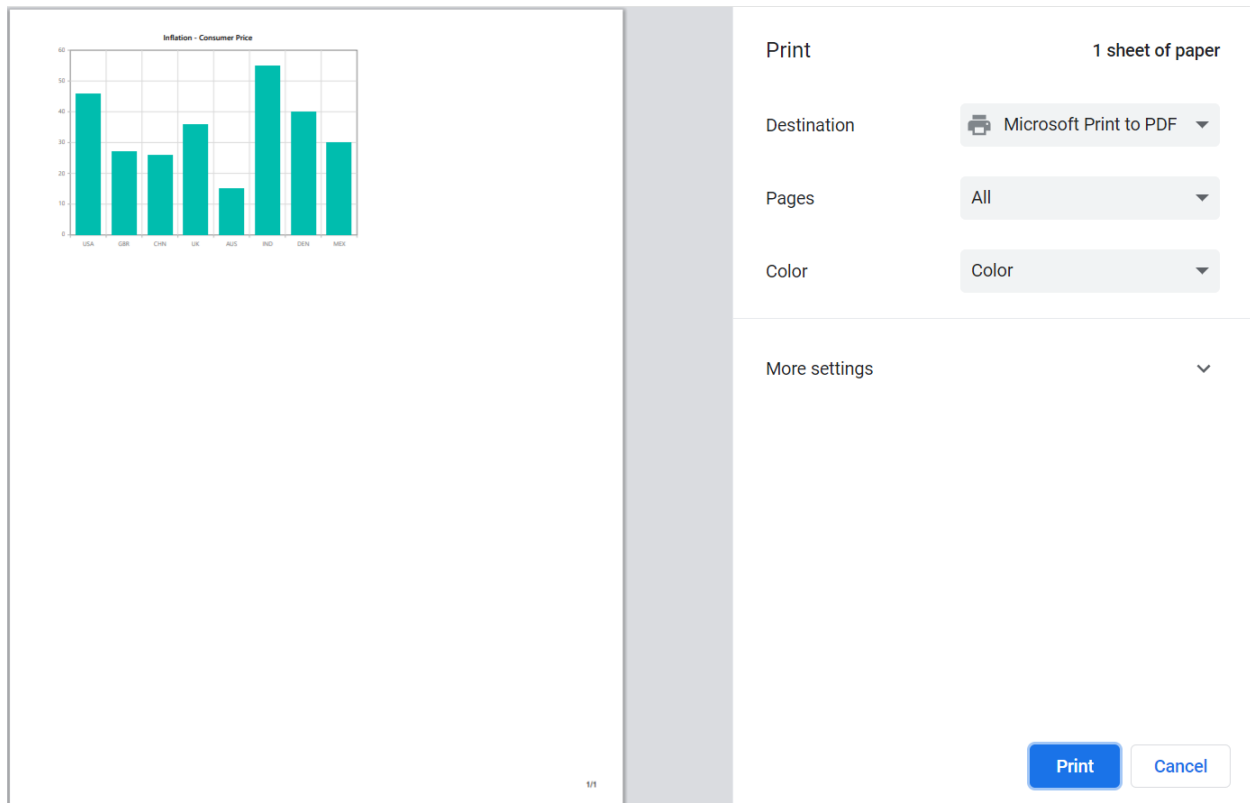
ASPX-CS

```
@using Syncfusion.Blazor.Charts
@using Syncfusion.Blazor.Buttons
<SfChart @ref="ChartObj" Title="Inflation - Consumer Price">
  <ChartPrimaryXAxis ValueType="Syncfusion.Blazor.Charts.ValueType.Category">
  </ChartPrimaryXAxis>
  <ChartSeriesCollection>
    <ChartSeries DataSource="@ConsumerDetails" XName="X" YName="YValue"
      Type="ChartSeriesType.Column">
    </ChartSeries>
  </ChartSeriesCollection>
</SfChart>
```

```

</ChartSeriesCollection>
</SfChart>
<SfButton Id="button" Content="Print" @onclick="Print" IsPrimary="true"
CssClass="e-flat"></SfButton>
@code{
SfChart ChartObj;
private async Task Print(MouseEventArgs args)
{
await ChartObj.PrintAsync();
}
public class ChartData
{
public string X { get; set; }
public double YValue { get; set; }
}
public List<ChartData> ConsumerDetails = new List<ChartData>
{
new ChartData { X= "USA", YValue= 46 },
new ChartData { X= "GBR", YValue= 27 },
new ChartData { X= "CHN", YValue= 26 },
new ChartData { X= "UK", YValue= 36 },
new ChartData { X= "AUS", YValue= 15 },
new ChartData { X= "IND", YValue= 55 },
new ChartData { X= "DEN", YValue= 40 },
new ChartData { X= "MEX", YValue= 30 }
};
}

```



Export

Using the `ExportAsync` method, the rendered chart can be exported to [JPEG](#), [PNG](#), [SVG](#), or [PDF](#) format. The [Export Type](#) specifies the image format and [FileName](#) specifies the name of the exported file. Both of these parameters are required input parameters for this method.

The optional parameters for this method are,

- `Orientation` - Specifies the portrait or landscape orientation in the PDF document.
- `AllowDownload` - Specifies whether to download or not. If not, base64 string will be returned.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
@using Syncfusion.Blazor.Buttons
<SfChart @ref="ChartObj" Title="Inflation - Consumer Price">
<ChartPrimaryXAxis ValueType="Syncfusion.Blazor.Charts.ValueType.Category">
</ChartPrimaryXAxis>
<ChartSeriesCollection>
<ChartSeries DataSource="@ConsumerDetails" XName="X" YName="YValue"
Type="ChartSeriesType.Column">
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
<SfButton Id="button" Content="Export" @onclick="Export" IsPrimary="true"
CssClass="e-flat"></SfButton>
@code{
SfChart ChartObj;
private async Task Export(MouseEventArgs args)
{
await ChartObj.ExportAsync(ExportType.PNG, "pngImage");
}
public class ChartData
{
public string X { get; set; }
public double YValue { get; set; }
}
public List<ChartData> ConsumerDetails = new List<ChartData>
{
new ChartData { X= "USA", YValue= 46 },
new ChartData { X= "GBR", YValue= 27 },
new ChartData { X= "CHN", YValue= 26 },
new ChartData { X= "UK", YValue= 36 },
new ChartData { X= "AUS", YValue= 15 },
new ChartData { X= "IND", YValue= 55 },
new ChartData { X= "DEN", YValue= 40 },
new ChartData { X= "MEX", YValue= 30 }
};
}
```

Refer to our [Blazor Charts](#) feature tour page for its groundbreaking feature representations and also explore our [Blazor Chart Example](#) to know various chart types and how to represent time-dependent data, showing trends at equal intervals.

See Also

- [Data Label](#)
- [Tooltip](#)
- [Legend](#)

<!-- markdownlint-disable MD036 -->

Technical Indicators in Blazor Charts Component

A technical indicator is a mathematical calculation that forecasts financial market direction using historical price, volume, or open interest data. There are 10 different types of technical indicators that can be used with the chart.

Accumulation Distribution

Accumulation Distribution combines price and volume to show how money may be flowing into or out of stock. To render an Accumulation Distribution Indicator, set the indicator [Type](#) as [AccumulationDistribution](#). To calculate the signal line [Volume](#) field is additionally added with [DataSource](#).

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart Title="AAPL - 2012-2017">
  <ChartAxes>
    <ChartAxis Name="secondary" OpposedPosition="true"RowIndex="0" Minimum="-
70000000000" Maximum="50000000000" Title="Accumulation Distribution"
Interval="6000000000"></ChartAxis>
  </ChartAxes>
  <ChartPrimaryXAxis ValueType="Syncfusion.Blazor.Charts.ValueType.DateTime"
Title="months" IntervalType="IntervalType.Months">
    <ChartAxisMajorGridLines Width="0"></ChartAxisMajorGridLines>
    <ChartAxisCrosshairTooltip Enable="true"></ChartAxisCrosshairTooltip>
  </ChartPrimaryXAxis>
  <ChartSeriesCollection>
    <ChartSeries DataSource="@StockDetails" Name="Apple Inc" XName="X"
Low="Low" High="High" Close="Close" Volume="Volume" Open="Open"
Width="2" Type="ChartSeriesType.Candle">
    </ChartSeries>
  </ChartSeriesCollection>
  <ChartIndicators>
    <ChartIndicator Type="TechnicalIndicators.AccumulationDistribution"
Field="FinancialDataFields.Close" SeriesName="Apple Inc"
YAxisName="secondary" Fill="#6063ff" Period=3>
    </ChartIndicator>
  </ChartIndicators>
</SfChart>

@code{
public class Data
{
  public DateTime X { get; set; }
  public double High { get; set; }
  public double Low { get; set; }
  public double Open { get; set; }
  public double Close { get; set; }
  public double Volume { get; set; }
}
```

```

}
public List<Data> StockDetails = new List<Data>
{
    new Data{X= new DateTime(2012,10,15), Open= 90.3357, High= 93.2557, Low=
    87.0885,Close= 87.12,Volume= 646996264},
    new Data{X= new DateTime(2012,10,22), Open= 87.4885, High= 90.7685, Low=
    84.4285,Close= 86.2857,Volume= 866040680 },
    new Data{X= new DateTime(2012,10,29), Open= 84.9828, High= 86.1428, Low=
    82.1071,Close= 82.4,Volume= 367371310},
    new Data{X= new DateTime(2012,11,05), Open= 83.3593, High= 84.3914, Low=
    76.2457,Close= 78.1514,Volume= 919719846},
    new Data{X= new DateTime(2012,11,12), Open= 79.1643, High= 79.2143, Low=
    72.25,Close= 75.3825,Volume= 894382149},
    new Data{X= new DateTime(2012,11,19), Open= 77.2443, High= 81.7143, Low=
    77.1257,Close= 81.6428,Volume= 527416747},
    new Data{X= new DateTime(2012,11,26), Open= 82.2714, High= 84.8928, Low=
    81.7514,Close= 83.6114,Volume= 646467974},
    new Data{X= new DateTime(2012,12,03), Open= 84.8071, High= 84.9414, Low=
    74.09,Close= 76.1785,Volume= 980096264},
    new Data{X= new DateTime(2012,12,10), Open= 75, High= 78.5085, Low=
    72.2257,Close= 72.8277,Volume= 835016110},
    new Data{X= new DateTime(2012,12,17), Open= 72.7043, High= 76.4143, Low=
    71.6043,Close= 74.19,Volume= 726150329},
    new Data{X= new DateTime(2012,12,24), Open= 74.3357, High= 74.8928, Low=
    72.0943,Close= 72.7984,Volume= 321104733},
    new Data{X= new DateTime(2012,12,31), Open= 72.9328, High= 79.2857, Low=
    72.7143,Close= 75.2857,Volume= 540854882},
    new Data{X= new DateTime(2013,01,07), Open= 74.5714, High= 75.9843, Low=
    73.6,Close= 74.3285,Volume= 574594262},
    new Data{X= new DateTime(2013,01,14), Open= 71.8114, High= 72.9643, Low=
    69.0543,Close= 71.4285,Volume= 803105621},
    new Data{X= new DateTime(2013,01,21), Open= 72.08, High= 73.57, Low=
    62.1428,Close= 62.84,Volume= 971912560},
    new Data{X= new DateTime(2013,01,28), Open= 62.5464, High= 66.0857, Low=
    62.2657,Close= 64.8028,Volume= 656549587},
    new Data{X= new DateTime(2013,02,04), Open= 64.8443, High= 68.4014, Low=
    63.1428,Close= 67.8543,Volume= 743778993},
    new Data{X= new DateTime(2013,02,11), Open= 68.0714, High= 69.2771, Low=
    65.7028,Close= 65.7371,Volume= 585292366},
    new Data{X= new DateTime(2013,02,18), Open= 65.8714, High= 66.1043, Low=
    63.26,Close= 64.4014,Volume= 421766997},
    new Data{X= new DateTime(2013,02,25), Open= 64.8357, High= 65.0171, Low=
    61.4257,Close= 61.4957,Volume= 582741215},
    new Data{X= new DateTime(2013,03,04), Open= 61.1143, High= 62.2043, Low=
    59.8571,Close= 61.6743,Volume= 632856539},
    new Data{X= new DateTime(2013,03,11), Open= 61.3928, High= 63.4614, Low=
    60.7343,Close= 63.38,Volume= 572066981},
    new Data{X= new DateTime(2013,03,18), Open= 63.0643, High= 66.0143, Low=
    63.0286,Close= 65.9871,Volume= 552156035},
    new Data{X= new DateTime(2013,03,25), Open= 66.3843, High= 67.1357, Low=
    63.0886,Close= 63.2371,Volume= 390762517},
    new Data{X= new DateTime(2013,04,01), Open= 63.1286, High= 63.3854, Low=
    59.9543,Close= 60.4571,Volume= 505273732},
    new Data{X= new DateTime(2013,04,08), Open= 60.6928, High= 62.57, Low=
    60.3557,Close= 61.4,Volume= 387323550},
    new Data{X= new DateTime(2013,04,15), Open= 61, High= 61.1271, Low=
    55.0143,Close= 55.79,Volume= 709945604},

```

```
new Data{X= new DateTime(2013,04,22), Open= 56.0914, High= 59.8241, Low=
55.8964,Close= 59.6007,Volume= 787007506},
new Data{X= new DateTime(2013,04,29), Open= 60.0643, High= 64.7471, Low=
60,Close= 64.2828,Volume= 655020017},
new Data{X= new DateTime(2013,05,06), Open= 65.1014, High= 66.5357, Low=
64.3543,Close= 64.71,Volume= 545488533},
new Data{X= new DateTime(2013,05,13), Open= 64.5014, High= 65.4143, Low=
59.8428,Close= 61.8943,Volume= 633706550},
new Data{X= new DateTime(2013,05,20), Open= 61.7014, High= 64.05, Low=
61.4428,Close= 63.5928,Volume= 494379068},
new Data{X= new DateTime(2013,05,27), Open= 64.2714, High= 65.3, Low=
62.7714,Close= 64.2478,Volume= 362907830},
new Data{X= new DateTime(2013,06,03), Open= 64.39, High= 64.9186, Low=
61.8243,Close= 63.1158,Volume= 443249793},
new Data{X= new DateTime(2013,06,10), Open= 63.5328, High= 64.1541, Low=
61.2143,Close= 61.4357,Volume= 389680092},
new Data{X= new DateTime(2013,06,17), Open= 61.6343, High= 62.2428, Low=
58.3,Close= 59.0714,Volume= 400384818},
new Data{X= new DateTime(2013,06,24), Open= 58.2, High= 58.38, Low=
55.5528,Close= 56.6471,Volume= 519314826},
new Data{X= new DateTime(2013,07,01), Open= 57.5271, High= 60.47, Low=
57.3171,Close= 59.6314,Volume= 343878841},
new Data{X= new DateTime(2013,07,08), Open= 60.0157, High= 61.3986, Low=
58.6257,Close= 60.93,Volume= 384106977},
new Data{X= new DateTime(2013,07,15), Open= 60.7157, High= 62.1243, Low=
60.5957,Close= 60.7071,Volume= 286035513},
new Data{X= new DateTime(2013,07,22), Open= 61.3514, High= 63.5128, Low=
59.8157,Close= 62.9986,Volume= 395816827},
new Data{X= new DateTime(2013,07,29), Open= 62.9714, High= 66.1214, Low=
62.8857,Close= 66.0771,Volume= 339668858},
new Data{X= new DateTime(2013,08,12), Open= 65.2657, High= 72.0357, Low=
65.2328,Close= 71.7614,Volume= 711563584},
new Data{X= new DateTime(2013,08,19), Open= 72.0485, High= 73.3914, Low=
71.1714,Close= 71.5743,Volume= 417119660},
new Data{X= new DateTime(2013,08,26), Open= 71.5357, High= 72.8857, Low=
69.4286,Close= 69.6023,Volume= 392805888},
new Data{X= new DateTime(2013,09,02), Open= 70.4428, High= 71.7485, Low=
69.6214,Close= 71.1743,Volume= 317244380},
new Data{X= new DateTime(2013,09,09), Open= 72.1428, High= 72.56, Low=
66.3857,Close= 66.4143,Volume= 669376320},
new Data{X= new DateTime(2013,09,16), Open= 65.8571, High= 68.3643, Low=
63.8886,Close= 66.7728,Volume= 625142677},
new Data{X= new DateTime(2013,09,23), Open= 70.8714, High= 70.9871, Low=
68.6743,Close= 68.9643,Volume= 475274537},
new Data{X= new DateTime(2013,09,30), Open= 68.1786, High= 70.3357, Low=
67.773,Close= 69.0043,Volume= 368198906},
new Data{X= new DateTime(2013,10,07), Open= 69.5086, High= 70.5486, Low=
68.3257,Close= 70.4017,Volume= 361437661},
new Data{X= new DateTime(2013,10,14), Open= 69.9757, High= 72.7514, Low=
69.9071,Close= 72.6985,Volume= 342694379},
new Data{X= new DateTime(2013,10,21), Open= 73.11, High= 76.1757, Low=
72.5757,Close= 75.1368,Volume= 490458997},
new Data{X= new DateTime(2013,10,28), Open= 75.5771, High= 77.0357, Low=
73.5057,Close= 74.29,Volume= 508130174},
new Data{X= new DateTime(2013,11,04), Open= 74.4428, High= 75.555, Low=
73.1971,Close= 74.3657,Volume= 318132218},
```

```
new Data{X= new DateTime(2013,11,11), Open= 74.2843, High= 75.6114, Low=
73.4871,Close= 74.9987,Volume= 306711021},
new Data{X= new DateTime(2013,11,18), Open= 74.9985, High= 75.3128, Low=
73.3814,Close= 74.2571,Volume= 282778778},
};
}
```



Average True Range (ATR)

Average True Range (ATR) measures the stock volatility by comparing the current value with the previous value. To render an Average True Range (ATR) Indicator, set the indicator [Type](#) as [Atr](#).

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart Title="AAPL - 2012-2017">
  <ChartAxes>
    <ChartAxis Name="secondary" OpposedPosition="true" RowIndex="0" Minimum="-
7000000000" Maximum="5000000000" Title="Accumulation Distribution"
Interval="6000000000"></ChartAxis>
  </ChartAxes>
  <ChartPrimaryXAxis ValueType="Syncfusion.Blazor.Charts.ValueType.DateTime"
Title="months" IntervalType="IntervalType.Months">
    <ChartAxisMajorGridLines Width="0"></ChartAxisMajorGridLines>
    <ChartAxisCrosshairTooltip Enable="true"></ChartAxisCrosshairTooltip>
  </ChartPrimaryXAxis>
  <ChartSeriesCollection>
```

```

<ChartSeries DataSource="@StockDetails" XName="X" YName="Y" Low="Low"
High="High" Close="Close" Volume="Volume" Open="Open"
Width="2" Name="Apple Inc" Type="ChartSeriesType.Candle">
</ChartSeries>
</ChartSeriesCollection>
<ChartIndicators>
<ChartIndicator Type="TechnicalIndicators.Atr"
Field="FinancialDataFields.Low" SeriesName="Apple Inc" YAxisName="secondary"
Fill="#6063ff" Period=3>
</ChartIndicator>
</ChartIndicators>
</SfChart>
@code{
public class Data
{
public DateTime X { get; set; }
public double High { get; set; }
public double Low { get; set; }
public double Open { get; set; }
public double Close { get; set; }
public double Volume { get; set; }
}
public List<Data> StockDetails = new List<Data>
{
new Data{X= new DateTime(2012,10,15), Open= 90.3357, High= 93.2557, Low=
87.0885,Close= 87.12,Volume= 646996264},
new Data{X= new DateTime(2012,10,22), Open= 87.4885, High= 90.7685, Low=
84.4285,Close= 86.2857,Volume= 866040680 },
new Data{X= new DateTime(2012,10,29), Open= 84.9828, High= 86.1428, Low=
82.1071,Close= 82.4,Volume= 367371310},
new Data{X= new DateTime(2012,11,05), Open= 83.3593, High= 84.3914, Low=
76.2457,Close= 78.1514,Volume= 919719846},
new Data{X= new DateTime(2012,11,12), Open= 79.1643, High= 79.2143, Low=
72.25,Close= 75.3825,Volume= 894382149},
new Data{X= new DateTime(2012,11,19), Open= 77.2443, High= 81.7143, Low=
77.1257,Close= 81.6428,Volume= 527416747},
new Data{X= new DateTime(2012,11,26), Open= 82.2714, High= 84.8928, Low=
81.7514,Close= 83.6114,Volume= 646467974},
new Data{X= new DateTime(2012,12,03), Open= 84.8071, High= 84.9414, Low=
74.09,Close= 76.1785,Volume= 980096264},
new Data{X= new DateTime(2012,12,10), Open= 75, High= 78.5085, Low=
72.2257,Close= 72.8277,Volume= 835016110},
new Data{X= new DateTime(2012,12,17), Open= 72.7043, High= 76.4143, Low=
71.6043,Close= 74.19,Volume= 726150329},
new Data{X= new DateTime(2012,12,24), Open= 74.3357, High= 74.8928, Low=
72.0943,Close= 72.7984,Volume= 321104733},
new Data{X= new DateTime(2012,12,31), Open= 72.9328, High= 79.2857, Low=
72.7143,Close= 75.2857,Volume= 540854882},
new Data{X= new DateTime(2013,01,07), Open= 74.5714, High= 75.9843, Low=
73.6,Close= 74.3285,Volume= 574594262},
new Data{X= new DateTime(2013,01,14), Open= 71.8114, High= 72.9643, Low=
69.0543,Close= 71.4285,Volume= 803105621},
new Data{X= new DateTime(2013,01,21), Open= 72.08, High= 73.57, Low=
62.1428,Close= 62.84,Volume= 971912560},
new Data{X= new DateTime(2013,01,28), Open= 62.5464, High= 66.0857, Low=
62.2657,Close= 64.8028,Volume= 656549587},

```



```
new Data{X= new DateTime(2013,02,04), Open= 64.8443, High= 68.4014, Low=
63.1428,Close= 67.8543,Volume= 743778993},
new Data{X= new DateTime(2013,02,11), Open= 68.0714, High= 69.2771, Low=
65.7028,Close= 65.7371,Volume= 585292366},
new Data{X= new DateTime(2013,02,18), Open= 65.8714, High= 66.1043, Low=
63.26,Close= 64.4014,Volume= 421766997},
new Data{X= new DateTime(2013,02,25), Open= 64.8357, High= 65.0171, Low=
61.4257,Close= 61.4957,Volume= 582741215},
new Data{X= new DateTime(2013,03,04), Open= 61.1143, High= 62.2043, Low=
59.8571,Close= 61.6743,Volume= 632856539},
new Data{X= new DateTime(2013,03,11), Open= 61.3928, High= 63.4614, Low=
60.7343,Close= 63.38,Volume= 572066981},
new Data{X= new DateTime(2013,03,18), Open= 63.0643, High= 66.0143, Low=
63.0286,Close= 65.9871,Volume= 552156035},
new Data{X= new DateTime(2013,03,25), Open= 66.3843, High= 67.1357, Low=
63.0886,Close= 63.2371,Volume= 390762517},
new Data{X= new DateTime(2013,04,01), Open= 63.1286, High= 63.3854, Low=
59.9543,Close= 60.4571,Volume= 505273732},
new Data{X= new DateTime(2013,04,08), Open= 60.6928, High= 62.57, Low=
60.3557,Close= 61.4,Volume= 387323550},
new Data{X= new DateTime(2013,04,15), Open= 61, High= 61.1271, Low=
55.0143,Close= 55.79,Volume= 709945604},
new Data{X= new DateTime(2013,04,22), Open= 56.0914, High= 59.8241, Low=
55.8964,Close= 59.6007,Volume= 787007506},
new Data{X= new DateTime(2013,04,29), Open= 60.0643, High= 64.7471, Low=
60,Close= 64.2828,Volume= 655020017},
new Data{X= new DateTime(2013,05,06), Open= 65.1014, High= 66.5357, Low=
64.3543,Close= 64.71,Volume= 545488533},
new Data{X= new DateTime(2013,05,13), Open= 64.5014, High= 65.4143, Low=
59.8428,Close= 61.8943,Volume= 633706550},
new Data{X= new DateTime(2013,05,20), Open= 61.7014, High= 64.05, Low=
61.4428,Close= 63.5928,Volume= 494379068},
new Data{X= new DateTime(2013,05,27), Open= 64.2714, High= 65.3, Low=
62.7714,Close= 64.2478,Volume= 362907830},
new Data{X= new DateTime(2013,06,03), Open= 64.39, High= 64.9186, Low=
61.8243,Close= 63.1158,Volume= 443249793},
new Data{X= new DateTime(2013,06,10), Open= 63.5328, High= 64.1541, Low=
61.2143,Close= 61.4357,Volume= 389680092},
new Data{X= new DateTime(2013,06,17), Open= 61.6343, High= 62.2428, Low=
58.3,Close= 59.0714,Volume= 400384818},
new Data{X= new DateTime(2013,06,24), Open= 58.2, High= 58.38, Low=
55.5528,Close= 56.6471,Volume= 519314826},
new Data{X= new DateTime(2013,07,01), Open= 57.5271, High= 60.47, Low=
57.3171,Close= 59.6314,Volume= 343878841},
new Data{X= new DateTime(2013,07,08), Open= 60.0157, High= 61.3986, Low=
58.6257,Close= 60.93,Volume= 384106977},
new Data{X= new DateTime(2013,07,15), Open= 60.7157, High= 62.1243, Low=
60.5957,Close= 60.7071,Volume= 286035513},
new Data{X= new DateTime(2013,07,22), Open= 61.3514, High= 63.5128, Low=
59.8157,Close= 62.9986,Volume= 395816827},
new Data{X= new DateTime(2013,07,29), Open= 62.9714, High= 66.1214, Low=
62.8857,Close= 66.0771,Volume= 339668858},
new Data{X= new DateTime(2013,08,12), Open= 65.2657, High= 72.0357, Low=
65.2328,Close= 71.7614,Volume= 711563584},
new Data{X= new DateTime(2013,08,19), Open= 72.0485, High= 73.3914, Low=
71.1714,Close= 71.5743,Volume= 417119660},
```

```

new Data{X= new DateTime(2013,08,26), Open= 71.5357, High= 72.8857, Low=
69.4286,Close= 69.6023,Volume= 392805888},
new Data{X= new DateTime(2013,09,02), Open= 70.4428, High= 71.7485, Low=
69.6214,Close= 71.1743,Volume= 317244380},
new Data{X= new DateTime(2013,09,09), Open= 72.1428, High= 72.56, Low=
66.3857,Close= 66.4143,Volume= 669376320},
new Data{X= new DateTime(2013,09,16), Open= 65.8571, High= 68.3643, Low=
63.8886,Close= 66.7728,Volume= 625142677},
new Data{X= new DateTime(2013,09,23), Open= 70.8714, High= 70.9871, Low=
68.6743,Close= 68.9643,Volume= 475274537},
new Data{X= new DateTime(2013,09,30), Open= 68.1786, High= 70.3357, Low=
67.773,Close= 69.0043,Volume= 368198906},
new Data{X= new DateTime(2013,10,07), Open= 69.5086, High= 70.5486, Low=
68.3257,Close= 70.4017,Volume= 361437661},
new Data{X= new DateTime(2013,10,14), Open= 69.9757, High= 72.7514, Low=
69.9071,Close= 72.6985,Volume= 342694379},
new Data{X= new DateTime(2013,10,21), Open= 73.11, High= 76.1757, Low=
72.5757,Close= 75.1368,Volume= 490458997},
new Data{X= new DateTime(2013,10,28), Open= 75.5771, High= 77.0357, Low=
73.5057,Close= 74.29,Volume= 508130174},
new Data{X= new DateTime(2013,11,04), Open= 74.4428, High= 75.555, Low=
73.1971,Close= 74.3657,Volume= 318132218},
new Data{X= new DateTime(2013,11,11), Open= 74.2843, High= 75.6114, Low=
73.4871,Close= 74.9987,Volume= 306711021},
new Data{X= new DateTime(2013,11,18), Open= 74.9985, High= 75.3128, Low=
73.3814,Close= 74.2571,Volume= 282778778},
};
}

```

Bollinger Bands

A chart overlay that shows the upper and lower limits of normal price movements based on the standard deviation of prices. To render Bollinger Bands, set indicator [Type](#) as [BollingerBands](#). Bollinger Bands will be represented by three lines - upper line, lower line, and signal line. In Bollinger Bands, default value of the [Period](#) is **14** and the [StandardDeviations](#) is **2**.

ASPX-CS

```

@using Syncfusion.Blazor.Charts
<SfChart Title="AAPL - 2012-2017">
<ChartPrimaryXAxis ValueTypes="Syncfusion.Blazor.Charts.ValueType.DateTime"
Title="months" IntervalType="IntervalType.Months">
<ChartAxisMajorGridLines Width="0"></ChartAxisMajorGridLines>
<ChartAxisCrosshairTooltip Enable="true"></ChartAxisCrosshairTooltip>
</ChartPrimaryXAxis>
<ChartSeriesCollection>
<ChartSeries DataSource="@StockDetails" XName="X" YName="Y" Low="Low"
High="High" Close="Close" Volume="Volume" Open="Open"
Width="2" Name="Apple Inc" Type="ChartSeriesType.Candle">
</ChartSeries>
</ChartSeriesCollection>
<ChartIndicators>
<ChartIndicator Type="TechnicalIndicators.BollingerBands"
Field="FinancialDataFields.Close" SeriesName="Apple Inc" Fill="#6063ff"
Period=3>
<ChartIndicatorUpperLine Color="Orange"></ChartIndicatorUpperLine>
<ChartIndicatorLowerLine Color="Yellow"></ChartIndicatorLowerLine>

```

```

</ChartIndicator>
</ChartIndicators>
</SfChart>
@code{
public class Data
{
public DateTime X { get; set; }
public double High { get; set; }
public double Low { get; set; }
public double Open { get; set; }
public double Close { get; set; }
public double Volume { get; set; }
}
public List<Data> StockDetails = new List<Data>
{
new Data{X= new DateTime(2012,10,15), Open= 90.3357, High= 93.2557, Low=
87.0885,Close= 87.12,Volume= 646996264},
new Data{X= new DateTime(2012,10,22), Open= 87.4885, High= 90.7685, Low=
84.4285,Close= 86.2857,Volume= 866040680 },
new Data{X= new DateTime(2012,10,29), Open= 84.9828, High= 86.1428, Low=
82.1071,Close= 82.4,Volume= 367371310},
new Data{X= new DateTime(2012,11,05), Open= 83.3593, High= 84.3914, Low=
76.2457,Close= 78.1514,Volume= 919719846},
new Data{X= new DateTime(2012,11,12), Open= 79.1643, High= 79.2143, Low=
72.25,Close= 75.3825,Volume= 894382149},
new Data{X= new DateTime(2012,11,19), Open= 77.2443, High= 81.7143, Low=
77.1257,Close= 81.6428,Volume= 527416747},
new Data{X= new DateTime(2012,11,26), Open= 82.2714, High= 84.8928, Low=
81.7514,Close= 83.6114,Volume= 646467974},
new Data{X= new DateTime(2012,12,03), Open= 84.8071, High= 84.9414, Low=
74.09,Close= 76.1785,Volume= 980096264},
new Data{X= new DateTime(2012,12,10), Open= 75, High= 78.5085, Low=
72.2257,Close= 72.8277,Volume= 835016110},
new Data{X= new DateTime(2012,12,17), Open= 72.7043, High= 76.4143, Low=
71.6043,Close= 74.19,Volume= 726150329},
new Data{X= new DateTime(2012,12,24), Open= 74.3357, High= 74.8928, Low=
72.0943,Close= 72.7984,Volume= 321104733},
new Data{X= new DateTime(2012,12,31), Open= 72.9328, High= 79.2857, Low=
72.7143,Close= 75.2857,Volume= 540854882},
new Data{X= new DateTime(2013,01,07), Open= 74.5714, High= 75.9843, Low=
73.6,Close= 74.3285,Volume= 574594262},
new Data{X= new DateTime(2013,01,14), Open= 71.8114, High= 72.9643, Low=
69.0543,Close= 71.4285,Volume= 803105621},
new Data{X= new DateTime(2013,01,21), Open= 72.08, High= 73.57, Low=
62.1428,Close= 62.84,Volume= 971912560},
new Data{X= new DateTime(2013,01,28), Open= 62.5464, High= 66.0857, Low=
62.2657,Close= 64.8028,Volume= 656549587},
new Data{X= new DateTime(2013,02,04), Open= 64.8443, High= 68.4014, Low=
63.1428,Close= 67.8543,Volume= 743778993},
new Data{X= new DateTime(2013,02,11), Open= 68.0714, High= 69.2771, Low=
65.7028,Close= 65.7371,Volume= 585292366},
new Data{X= new DateTime(2013,02,18), Open= 65.8714, High= 66.1043, Low=
63.26,Close= 64.4014,Volume= 421766997},
new Data{X= new DateTime(2013,02,25), Open= 64.8357, High= 65.0171, Low=
61.4257,Close= 61.4957,Volume= 582741215},
new Data{X= new DateTime(2013,03,04), Open= 61.1143, High= 62.2043, Low=
59.8571,Close= 61.6743,Volume= 632856539},

```

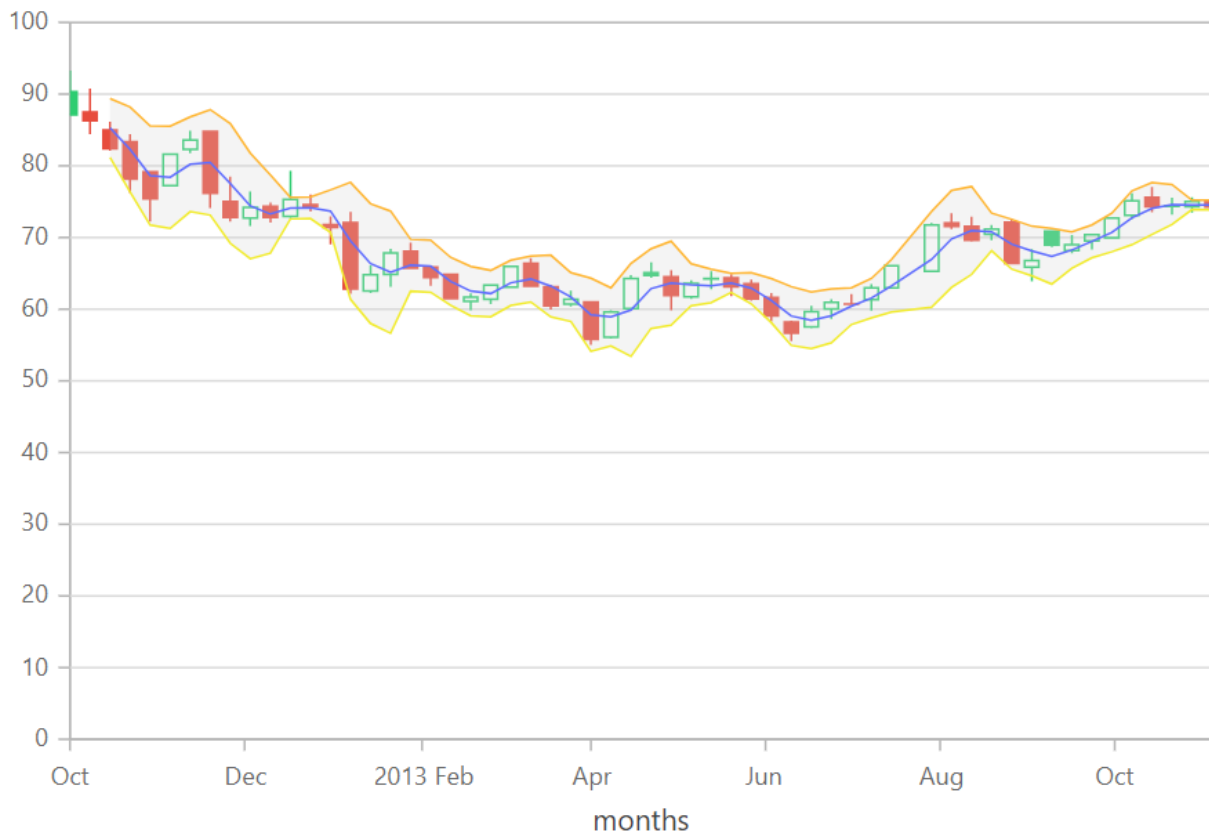
```
new Data{X= new DateTime(2013,03,11), Open= 61.3928, High= 63.4614, Low=
60.7343,Close= 63.38,Volume= 572066981},
new Data{X= new DateTime(2013,03,18), Open= 63.0643, High= 66.0143, Low=
63.0286,Close= 65.9871,Volume= 552156035},
new Data{X= new DateTime(2013,03,25), Open= 66.3843, High= 67.1357, Low=
63.0886,Close= 63.2371,Volume= 390762517},
new Data{X= new DateTime(2013,04,01), Open= 63.1286, High= 63.3854, Low=
59.9543,Close= 60.4571,Volume= 505273732},
new Data{X= new DateTime(2013,04,08), Open= 60.6928, High= 62.57, Low=
60.3557,Close= 61.4,Volume= 387323550},
new Data{X= new DateTime(2013,04,15), Open= 61, High= 61.1271, Low=
55.0143,Close= 55.79,Volume= 709945604},
new Data{X= new DateTime(2013,04,22), Open= 56.0914, High= 59.8241, Low=
55.8964,Close= 59.6007,Volume= 787007506},
new Data{X= new DateTime(2013,04,29), Open= 60.0643, High= 64.7471, Low=
60,Close= 64.2828,Volume= 655020017},
new Data{X= new DateTime(2013,05,06), Open= 65.1014, High= 66.5357, Low=
64.3543,Close= 64.71,Volume= 545488533},
new Data{X= new DateTime(2013,05,13), Open= 64.5014, High= 65.4143, Low=
59.8428,Close= 61.8943,Volume= 633706550},
new Data{X= new DateTime(2013,05,20), Open= 61.7014, High= 64.05, Low=
61.4428,Close= 63.5928,Volume= 494379068},
new Data{X= new DateTime(2013,05,27), Open= 64.2714, High= 65.3, Low=
62.7714,Close= 64.2478,Volume= 362907830},
new Data{X= new DateTime(2013,06,03), Open= 64.39, High= 64.9186, Low=
61.8243,Close= 63.1158,Volume= 443249793},
new Data{X= new DateTime(2013,06,10), Open= 63.5328, High= 64.1541, Low=
61.2143,Close= 61.4357,Volume= 389680092},
new Data{X= new DateTime(2013,06,17), Open= 61.6343, High= 62.2428, Low=
58.3,Close= 59.0714,Volume= 400384818},
new Data{X= new DateTime(2013,06,24), Open= 58.2, High= 58.38, Low=
55.5528,Close= 56.6471,Volume= 519314826},
new Data{X= new DateTime(2013,07,01), Open= 57.5271, High= 60.47, Low=
57.3171,Close= 59.6314,Volume= 343878841},
new Data{X= new DateTime(2013,07,08), Open= 60.0157, High= 61.3986, Low=
58.6257,Close= 60.93,Volume= 384106977},
new Data{X= new DateTime(2013,07,15), Open= 60.7157, High= 62.1243, Low=
60.5957,Close= 60.7071,Volume= 286035513},
new Data{X= new DateTime(2013,07,22), Open= 61.3514, High= 63.5128, Low=
59.8157,Close= 62.9986,Volume= 395816827},
new Data{X= new DateTime(2013,07,29), Open= 62.9714, High= 66.1214, Low=
62.8857,Close= 66.0771,Volume= 339668858},
new Data{X= new DateTime(2013,08,12), Open= 65.2657, High= 72.0357, Low=
65.2328,Close= 71.7614,Volume= 711563584},
new Data{X= new DateTime(2013,08,19), Open= 72.0485, High= 73.3914, Low=
71.1714,Close= 71.5743,Volume= 417119660},
new Data{X= new DateTime(2013,08,26), Open= 71.5357, High= 72.8857, Low=
69.4286,Close= 69.6023,Volume= 392805888},
new Data{X= new DateTime(2013,09,02), Open= 70.4428, High= 71.7485, Low=
69.6214,Close= 71.1743,Volume= 317244380},
new Data{X= new DateTime(2013,09,09), Open= 72.1428, High= 72.56, Low=
66.3857,Close= 66.4143,Volume= 669376320},
new Data{X= new DateTime(2013,09,16), Open= 65.8571, High= 68.3643, Low=
63.8886,Close= 66.7728,Volume= 625142677},
new Data{X= new DateTime(2013,09,23), Open= 70.8714, High= 70.9871, Low=
68.6743,Close= 68.9643,Volume= 475274537},
```

```

new Data{X= new DateTime(2013,09,30), Open= 68.1786, High= 70.3357, Low=
67.773,Close= 69.0043,Volume= 368198906},
new Data{X= new DateTime(2013,10,07), Open= 69.5086, High= 70.5486, Low=
68.3257,Close= 70.4017,Volume= 361437661},
new Data{X= new DateTime(2013,10,14), Open= 69.9757, High= 72.7514, Low=
69.9071,Close= 72.6985,Volume= 342694379},
new Data{X= new DateTime(2013,10,21), Open= 73.11, High= 76.1757, Low=
72.5757,Close= 75.1368,Volume= 490458997},
new Data{X= new DateTime(2013,10,28), Open= 75.5771, High= 77.0357, Low=
73.5057,Close= 74.29,Volume= 508130174},
new Data{X= new DateTime(2013,11,04), Open= 74.4428, High= 75.555, Low=
73.1971,Close= 74.3657,Volume= 318132218},
new Data{X= new DateTime(2013,11,11), Open= 74.2843, High= 75.6114, Low=
73.4871,Close= 74.9987,Volume= 306711021},
new Data{X= new DateTime(2013,11,18), Open= 74.9985, High= 75.3128, Low=
73.3814,Close= 74.2571,Volume= 282778778},
};
}

```

AAPL - 2012-2017



Customization of Bollinger Bands

The upper line's [Width](#) and [Color](#) can be customized using the [UpperLine](#) property of the indicator, while the lower line's [Width](#) and [Color](#) can be customized using the [LowerLine](#) property of the indicator.

ASPX-CS

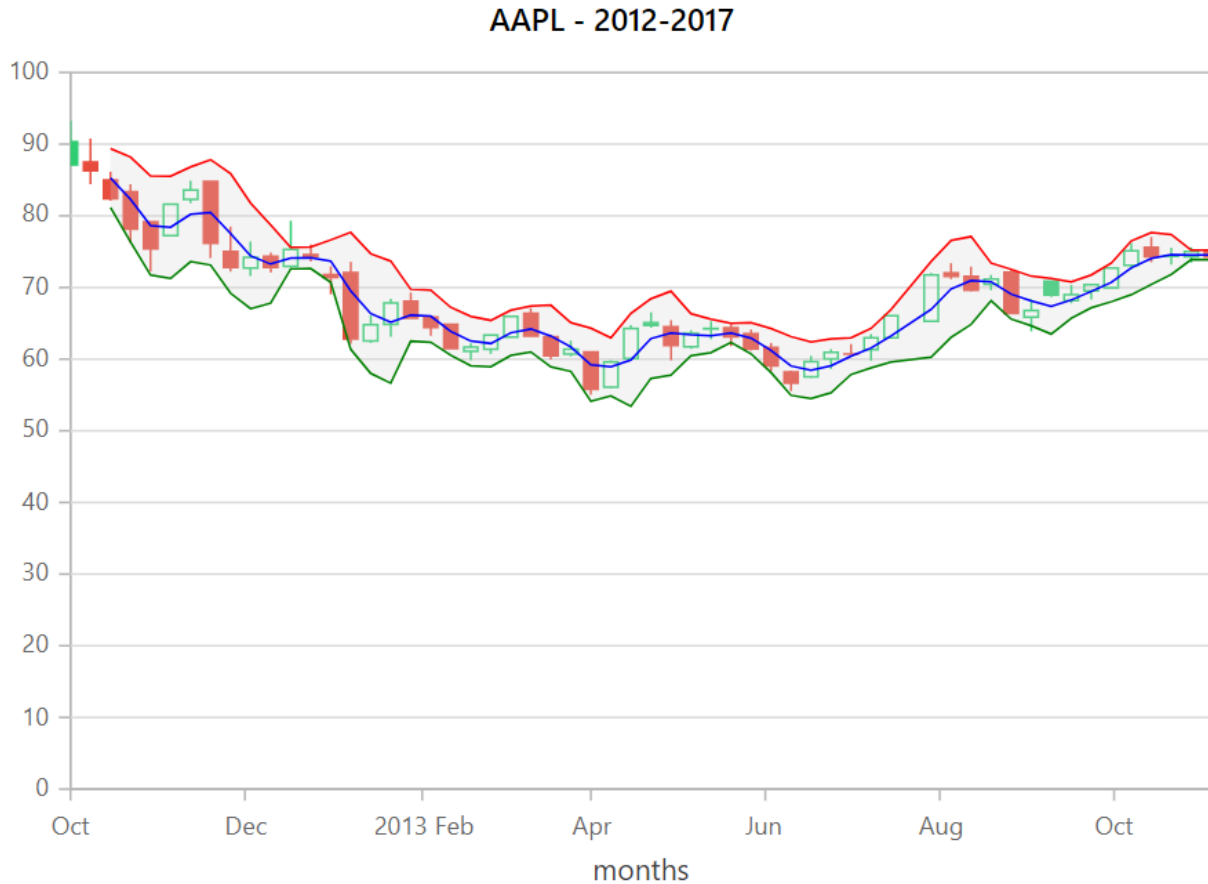
```

@using Syncfusion.Blazor.Charts
<SfChart Title="AAPL - 2012-2017">
  <ChartPrimaryXAxis ValueTypes="Syncfusion.Blazor.Charts.ValueType.DateTime"
    Title="months" IntervalType="IntervalType.Months">
  <ChartAxisMajorGridLines Width="0"></ChartAxisMajorGridLines>
  <ChartAxisCrosshairTooltip Enable="true"></ChartAxisCrosshairTooltip>
</ChartPrimaryXAxis>
<ChartSeriesCollection>
  <ChartSeries DataSource="@StockDetails" XName="X" YName="Y" Low="Low"
    High="High" Close="Close" Volume="Volume" Open="Open"
    Width="2" Name="Apple Inc" Type="ChartSeriesType.Candle">
  </ChartSeries>
</ChartSeriesCollection>
<ChartIndicators>
  <ChartIndicator Type="TechnicalIndicators.BollingerBands"
    Field="FinancialDataFields.Close" SeriesName="Apple Inc" Fill="blue"
    Period=3>
  <ChartIndicatorUpperLine Color="red"></ChartIndicatorUpperLine>
  <ChartIndicatorLowerLine Color="green"></ChartIndicatorLowerLine>
</ChartIndicator>
</ChartIndicators>
</SfChart>
@code{
public class Data
{
  public DateTime X { get; set; }
  public double High { get; set; }
  public double Low { get; set; }
  public double Open { get; set; }
  public double Close { get; set; }
  public double Volume { get; set; }
}
public List<Data> StockDetails = new List<Data>
{
  new Data{X= new DateTime(2012,10,15), Open= 90.3357, High= 93.2557, Low=
87.0885,Close= 87.12,Volume= 646996264},
  new Data{X= new DateTime(2012,10,22), Open= 87.4885, High= 90.7685, Low=
84.4285,Close= 86.2857,Volume= 866040680 },
  new Data{X= new DateTime(2012,10,29), Open= 84.9828, High= 86.1428, Low=
82.1071,Close= 82.4,Volume= 367371310},
  new Data{X= new DateTime(2012,11,05), Open= 83.3593, High= 84.3914, Low=
76.2457,Close= 78.1514,Volume= 919719846},
  new Data{X= new DateTime(2012,11,12), Open= 79.1643, High= 79.2143, Low=
72.25,Close= 75.3825,Volume= 894382149},
  new Data{X= new DateTime(2012,11,19), Open= 77.2443, High= 81.7143, Low=
77.1257,Close= 81.6428,Volume= 527416747},
  new Data{X= new DateTime(2012,11,26), Open= 82.2714, High= 84.8928, Low=
81.7514,Close= 83.6114,Volume= 646467974},
  new Data{X= new DateTime(2012,12,03), Open= 84.8071, High= 84.9414, Low=
74.09,Close= 76.1785,Volume= 980096264},
  new Data{X= new DateTime(2012,12,10), Open= 75, High= 78.5085, Low=
72.2257,Close= 72.8277,Volume= 835016110},
  new Data{X= new DateTime(2012,12,17), Open= 72.7043, High= 76.4143, Low=
71.6043,Close= 74.19,Volume= 726150329},
  new Data{X= new DateTime(2012,12,24), Open= 74.3357, High= 74.8928, Low=
72.0943,Close= 72.7984,Volume= 321104733},

```

```
new Data{X= new DateTime(2012,12,31), Open= 72.9328, High= 79.2857, Low=
72.7143,Close= 75.2857,Volume= 540854882},
new Data{X= new DateTime(2013,01,07), Open= 74.5714, High= 75.9843, Low=
73.6,Close= 74.3285,Volume= 574594262},
new Data{X= new DateTime(2013,01,14), Open= 71.8114, High= 72.9643, Low=
69.0543,Close= 71.4285,Volume= 803105621},
new Data{X= new DateTime(2013,01,21), Open= 72.08, High= 73.57, Low=
62.1428,Close= 62.84,Volume= 971912560},
new Data{X= new DateTime(2013,01,28), Open= 62.5464, High= 66.0857, Low=
62.2657,Close= 64.8028,Volume= 656549587},
new Data{X= new DateTime(2013,02,04), Open= 64.8443, High= 68.4014, Low=
63.1428,Close= 67.8543,Volume= 743778993},
new Data{X= new DateTime(2013,02,11), Open= 68.0714, High= 69.2771, Low=
65.7028,Close= 65.7371,Volume= 585292366},
new Data{X= new DateTime(2013,02,18), Open= 65.8714, High= 66.1043, Low=
63.26,Close= 64.4014,Volume= 421766997},
new Data{X= new DateTime(2013,02,25), Open= 64.8357, High= 65.0171, Low=
61.4257,Close= 61.4957,Volume= 582741215},
new Data{X= new DateTime(2013,03,04), Open= 61.1143, High= 62.2043, Low=
59.8571,Close= 61.6743,Volume= 632856539},
new Data{X= new DateTime(2013,03,11), Open= 61.3928, High= 63.4614, Low=
60.7343,Close= 63.38,Volume= 572066981},
new Data{X= new DateTime(2013,03,18), Open= 63.0643, High= 66.0143, Low=
63.0286,Close= 65.9871,Volume= 552156035},
new Data{X= new DateTime(2013,03,25), Open= 66.3843, High= 67.1357, Low=
63.0886,Close= 63.2371,Volume= 390762517},
new Data{X= new DateTime(2013,04,01), Open= 63.1286, High= 63.3854, Low=
59.9543,Close= 60.4571,Volume= 505273732},
new Data{X= new DateTime(2013,04,08), Open= 60.6928, High= 62.57, Low=
60.3557,Close= 61.4,Volume= 387323550},
new Data{X= new DateTime(2013,04,15), Open= 61, High= 61.1271, Low=
55.0143,Close= 55.79,Volume= 709945604},
new Data{X= new DateTime(2013,04,22), Open= 56.0914, High= 59.8241, Low=
55.8964,Close= 59.6007,Volume= 787007506},
new Data{X= new DateTime(2013,04,29), Open= 60.0643, High= 64.7471, Low=
60,Close= 64.2828,Volume= 655020017},
new Data{X= new DateTime(2013,05,06), Open= 65.1014, High= 66.5357, Low=
64.3543,Close= 64.71,Volume= 545488533},
new Data{X= new DateTime(2013,05,13), Open= 64.5014, High= 65.4143, Low=
59.8428,Close= 61.8943,Volume= 633706550},
new Data{X= new DateTime(2013,05,20), Open= 61.7014, High= 64.05, Low=
61.4428,Close= 63.5928,Volume= 494379068},
new Data{X= new DateTime(2013,05,27), Open= 64.2714, High= 65.3, Low=
62.7714,Close= 64.2478,Volume= 362907830},
new Data{X= new DateTime(2013,06,03), Open= 64.39, High= 64.9186, Low=
61.8243,Close= 63.1158,Volume= 443249793},
new Data{X= new DateTime(2013,06,10), Open= 63.5328, High= 64.1541, Low=
61.2143,Close= 61.4357,Volume= 389680092},
new Data{X= new DateTime(2013,06,17), Open= 61.6343, High= 62.2428, Low=
58.3,Close= 59.0714,Volume= 400384818},
new Data{X= new DateTime(2013,06,24), Open= 58.2, High= 58.38, Low=
55.5528,Close= 56.6471,Volume= 519314826},
new Data{X= new DateTime(2013,07,01), Open= 57.5271, High= 60.47, Low=
57.3171,Close= 59.6314,Volume= 343878841},
new Data{X= new DateTime(2013,07,08), Open= 60.0157, High= 61.3986, Low=
58.6257,Close= 60.93,Volume= 384106977},
```

```
new Data{X= new DateTime(2013,07,15), Open= 60.7157, High= 62.1243, Low=
60.5957,Close= 60.7071,Volume= 286035513},
new Data{X= new DateTime(2013,07,22), Open= 61.3514, High= 63.5128, Low=
59.8157,Close= 62.9986,Volume= 395816827},
new Data{X= new DateTime(2013,07,29), Open= 62.9714, High= 66.1214, Low=
62.8857,Close= 66.0771,Volume= 339668858},
new Data{X= new DateTime(2013,08,12), Open= 65.2657, High= 72.0357, Low=
65.2328,Close= 71.7614,Volume= 711563584},
new Data{X= new DateTime(2013,08,19), Open= 72.0485, High= 73.3914, Low=
71.1714,Close= 71.5743,Volume= 417119660},
new Data{X= new DateTime(2013,08,26), Open= 71.5357, High= 72.8857, Low=
69.4286,Close= 69.6023,Volume= 392805888},
new Data{X= new DateTime(2013,09,02), Open= 70.4428, High= 71.7485, Low=
69.6214,Close= 71.1743,Volume= 317244380},
new Data{X= new DateTime(2013,09,09), Open= 72.1428, High= 72.56, Low=
66.3857,Close= 66.4143,Volume= 669376320},
new Data{X= new DateTime(2013,09,16), Open= 65.8571, High= 68.3643, Low=
63.8886,Close= 66.7728,Volume= 625142677},
new Data{X= new DateTime(2013,09,23), Open= 70.8714, High= 70.9871, Low=
68.6743,Close= 68.9643,Volume= 475274537},
new Data{X= new DateTime(2013,09,30), Open= 68.1786, High= 70.3357, Low=
67.773,Close= 69.0043,Volume= 368198906},
new Data{X= new DateTime(2013,10,07), Open= 69.5086, High= 70.5486, Low=
68.3257,Close= 70.4017,Volume= 361437661},
new Data{X= new DateTime(2013,10,14), Open= 69.9757, High= 72.7514, Low=
69.9071,Close= 72.6985,Volume= 342694379},
new Data{X= new DateTime(2013,10,21), Open= 73.11, High= 76.1757, Low=
72.5757,Close= 75.1368,Volume= 490458997},
new Data{X= new DateTime(2013,10,28), Open= 75.5771, High= 77.0357, Low=
73.5057,Close= 74.29,Volume= 508130174},
new Data{X= new DateTime(2013,11,04), Open= 74.4428, High= 75.555, Low=
73.1971,Close= 74.3657,Volume= 318132218},
new Data{X= new DateTime(2013,11,11), Open= 74.2843, High= 75.6114, Low=
73.4871,Close= 74.9987,Volume= 306711021},
new Data{X= new DateTime(2013,11,18), Open= 74.9985, High= 75.3128, Low=
73.3814,Close= 74.2571,Volume= 282778778},
};
}
```

Exponential Moving Average (EMA)

Moving Average Indicators are used to define the direction of the trend. To render an EMA Indicator, set the indicator [Type](#) as [Ema](#).

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart Title="AAPL - 2012-2017">
  <ChartAxes>
    <ChartAxis Name="secondary" OpposedPosition="true"RowIndex="0" Minimum="-
    7000000000" Maximum="5000000000"
    Title="Accumulation Distribution" Interval="6000000000"></ChartAxis>
  </ChartAxes>
  <ChartPrimaryXAxis ValueType="Syncfusion.Blazor.Charts.ValueType.DateTime"
  Title="months" IntervalType="IntervalType.Months">
    <ChartAxisMajorGridLines Width="0"></ChartAxisMajorGridLines>
    <ChartAxisCrosshairTooltip Enable="true"></ChartAxisCrosshairTooltip>
  </ChartPrimaryXAxis>
  <ChartSeriesCollection>
    <ChartSeries DataSource="@StockDetails" Name="Apple Inc" XName="X" Low="Low"
    High="High" Close="Close" Volume="Volume" Open="Open"
    Width="2" Type="ChartSeriesType.Candle">
    </ChartSeries>
  </ChartSeriesCollection>
  <ChartIndicators>
```

```

<ChartIndicator Type="TechnicalIndicators.Ema"
Field="FinancialDataFields.Close" SeriesName="Apple Inc" Fill="blue"
Period=3>
<ChartIndicatorUpperLine Color="red"></ChartIndicatorUpperLine>
<ChartIndicatorLowerLine Color="green"></ChartIndicatorLowerLine>
</ChartIndicator>
</ChartIndicators>
<ChartLegendSettings Visible="false" />
</SfChart>
@code{
public class Data
{
public DateTime X { get; set; }
public double High { get; set; }
public double Low { get; set; }
public double Open { get; set; }
public double Close { get; set; }
public double Volume { get; set; }
}
public List<Data> StockDetails = new List<Data>
{
new Data{X= new DateTime(2012,10,15), Open= 90.3357, High= 93.2557, Low=
87.0885,Close= 87.12,Volume= 646996264},
new Data{X= new DateTime(2012,10,22), Open= 87.4885, High= 90.7685, Low=
84.4285,Close= 86.2857,Volume= 866040680 },
new Data{X= new DateTime(2012,10,29), Open= 84.9828, High= 86.1428, Low=
82.1071,Close= 82.4,Volume= 367371310},
new Data{X= new DateTime(2012,11,05), Open= 83.3593, High= 84.3914, Low=
76.2457,Close= 78.1514,Volume= 919719846},
new Data{X= new DateTime(2012,11,12), Open= 79.1643, High= 79.2143, Low=
72.25,Close= 75.3825,Volume= 894382149},
new Data{X= new DateTime(2012,11,19), Open= 77.2443, High= 81.7143, Low=
77.1257,Close= 81.6428,Volume= 527416747},
new Data{X= new DateTime(2012,11,26), Open= 82.2714, High= 84.8928, Low=
81.7514,Close= 83.6114,Volume= 646467974},
new Data{X= new DateTime(2012,12,03), Open= 84.8071, High= 84.9414, Low=
74.09,Close= 76.1785,Volume= 980096264},
new Data{X= new DateTime(2012,12,10), Open= 75, High= 78.5085, Low=
72.2257,Close= 72.8277,Volume= 835016110},
new Data{X= new DateTime(2012,12,17), Open= 72.7043, High= 76.4143, Low=
71.6043,Close= 74.19,Volume= 726150329},
new Data{X= new DateTime(2012,12,24), Open= 74.3357, High= 74.8928, Low=
72.0943,Close= 72.7984,Volume= 321104733},
new Data{X= new DateTime(2012,12,31), Open= 72.9328, High= 79.2857, Low=
72.7143,Close= 75.2857,Volume= 540854882},
new Data{X= new DateTime(2013,01,07), Open= 74.5714, High= 75.9843, Low=
73.6,Close= 74.3285,Volume= 574594262},
new Data{X= new DateTime(2013,01,14), Open= 71.8114, High= 72.9643, Low=
69.0543,Close= 71.4285,Volume= 803105621},
new Data{X= new DateTime(2013,01,21), Open= 72.08, High= 73.57, Low=
62.1428,Close= 62.84,Volume= 971912560},
new Data{X= new DateTime(2013,01,28), Open= 62.5464, High= 66.0857, Low=
62.2657,Close= 64.8028,Volume= 656549587},
new Data{X= new DateTime(2013,02,04), Open= 64.8443, High= 68.4014, Low=
63.1428,Close= 67.8543,Volume= 743778993},
new Data{X= new DateTime(2013,02,11), Open= 68.0714, High= 69.2771, Low=
65.7028,Close= 65.7371,Volume= 585292366},

```

```
new Data{X= new DateTime(2013,02,18), Open= 65.8714, High= 66.1043, Low=
63.26,Close= 64.4014,Volume= 421766997},
new Data{X= new DateTime(2013,02,25), Open= 64.8357, High= 65.0171, Low=
61.4257,Close= 61.4957,Volume= 582741215},
new Data{X= new DateTime(2013,03,04), Open= 61.1143, High= 62.2043, Low=
59.8571,Close= 61.6743,Volume= 632856539},
new Data{X= new DateTime(2013,03,11), Open= 61.3928, High= 63.4614, Low=
60.7343,Close= 63.38,Volume= 572066981},
new Data{X= new DateTime(2013,03,18), Open= 63.0643, High= 66.0143, Low=
63.0286,Close= 65.9871,Volume= 552156035},
new Data{X= new DateTime(2013,03,25), Open= 66.3843, High= 67.1357, Low=
63.0886,Close= 63.2371,Volume= 390762517},
new Data{X= new DateTime(2013,04,01), Open= 63.1286, High= 63.3854, Low=
59.9543,Close= 60.4571,Volume= 505273732},
new Data{X= new DateTime(2013,04,08), Open= 60.6928, High= 62.57, Low=
60.3557,Close= 61.4,Volume= 387323550},
new Data{X= new DateTime(2013,04,15), Open= 61, High= 61.1271, Low=
55.0143,Close= 55.79,Volume= 709945604},
new Data{X= new DateTime(2013,04,22), Open= 56.0914, High= 59.8241, Low=
55.8964,Close= 59.6007,Volume= 787007506},
new Data{X= new DateTime(2013,04,29), Open= 60.0643, High= 64.7471, Low=
60,Close= 64.2828,Volume= 655020017},
new Data{X= new DateTime(2013,05,06), Open= 65.1014, High= 66.5357, Low=
64.3543,Close= 64.71,Volume= 545488533},
new Data{X= new DateTime(2013,05,13), Open= 64.5014, High= 65.4143, Low=
59.8428,Close= 61.8943,Volume= 633706550},
new Data{X= new DateTime(2013,05,20), Open= 61.7014, High= 64.05, Low=
61.4428,Close= 63.5928,Volume= 494379068},
new Data{X= new DateTime(2013,05,27), Open= 64.2714, High= 65.3, Low=
62.7714,Close= 64.2478,Volume= 362907830},
new Data{X= new DateTime(2013,06,03), Open= 64.39, High= 64.9186, Low=
61.8243,Close= 63.1158,Volume= 443249793},
new Data{X= new DateTime(2013,06,10), Open= 63.5328, High= 64.1541, Low=
61.2143,Close= 61.4357,Volume= 389680092},
new Data{X= new DateTime(2013,06,17), Open= 61.6343, High= 62.2428, Low=
58.3,Close= 59.0714,Volume= 400384818},
new Data{X= new DateTime(2013,06,24), Open= 58.2, High= 58.38, Low=
55.5528,Close= 56.6471,Volume= 519314826},
new Data{X= new DateTime(2013,07,01), Open= 57.5271, High= 60.47, Low=
57.3171,Close= 59.6314,Volume= 343878841},
new Data{X= new DateTime(2013,07,08), Open= 60.0157, High= 61.3986, Low=
58.6257,Close= 60.93,Volume= 384106977},
new Data{X= new DateTime(2013,07,15), Open= 60.7157, High= 62.1243, Low=
60.5957,Close= 60.7071,Volume= 286035513},
new Data{X= new DateTime(2013,07,22), Open= 61.3514, High= 63.5128, Low=
59.8157,Close= 62.9986,Volume= 395816827},
new Data{X= new DateTime(2013,07,29), Open= 62.9714, High= 66.1214, Low=
62.8857,Close= 66.0771,Volume= 339668858},
new Data{X= new DateTime(2013,08,12), Open= 65.2657, High= 72.0357, Low=
65.2328,Close= 71.7614,Volume= 711563584},
new Data{X= new DateTime(2013,08,19), Open= 72.0485, High= 73.3914, Low=
71.1714,Close= 71.5743,Volume= 417119660},
new Data{X= new DateTime(2013,08,26), Open= 71.5357, High= 72.8857, Low=
69.4286,Close= 69.6023,Volume= 392805888},
new Data{X= new DateTime(2013,09,02), Open= 70.4428, High= 71.7485, Low=
69.6214,Close= 71.1743,Volume= 317244380},
```

```
new Data{X= new DateTime(2013,09,09), Open= 72.1428, High= 72.56, Low=
66.3857,Close= 66.4143,Volume= 669376320},
new Data{X= new DateTime(2013,09,16), Open= 65.8571, High= 68.3643, Low=
63.8886,Close= 66.7728,Volume= 625142677},
new Data{X= new DateTime(2013,09,23), Open= 70.8714, High= 70.9871, Low=
68.6743,Close= 68.9643,Volume= 475274537},
new Data{X= new DateTime(2013,09,30), Open= 68.1786, High= 70.3357, Low=
67.773,Close= 69.0043,Volume= 368198906},
new Data{X= new DateTime(2013,10,07), Open= 69.5086, High= 70.5486, Low=
68.3257,Close= 70.4017,Volume= 361437661},
new Data{X= new DateTime(2013,10,14), Open= 69.9757, High= 72.7514, Low=
69.9071,Close= 72.6985,Volume= 342694379},
new Data{X= new DateTime(2013,10,21), Open= 73.11, High= 76.1757, Low=
72.5757,Close= 75.1368,Volume= 490458997},
new Data{X= new DateTime(2013,10,28), Open= 75.5771, High= 77.0357, Low=
73.5057,Close= 74.29,Volume= 508130174},
new Data{X= new DateTime(2013,11,04), Open= 74.4428, High= 75.555, Low=
73.1971,Close= 74.3657,Volume= 318132218},
new Data{X= new DateTime(2013,11,11), Open= 74.2843, High= 75.6114, Low=
73.4871,Close= 74.9987,Volume= 306711021},
new Data{X= new DateTime(2013,11,18), Open= 74.9985, High= 75.3128, Low=
73.3814,Close= 74.2571,Volume= 282778778},
};
}
```

AAPL - 2012-2017



Momentum

Momentum shows the speed at which the price of the stock is changing. To render a Momentum Indicator, set the indicator [Type](#) as [Momentum](#). Momentum Indicator will be represented by two lines - upper line and signal line. In momentum indicator, the upper band is always render at the value **100**.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart Title="AAPL - 2012-2017">
  <ChartPrimaryXAxis ValueType="Syncfusion.Blazor.Charts.ValueType.DateTime"
    Title="months" IntervalType="IntervalType.Months">
  <ChartAxisMajorGridLines Width="0"></ChartAxisMajorGridLines>
  <ChartAxisCrosshairTooltip Enable="true"></ChartAxisCrosshairTooltip>
</ChartPrimaryXAxis>
  <ChartPrimaryYAxis Title="Price" LabelFormat="{value}" Minimum="50"
    Maximum="170" PlotOffset="25" Interval="30" RowIndex="1"
    OpposedPosition="true">
  <ChartAxisMajorTickLines Width="0"></ChartAxisMajorTickLines>
  <ChartAxisLineStyle Width="0"></ChartAxisLineStyle>
</ChartPrimaryYAxis>
  <ChartAxes>
  <ChartAxis Name="secondary" OpposedPosition="true" RowIndex="0"
    Interval="20"
    Minimum="80" Maximum="120" Title="Momentum">
  <ChartAxisLineStyle Width="0"></ChartAxisLineStyle>
  <ChartAxisMajorGridLines Width="0"></ChartAxisMajorGridLines>
  <ChartAxisMajorTickLines Width="0"></ChartAxisMajorTickLines>
</ChartAxis>
</ChartAxes>
  <ChartRows>
  <ChartRow Height="40%"></ChartRow>
  <ChartRow Height="60%"></ChartRow>
</ChartRows>
  <ChartSeriesCollection>
  <ChartSeries DataSource="@StockDetails" XName="X" YName="Y" Low="Low"
    High="High" Close="Close" Volume="Volume" Open="Open"
    Width="2" Name="Apple Inc" Type="ChartSeriesType.Candle">
  </ChartSeries>
</ChartSeriesCollection>
  <ChartIndicators>
  <ChartIndicator Type="TechnicalIndicators.Momentum"
    Field="FinancialDataFields.Close" SeriesName="Apple Inc" Fill="blue"
    Period=3 YAxisName="secondary">
  </ChartIndicator>
</ChartIndicators>
  <ChartLegendSettings Visible="false" />
</SfChart>

@code{
public class Data
{
  public DateTime X { get; set; }
  public double High { get; set; }
  public double Low { get; set; }
  public double Open { get; set; }
  public double Close { get; set; }
  public double Volume { get; set; }
}
```

```

public List<Data> StockDetails = new List<Data>
{
    new Data{X= new DateTime(2012,10,15), Open= 90.3357, High= 93.2557, Low=
    87.0885,Close= 87.12,Volume= 646996264},
    new Data{X= new DateTime(2012,10,22), Open= 87.4885, High= 90.7685, Low=
    84.4285,Close= 86.2857,Volume= 866040680 },
    new Data{X= new DateTime(2012,10,29), Open= 84.9828, High= 86.1428, Low=
    82.1071,Close= 82.4,Volume= 367371310},
    new Data{X= new DateTime(2012,11,05), Open= 83.3593, High= 84.3914, Low=
    76.2457,Close= 78.1514,Volume= 919719846},
    new Data{X= new DateTime(2012,11,12), Open= 79.1643, High= 79.2143, Low=
    72.25,Close= 75.3825,Volume= 894382149},
    new Data{X= new DateTime(2012,11,19), Open= 77.2443, High= 81.7143, Low=
    77.1257,Close= 81.6428,Volume= 527416747},
    new Data{X= new DateTime(2012,11,26), Open= 82.2714, High= 84.8928, Low=
    81.7514,Close= 83.6114,Volume= 646467974},
    new Data{X= new DateTime(2012,12,03), Open= 84.8071, High= 84.9414, Low=
    74.09,Close= 76.1785,Volume= 980096264},
    new Data{X= new DateTime(2012,12,10), Open= 75, High= 78.5085, Low=
    72.2257,Close= 72.8277,Volume= 835016110},
    new Data{X= new DateTime(2012,12,17), Open= 72.7043, High= 76.4143, Low=
    71.6043,Close= 74.19,Volume= 726150329},
    new Data{X= new DateTime(2012,12,24), Open= 74.3357, High= 74.8928, Low=
    72.0943,Close= 72.7984,Volume= 321104733},
    new Data{X= new DateTime(2012,12,31), Open= 72.9328, High= 79.2857, Low=
    72.7143,Close= 75.2857,Volume= 540854882},
    new Data{X= new DateTime(2013,01,07), Open= 74.5714, High= 75.9843, Low=
    73.6,Close= 74.3285,Volume= 574594262},
    new Data{X= new DateTime(2013,01,14), Open= 71.8114, High= 72.9643, Low=
    69.0543,Close= 71.4285,Volume= 803105621},
    new Data{X= new DateTime(2013,01,21), Open= 72.08, High= 73.57, Low=
    62.1428,Close= 62.84,Volume= 971912560},
    new Data{X= new DateTime(2013,01,28), Open= 62.5464, High= 66.0857, Low=
    62.2657,Close= 64.8028,Volume= 656549587},
    new Data{X= new DateTime(2013,02,04), Open= 64.8443, High= 68.4014, Low=
    63.1428,Close= 67.8543,Volume= 743778993},
    new Data{X= new DateTime(2013,02,11), Open= 68.0714, High= 69.2771, Low=
    65.7028,Close= 65.7371,Volume= 585292366},
    new Data{X= new DateTime(2013,02,18), Open= 65.8714, High= 66.1043, Low=
    63.26,Close= 64.4014,Volume= 421766997},
    new Data{X= new DateTime(2013,02,25), Open= 64.8357, High= 65.0171, Low=
    61.4257,Close= 61.4957,Volume= 582741215},
    new Data{X= new DateTime(2013,03,04), Open= 61.1143, High= 62.2043, Low=
    59.8571,Close= 61.6743,Volume= 632856539},
    new Data{X= new DateTime(2013,03,11), Open= 61.3928, High= 63.4614, Low=
    60.7343,Close= 63.38,Volume= 572066981},
    new Data{X= new DateTime(2013,03,18), Open= 63.0643, High= 66.0143, Low=
    63.0286,Close= 65.9871,Volume= 552156035},
    new Data{X= new DateTime(2013,03,25), Open= 66.3843, High= 67.1357, Low=
    63.0886,Close= 63.2371,Volume= 390762517},
    new Data{X= new DateTime(2013,04,01), Open= 63.1286, High= 63.3854, Low=
    59.9543,Close= 60.4571,Volume= 505273732},
    new Data{X= new DateTime(2013,04,08), Open= 60.6928, High= 62.57, Low=
    60.3557,Close= 61.4,Volume= 387323550},
    new Data{X= new DateTime(2013,04,15), Open= 61, High= 61.1271, Low=
    55.0143,Close= 55.79,Volume= 709945604},
}

```

```
new Data{X= new DateTime(2013,04,22), Open= 56.0914, High= 59.8241, Low=
55.8964,Close= 59.6007,Volume= 787007506},
new Data{X= new DateTime(2013,04,29), Open= 60.0643, High= 64.7471, Low=
60,Close= 64.2828,Volume= 655020017},
new Data{X= new DateTime(2013,05,06), Open= 65.1014, High= 66.5357, Low=
64.3543,Close= 64.71,Volume= 545488533},
new Data{X= new DateTime(2013,05,13), Open= 64.5014, High= 65.4143, Low=
59.8428,Close= 61.8943,Volume= 633706550},
new Data{X= new DateTime(2013,05,20), Open= 61.7014, High= 64.05, Low=
61.4428,Close= 63.5928,Volume= 494379068},
new Data{X= new DateTime(2013,05,27), Open= 64.2714, High= 65.3, Low=
62.7714,Close= 64.2478,Volume= 362907830},
new Data{X= new DateTime(2013,06,03), Open= 64.39, High= 64.9186, Low=
61.8243,Close= 63.1158,Volume= 443249793},
new Data{X= new DateTime(2013,06,10), Open= 63.5328, High= 64.1541, Low=
61.2143,Close= 61.4357,Volume= 389680092},
new Data{X= new DateTime(2013,06,17), Open= 61.6343, High= 62.2428, Low=
58.3,Close= 59.0714,Volume= 400384818},
new Data{X= new DateTime(2013,06,24), Open= 58.2, High= 58.38, Low=
55.5528,Close= 56.6471,Volume= 519314826},
new Data{X= new DateTime(2013,07,01), Open= 57.5271, High= 60.47, Low=
57.3171,Close= 59.6314,Volume= 343878841},
new Data{X= new DateTime(2013,07,08), Open= 60.0157, High= 61.3986, Low=
58.6257,Close= 60.93,Volume= 384106977},
new Data{X= new DateTime(2013,07,15), Open= 60.7157, High= 62.1243, Low=
60.5957,Close= 60.7071,Volume= 286035513},
new Data{X= new DateTime(2013,07,22), Open= 61.3514, High= 63.5128, Low=
59.8157,Close= 62.9986,Volume= 395816827},
new Data{X= new DateTime(2013,07,29), Open= 62.9714, High= 66.1214, Low=
62.8857,Close= 66.0771,Volume= 339668858},
new Data{X= new DateTime(2013,08,12), Open= 65.2657, High= 72.0357, Low=
65.2328,Close= 71.7614,Volume= 711563584},
new Data{X= new DateTime(2013,08,19), Open= 72.0485, High= 73.3914, Low=
71.1714,Close= 71.5743,Volume= 417119660},
new Data{X= new DateTime(2013,08,26), Open= 71.5357, High= 72.8857, Low=
69.4286,Close= 69.6023,Volume= 392805888},
new Data{X= new DateTime(2013,09,02), Open= 70.4428, High= 71.7485, Low=
69.6214,Close= 71.1743,Volume= 317244380},
new Data{X= new DateTime(2013,09,09), Open= 72.1428, High= 72.56, Low=
66.3857,Close= 66.4143,Volume= 669376320},
new Data{X= new DateTime(2013,09,16), Open= 65.8571, High= 68.3643, Low=
63.8886,Close= 66.7728,Volume= 625142677},
new Data{X= new DateTime(2013,09,23), Open= 70.8714, High= 70.9871, Low=
68.6743,Close= 68.9643,Volume= 475274537},
new Data{X= new DateTime(2013,09,30), Open= 68.1786, High= 70.3357, Low=
67.773,Close= 69.0043,Volume= 368198906},
new Data{X= new DateTime(2013,10,07), Open= 69.5086, High= 70.5486, Low=
68.3257,Close= 70.4017,Volume= 361437661},
new Data{X= new DateTime(2013,10,14), Open= 69.9757, High= 72.7514, Low=
69.9071,Close= 72.6985,Volume= 342694379},
new Data{X= new DateTime(2013,10,21), Open= 73.11, High= 76.1757, Low=
72.5757,Close= 75.1368,Volume= 490458997},
new Data{X= new DateTime(2013,10,28), Open= 75.5771, High= 77.0357, Low=
73.5057,Close= 74.29,Volume= 508130174},
new Data{X= new DateTime(2013,11,04), Open= 74.4428, High= 75.555, Low=
73.1971,Close= 74.3657,Volume= 318132218},
```

```
new Data{X= new DateTime(2013,11,11), Open= 74.2843, High= 75.6114, Low=
73.4871,Close= 74.9987,Volume= 306711021},
new Data{X= new DateTime(2013,11,18), Open= 74.9985, High= 75.3128, Low=
73.3814,Close= 74.2571,Volume= 282778778},
};
}
```

AAPL - 2012-2017



Customization of Momentum Indicator

The upper line's [Width](#) and [Color](#) can be customized using the [UpperLine](#) property of the indicator.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart Title="AAPL - 2012-2017">
  <ChartPrimaryXAxis Value="Syncfusion.Blazor.Charts.ValueType.DateTime"
  Title="months" IntervalType="IntervalType.Months">
    <ChartAxisMajorGridLines Width="0"></ChartAxisMajorGridLines>
    <ChartAxisCrosshairTooltip Enable="true"></ChartAxisCrosshairTooltip>
  </ChartPrimaryXAxis>
  <ChartPrimaryYAxis Title="Price" LabelFormat="{value}" Minimum="50"
  Maximum="170" PlotOffset="25" Interval="30" RowIndex="1"
  OpposedPosition="true">
    <ChartAxisMajorTickLines Width="0"></ChartAxisMajorTickLines>
    <ChartAxisLineStyle Width="0"></ChartAxisLineStyle>
  </ChartPrimaryYAxis>
```



```

<ChartAxes>
<ChartAxis Name="secondary" OpposedPosition="true"RowIndex="0"
Interval="20"
Minimum="80" Maximum="120" Title="Momentum">
<ChartAxisLineStyle Width="0"></ChartAxisLineStyle>
<ChartAxisMajorGridLines Width="0"></ChartAxisMajorGridLines>
<ChartAxisMajorTickLines Width="0"></ChartAxisMajorTickLines>
</ChartAxis>
</ChartAxes>
<ChartRows>
<ChartRow Height="40%"></ChartRow>
<ChartRow Height="60%"></ChartRow>
</ChartRows>
<ChartSeriesCollection>
<ChartSeries DataSource="@StockDetails" XName="X" YName="Y" Low="Low"
High="High" Close="Close" Volume="Volume" Open="Open"
Width="2" Name="Apple Inc" Type="ChartSeriesType.Candle">
</ChartSeries>
</ChartSeriesCollection>
<ChartIndicators>
<ChartIndicator Type="TechnicalIndicators.Momentum"
Field="FinancialDataFields.Close" SeriesName="Apple Inc" Fill="blue"
Period=3 YAxisName="secondary">
<ChartIndicatorUpperLine Color="green" Width="3"></ChartIndicatorUpperLine>
</ChartIndicator>
</ChartIndicators>
<ChartLegendSettings Visible="false" />
</SfChart>
@code{
public class Data
{
public DateTime X { get; set; }
public double High { get; set; }
public double Low { get; set; }
public double Open { get; set; }
public double Close { get; set; }
public double Volume { get; set; }
}
public List<Data> StockDetails = new List<Data>
{
new Data{X= new DateTime(2012,10,15), Open= 90.3357, High= 93.2557, Low=
87.0885,Close= 87.12,Volume=646996264},
new Data{X= new DateTime(2012,10,22), Open= 87.4885, High= 90.7685, Low=
84.4285,Close= 86.2857,Volume= 866040680 },
new Data{X= new DateTime(2012,10,29), Open= 84.9828, High= 86.1428, Low=
82.1071,Close= 82.4,Volume= 367371310},
new Data{X= new DateTime(2012,11,05), Open= 83.3593, High= 84.3914, Low=
76.2457,Close= 78.1514,Volume= 919719846},
new Data{X= new DateTime(2012,11,12), Open= 79.1643, High= 79.2143, Low=
72.25,Close= 75.3825,Volume= 894382149},
new Data{X= new DateTime(2012,11,19), Open= 77.2443, High= 81.7143, Low=
77.1257,Close= 81.6428,Volume= 527416747},
new Data{X= new DateTime(2012,11,26), Open= 82.2714, High= 84.8928, Low=
81.7514,Close= 83.6114,Volume= 646467974},
new Data{X= new DateTime(2012,12,03), Open= 84.8071, High= 84.9414, Low=
74.09,Close= 76.1785,Volume= 980096264},

```

```
new Data{X= new DateTime(2012,12,10), Open= 75, High= 78.5085, Low=
72.2257,Close= 72.8277,Volume= 835016110},
new Data{X= new DateTime(2012,12,17), Open= 72.7043, High= 76.4143, Low=
71.6043,Close= 74.19,Volume= 726150329},
new Data{X= new DateTime(2012,12,24), Open= 74.3357, High= 74.8928, Low=
72.0943,Close= 72.7984,Volume= 321104733},
new Data{X= new DateTime(2012,12,31), Open= 72.9328, High= 79.2857, Low=
72.7143,Close= 75.2857,Volume= 540854882},
new Data{X= new DateTime(2013,01,07), Open= 74.5714, High= 75.9843, Low=
73.6,Close= 74.3285,Volume= 574594262},
new Data{X= new DateTime(2013,01,14), Open= 71.8114, High= 72.9643, Low=
69.0543,Close= 71.4285,Volume= 803105621},
new Data{X= new DateTime(2013,01,21), Open= 72.08, High= 73.57, Low=
62.1428,Close= 62.84,Volume= 971912560},
new Data{X= new DateTime(2013,01,28), Open= 62.5464, High= 66.0857, Low=
62.2657,Close= 64.8028,Volume= 656549587},
new Data{X= new DateTime(2013,02,04), Open= 64.8443, High= 68.4014, Low=
63.1428,Close= 67.8543,Volume= 743778993},
new Data{X= new DateTime(2013,02,11), Open= 68.0714, High= 69.2771, Low=
65.7028,Close= 65.7371,Volume= 585292366},
new Data{X= new DateTime(2013,02,18), Open= 65.8714, High= 66.1043, Low=
63.26,Close= 64.4014,Volume= 421766997},
new Data{X= new DateTime(2013,02,25), Open= 64.8357, High= 65.0171, Low=
61.4257,Close= 61.4957,Volume= 582741215},
new Data{X= new DateTime(2013,03,04), Open= 61.1143, High= 62.2043, Low=
59.8571,Close= 61.6743,Volume= 632856539},
new Data{X= new DateTime(2013,03,11), Open= 61.3928, High= 63.4614, Low=
60.7343,Close= 63.38,Volume= 572066981},
new Data{X= new DateTime(2013,03,18), Open= 63.0643, High= 66.0143, Low=
63.0286,Close= 65.9871,Volume= 552156035},
new Data{X= new DateTime(2013,03,25), Open= 66.3843, High= 67.1357, Low=
63.0886,Close= 63.2371,Volume= 390762517},
new Data{X= new DateTime(2013,04,01), Open= 63.1286, High= 63.3854, Low=
59.9543,Close= 60.4571,Volume= 505273732},
new Data{X= new DateTime(2013,04,08), Open= 60.6928, High= 62.57, Low=
60.3557,Close= 61.4,Volume= 387323550},
new Data{X= new DateTime(2013,04,15), Open= 61, High= 61.1271, Low=
55.0143,Close= 55.79,Volume= 709945604},
new Data{X= new DateTime(2013,04,22), Open= 56.0914, High= 59.8241, Low=
55.8964,Close= 59.6007,Volume= 787007506},
new Data{X= new DateTime(2013,04,29), Open= 60.0643, High= 64.7471, Low=
60,Close= 64.2828,Volume= 655020017},
new Data{X= new DateTime(2013,05,06), Open= 65.1014, High= 66.5357, Low=
64.3543,Close= 64.71,Volume= 545488533},
new Data{X= new DateTime(2013,05,13), Open= 64.5014, High= 65.4143, Low=
59.8428,Close= 61.8943,Volume= 633706550},
new Data{X= new DateTime(2013,05,20), Open= 61.7014, High= 64.05, Low=
61.4428,Close= 63.5928,Volume= 494379068},
new Data{X= new DateTime(2013,05,27), Open= 64.2714, High= 65.3, Low=
62.7714,Close= 64.2478,Volume= 362907830},
new Data{X= new DateTime(2013,06,03), Open= 64.39, High= 64.9186, Low=
61.8243,Close= 63.1158,Volume= 443249793},
new Data{X= new DateTime(2013,06,10), Open= 63.5328, High= 64.1541, Low=
61.2143,Close= 61.4357,Volume= 389680092},
new Data{X= new DateTime(2013,06,17), Open= 61.6343, High= 62.2428, Low=
58.3,Close= 59.0714,Volume= 400384818},
```

```
new Data{X= new DateTime(2013,06,24), Open= 58.2, High= 58.38, Low=
55.5528,Close= 56.6471,Volume= 519314826},
new Data{X= new DateTime(2013,07,01), Open= 57.5271, High= 60.47, Low=
57.3171,Close= 59.6314,Volume= 343878841},
new Data{X= new DateTime(2013,07,08), Open= 60.0157, High= 61.3986, Low=
58.6257,Close= 60.93,Volume= 384106977},
new Data{X= new DateTime(2013,07,15), Open= 60.7157, High= 62.1243, Low=
60.5957,Close= 60.7071,Volume= 286035513},
new Data{X= new DateTime(2013,07,22), Open= 61.3514, High= 63.5128, Low=
59.8157,Close= 62.9986,Volume= 395816827},
new Data{X= new DateTime(2013,07,29), Open= 62.9714, High= 66.1214, Low=
62.8857,Close= 66.0771,Volume= 339668858},
new Data{X= new DateTime(2013,08,12), Open= 65.2657, High= 72.0357, Low=
65.2328,Close= 71.7614,Volume= 711563584},
new Data{X= new DateTime(2013,08,19), Open= 72.0485, High= 73.3914, Low=
71.1714,Close= 71.5743,Volume= 417119660},
new Data{X= new DateTime(2013,08,26), Open= 71.5357, High= 72.8857, Low=
69.4286,Close= 69.6023,Volume= 392805888},
new Data{X= new DateTime(2013,09,02), Open= 70.4428, High= 71.7485, Low=
69.6214,Close= 71.1743,Volume= 317244380},
new Data{X= new DateTime(2013,09,09), Open= 72.1428, High= 72.56, Low=
66.3857,Close= 66.4143,Volume= 669376320},
new Data{X= new DateTime(2013,09,16), Open= 65.8571, High= 68.3643, Low=
63.8886,Close= 66.7728,Volume= 625142677},
new Data{X= new DateTime(2013,09,23), Open= 70.8714, High= 70.9871, Low=
68.6743,Close= 68.9643,Volume= 475274537},
new Data{X= new DateTime(2013,09,30), Open= 68.1786, High= 70.3357, Low=
67.773,Close= 69.0043,Volume= 368198906},
new Data{X= new DateTime(2013,10,07), Open= 69.5086, High= 70.5486, Low=
68.3257,Close= 70.4017,Volume= 361437661},
new Data{X= new DateTime(2013,10,14), Open= 69.9757, High= 72.7514, Low=
69.9071,Close= 72.6985,Volume= 342694379},
new Data{X= new DateTime(2013,10,21), Open= 73.11, High= 76.1757, Low=
72.5757,Close= 75.1368,Volume= 490458997},
new Data{X= new DateTime(2013,10,28), Open= 75.5771, High= 77.0357, Low=
73.5057,Close= 74.29,Volume= 508130174},
new Data{X= new DateTime(2013,11,04), Open= 74.4428, High= 75.555, Low=
73.1971,Close= 74.3657,Volume= 318132218},
new Data{X= new DateTime(2013,11,11), Open= 74.2843, High= 75.6114, Low=
73.4871,Close= 74.9987,Volume= 306711021},
new Data{X= new DateTime(2013,11,18), Open= 74.9985, High= 75.3128, Low=
73.3814,Close= 74.2571,Volume= 282778778},
};
}
```



Moving Average Convergence Divergence (MACD)

Moving Average Convergence Divergence (MACD) is based on the difference between two EMA's. To render a MACD Indicator, set the indicator [Type](#) as [Macd](#). MACD Indicator will be represented by MACD line, signal line and MACD histogram. MACD histogram is used to differentiate MACD line and signal line.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart Title="AAPL - 2012-2017">
  <ChartPrimaryXAxis Value="Syncfusion.Blazor.Charts.ValueType.DateTime"
    Title="months" IntervalType="IntervalType.Months">
  <ChartAxisMajorGridLines Width="0"></ChartAxisMajorGridLines>
  <ChartAxisCrosshairTooltip Enable="true"></ChartAxisCrosshairTooltip>
</ChartPrimaryXAxis>
  <ChartPrimaryYAxis Title="Price" LabelFormat="{value}" Minimum="50"
    Maximum="170" PlotOffset="25" Interval="30" RowIndex="1"
    OpposedPosition="true">
  <ChartAxisMajorTickLines Width="0"></ChartAxisMajorTickLines>
  <ChartAxisLineStyle Width="0"></ChartAxisLineStyle>
</ChartPrimaryYAxis>
  <ChartAxes>
  <ChartAxis Name="secondary" OpposedPosition="true" RowIndex="0" Interval="3"
    Minimum="-3.5" Maximum="3.5" Title="MACD">
  <ChartAxisLineStyle Width="0"></ChartAxisLineStyle>
  <ChartAxisMajorGridLines Width="0"></ChartAxisMajorGridLines>
  <ChartAxisMajorTickLines Width="0"></ChartAxisMajorTickLines>
```

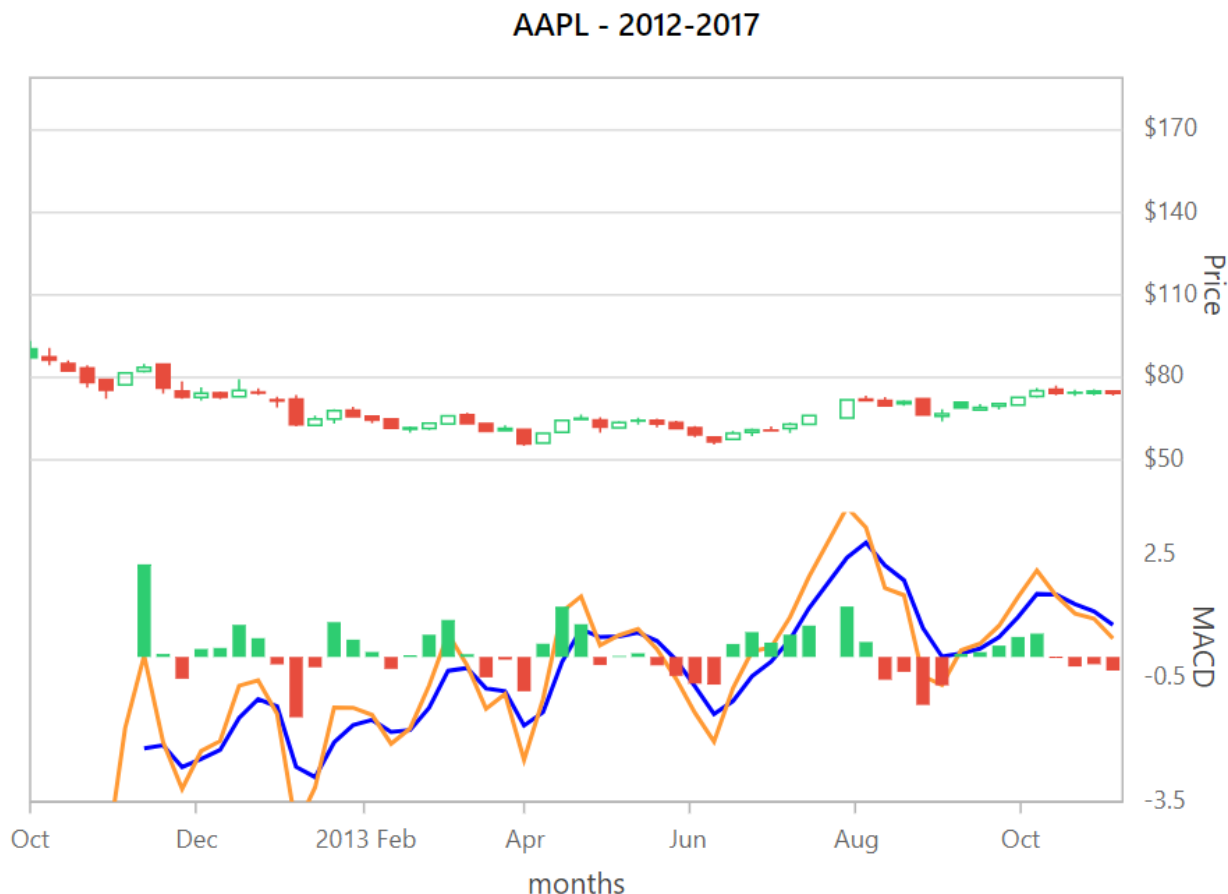
```

</ChartAxis>
</ChartAxes>
<ChartRows>
<ChartRow Height="40%"></ChartRow>
<ChartRow Height="60%"></ChartRow>
</ChartRows>
<ChartSeriesCollection>
<ChartSeries DataSource="@StockDetails" XName="X" YName="Y" Low="Low"
High="High" Close="Close" Volume="Volume" Open="Open"
Width="2" Name="Apple Inc" Type="ChartSeriesType.Candle">
</ChartSeries>
</ChartSeriesCollection>
<ChartIndicators>
<ChartIndicator Type="TechnicalIndicators.Macd" MacdType="MacdType.Both"
SeriesName="Apple Inc" Fill="blue" Period=3 YAxisName="secondary"
FastPeriod="5" SlowPeriod="2" Width="2">
</ChartIndicator>
</ChartIndicators>
<ChartLegendSettings Visible="false" />
</SfChart>
@code{
public class Data
{
public DateTime X { get; set; }
public double High { get; set; }
public double Low { get; set; }
public double Open { get; set; }
public double Close { get; set; }
public double Volume { get; set; }
}
public List<Data> StockDetails = new List<Data>
{
new Data{X= new DateTime(2012,10,15), Open= 90.3357, High= 93.2557, Low=
87.0885,Close= 87.12,Volume= 646996264},
new Data{X= new DateTime(2012,10,22), Open= 87.4885, High= 90.7685, Low=
84.4285,Close= 86.2857,Volume= 866040680 },
new Data{X= new DateTime(2012,10,29), Open= 84.9828, High= 86.1428, Low=
82.1071,Close= 82.4,Volume= 367371310},
new Data{X= new DateTime(2012,11,05), Open= 83.3593, High= 84.3914, Low=
76.2457,Close= 78.1514,Volume= 919719846},
new Data{X= new DateTime(2012,11,12), Open= 79.1643, High= 79.2143, Low=
72.25,Close= 75.3825,Volume=894382149},
new Data{X= new DateTime(2012,11,19), Open= 77.2443, High= 81.7143, Low=
77.1257,Close= 81.6428,Volume= 527416747},
new Data{X= new DateTime(2012,11,26), Open= 82.2714, High= 84.8928, Low=
81.7514,Close= 83.6114,Volume= 646467974},
new Data{X= new DateTime(2012,12,03), Open= 84.8071, High= 84.9414, Low=
74.09,Close= 76.1785,Volume= 980096264},
new Data{X= new DateTime(2012,12,10), Open= 75, High= 78.5085, Low=
72.2257,Close= 72.8277,Volume= 835016110},
new Data{X= new DateTime(2012,12,17), Open= 72.7043, High= 76.4143, Low=
71.6043,Close= 74.19,Volume= 726150329},
new Data{X= new DateTime(2012,12,24), Open= 74.3357, High= 74.8928, Low=
72.0943,Close= 72.7984,Volume= 321104733},
new Data{X= new DateTime(2012,12,31), Open= 72.9328, High= 79.2857, Low=
72.7143,Close= 75.2857,Volume= 540854882},

```

```
new Data{X= new DateTime(2013,01,07), Open= 74.5714, High= 75.9843, Low=
73.6,Close= 74.3285,Volume= 574594262},
new Data{X= new DateTime(2013,01,14), Open= 71.8114, High= 72.9643, Low=
69.0543,Close= 71.4285,Volume= 803105621},
new Data{X= new DateTime(2013,01,21), Open= 72.08, High= 73.57, Low=
62.1428,Close= 62.84,Volume= 971912560},
new Data{X= new DateTime(2013,01,28), Open= 62.5464, High= 66.0857, Low=
62.2657,Close= 64.8028,Volume= 656549587},
new Data{X= new DateTime(2013,02,04), Open= 64.8443, High= 68.4014, Low=
63.1428,Close= 67.8543,Volume= 743778993},
new Data{X= new DateTime(2013,02,11), Open= 68.0714, High= 69.2771, Low=
65.7028,Close= 65.7371,Volume= 585292366},
new Data{X= new DateTime(2013,02,18), Open= 65.8714, High= 66.1043, Low=
63.26,Close= 64.4014,Volume= 421766997},
new Data{X= new DateTime(2013,02,25), Open= 64.8357, High= 65.0171, Low=
61.4257,Close= 61.4957,Volume= 582741215},
new Data{X= new DateTime(2013,03,04), Open= 61.1143, High= 62.2043, Low=
59.8571,Close= 61.6743,Volume= 632856539},
new Data{X= new DateTime(2013,03,11), Open= 61.3928, High= 63.4614, Low=
60.7343,Close= 63.38,Volume= 572066981},
new Data{X= new DateTime(2013,03,18), Open= 63.0643, High= 66.0143, Low=
63.0286,Close= 65.9871,Volume= 552156035},
new Data{X= new DateTime(2013,03,25), Open= 66.3843, High= 67.1357, Low=
63.0886,Close= 63.2371,Volume= 390762517},
new Data{X= new DateTime(2013,04,01), Open= 63.1286, High= 63.3854, Low=
59.9543,Close= 60.4571,Volume= 505273732},
new Data{X= new DateTime(2013,04,08), Open= 60.6928, High= 62.57, Low=
60.3557,Close= 61.4,Volume= 387323550},
new Data{X= new DateTime(2013,04,15), Open= 61, High= 61.1271, Low=
55.0143,Close= 55.79,Volume= 709945604},
new Data{X= new DateTime(2013,04,22), Open= 56.0914, High= 59.8241, Low=
55.8964,Close= 59.6007,Volume= 787007506},
new Data{X= new DateTime(2013,04,29), Open= 60.0643, High= 64.7471, Low=
60,Close= 64.2828,Volume= 655020017},
new Data{X= new DateTime(2013,05,06), Open= 65.1014, High= 66.5357, Low=
64.3543,Close= 64.71,Volume= 545488533},
new Data{X= new DateTime(2013,05,13), Open= 64.5014, High= 65.4143, Low=
59.8428,Close= 61.8943,Volume= 633706550},
new Data{X= new DateTime(2013,05,20), Open= 61.7014, High= 64.05, Low=
61.4428,Close= 63.5928,Volume= 494379068},
new Data{X= new DateTime(2013,05,27), Open= 64.2714, High= 65.3, Low=
62.7714,Close= 64.2478,Volume= 362907830},
new Data{X= new DateTime(2013,06,03), Open= 64.39, High= 64.9186, Low=
61.8243,Close= 63.1158,Volume= 443249793},
new Data{X= new DateTime(2013,06,10), Open= 63.5328, High= 64.1541, Low=
61.2143,Close= 61.4357,Volume= 389680092},
new Data{X= new DateTime(2013,06,17), Open= 61.6343, High= 62.2428, Low=
58.3,Close= 59.0714,Volume= 400384818},
new Data{X= new DateTime(2013,06,24), Open= 58.2, High= 58.38, Low=
55.5528,Close= 56.6471,Volume= 519314826},
new Data{X= new DateTime(2013,07,01), Open= 57.5271, High= 60.47, Low=
57.3171,Close= 59.6314,Volume= 343878841},
new Data{X= new DateTime(2013,07,08), Open= 60.0157, High= 61.3986, Low=
58.6257,Close= 60.93,Volume= 384106977},
new Data{X= new DateTime(2013,07,15), Open= 60.7157, High= 62.1243, Low=
60.5957,Close= 60.7071,Volume= 286035513},
```

```
new Data{X= new DateTime(2013,07,22), Open= 61.3514, High= 63.5128, Low=
59.8157,Close= 62.9986,Volume= 395816827},
new Data{X= new DateTime(2013,07,29), Open= 62.9714, High= 66.1214, Low=
62.8857,Close= 66.0771,Volume= 339668858},
new Data{X= new DateTime(2013,08,12), Open= 65.2657, High= 72.0357, Low=
65.2328,Close= 71.7614,Volume= 711563584},
new Data{X= new DateTime(2013,08,19), Open= 72.0485, High= 73.3914, Low=
71.1714,Close= 71.5743,Volume= 417119660},
new Data{X= new DateTime(2013,08,26), Open= 71.5357, High= 72.8857, Low=
69.4286,Close= 69.6023,Volume= 392805888},
new Data{X= new DateTime(2013,09,02), Open= 70.4428, High= 71.7485, Low=
69.6214,Close= 71.1743,Volume= 317244380},
new Data{X= new DateTime(2013,09,09), Open= 72.1428, High= 72.56, Low=
66.3857,Close= 66.4143,Volume= 669376320},
new Data{X= new DateTime(2013,09,16), Open= 65.8571, High= 68.3643, Low=
63.8886,Close= 66.7728,Volume= 625142677},
new Data{X= new DateTime(2013,09,23), Open= 70.8714, High= 70.9871, Low=
68.6743,Close= 68.9643,Volume= 475274537},
new Data{X= new DateTime(2013,09,30), Open= 68.1786, High= 70.3357, Low=
67.773,Close= 69.0043,Volume= 368198906},
new Data{X= new DateTime(2013,10,07), Open= 69.5086, High= 70.5486, Low=
68.3257,Close= 70.4017,Volume= 361437661},
new Data{X= new DateTime(2013,10,14), Open= 69.9757, High= 72.7514, Low=
69.9071,Close= 72.6985,Volume= 342694379},
new Data{X= new DateTime(2013,10,21), Open= 73.11, High= 76.1757, Low=
72.5757,Close= 75.1368,Volume= 490458997},
new Data{X= new DateTime(2013,10,28), Open= 75.5771, High= 77.0357, Low=
73.5057,Close= 74.29,Volume= 508130174},
new Data{X= new DateTime(2013,11,04), Open= 74.4428, High= 75.555, Low=
73.1971,Close= 74.3657,Volume= 318132218},
new Data{X= new DateTime(2013,11,11), Open= 74.2843, High= 75.6114, Low=
73.4871,Close= 74.9987,Volume= 306711021},
new Data{X= new DateTime(2013,11,18), Open= 74.9985, High= 75.3128, Low=
73.3814,Close= 74.2571,Volume= 282778778},
};
}
```



Customization of MACD

The [Width](#) and [Color](#) of [MacdLine](#) can be customized by using [MacdLine](#) property. The positive and negative changes of histogram can be customized by [MacdPositiveColor](#) and [MacdNegativeColor](#) properties. The [MacdType](#) is used to define the type of MACD Indicator. The MACD period can be customized using [SlowPeriod](#) and [FastPeriod](#) properties.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart Title="AAPL - 2012-2017">
  <ChartPrimaryXAxis ValueType="Syncfusion.Blazor.Charts.ValueType.DateTime"
    Title="months" IntervalType="IntervalType.Months">
    <ChartAxisMajorGridLines Width="0"></ChartAxisMajorGridLines>
    <ChartAxisCrosshairTooltip Enable="true"></ChartAxisCrosshairTooltip>
  </ChartPrimaryXAxis>
  <ChartPrimaryYAxis Title="Price" LabelFormat="{value}" Minimum="50"
    Maximum="170" PlotOffset="25" Interval="30" RowIndex="1"
    OpposedPosition="true">
    <ChartAxisMajorTickLines Width="0"></ChartAxisMajorTickLines>
    <ChartAxisLineStyle Width="0"></ChartAxisLineStyle>
  </ChartPrimaryYAxis>
  <ChartAxes>
    <ChartAxis Name="secondary" OpposedPosition="true" RowIndex="0" Interval="3"
      Minimum="-3.5" Maximum="3.5" Title="MACD">
      <ChartAxisLineStyle Width="0"></ChartAxisLineStyle>
```



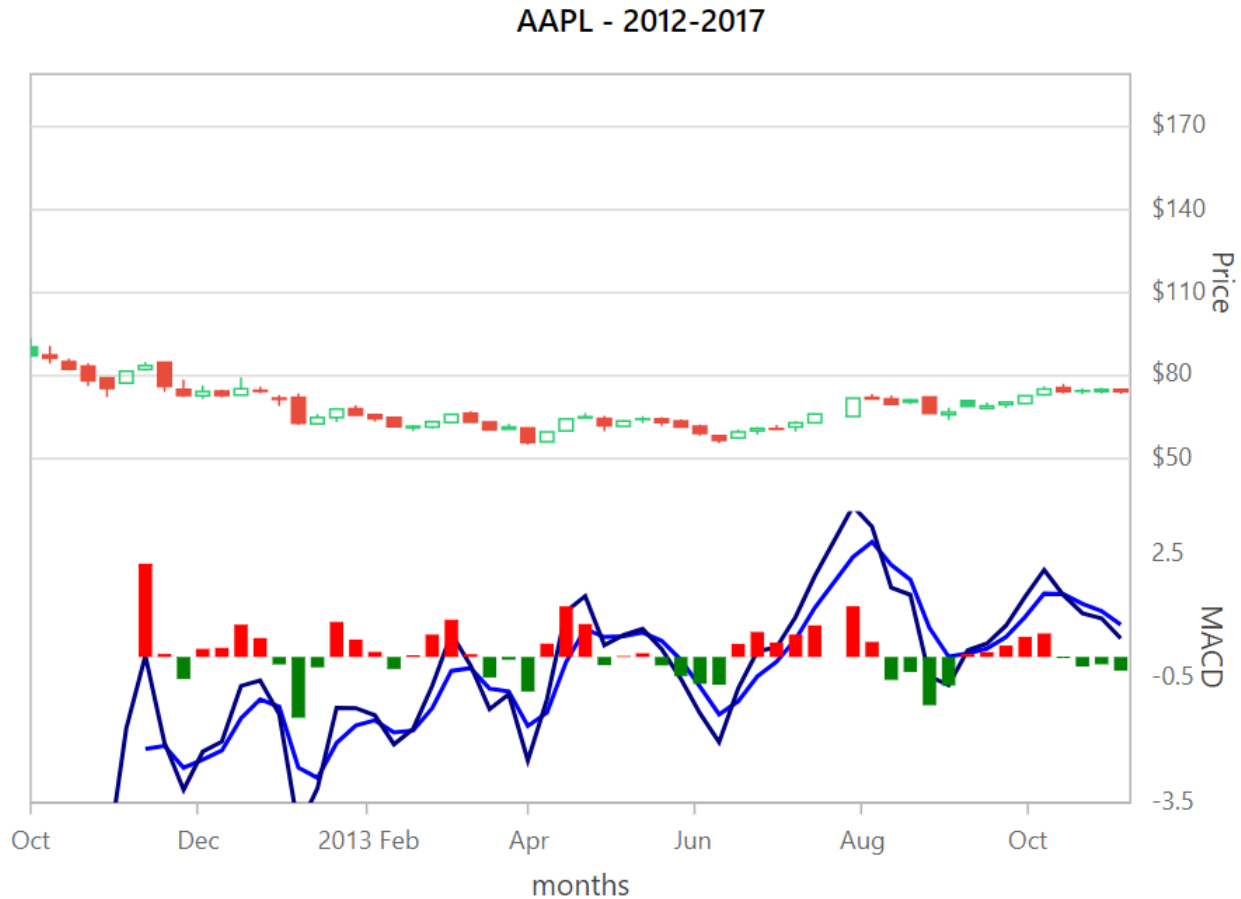
```

<ChartAxisMajorGridLines Width="0"></ChartAxisMajorGridLines>
<ChartAxisMajorTickLines Width="0"></ChartAxisMajorTickLines>
</ChartAxis>
</ChartAxes>
<ChartRows>
<ChartRow Height="40%"></ChartRow>
<ChartRow Height="60%"></ChartRow>
</ChartRows>
<ChartSeriesCollection>
<ChartSeries DataSource="@StockDetails" XName="X" YName="Y" Low="Low"
High="High" Close="Close" Volume="Volume" Open="Open"
Width="2" Name="Apple Inc" Type="ChartSeriesType.Candle">
</ChartSeries>
</ChartSeriesCollection>
<ChartIndicators>
<ChartIndicator MacdType="MacdType.Both" Type="TechnicalIndicators.Macd"
SeriesName="Apple Inc" Fill="blue" Period=3 YAxisName="secondary"
FastPeriod="5" SlowPeriod="2" Width="2" MacdPositiveColor="red"
MacdNegativeColor="green">
<ChartIndicatorMacdLine Color="navy"></ChartIndicatorMacdLine>
</ChartIndicator>
</ChartIndicators>
<ChartLegendSettings Visible="false" />
</SfChart>
@code{
public class Data
{
public DateTime X { get; set; }
public double High { get; set; }
public double Low { get; set; }
public double Open { get; set; }
public double Close { get; set; }
public double Volume { get; set; }
}
public List<Data> StockDetails = new List<Data>
{
new Data{X= new DateTime(2012,10,15), Open= 90.3357, High= 93.2557, Low=
87.0885,Close= 87.12,Volume= 646996264},
new Data{X= new DateTime(2012,10,22), Open= 87.4885, High= 90.7685, Low=
84.4285,Close= 86.2857,Volume= 866040680 },
new Data{X= new DateTime(2012,10,29), Open= 84.9828, High= 86.1428, Low=
82.1071,Close= 82.4,Volume= 367371310},
new Data{X= new DateTime(2012,11,05), Open= 83.3593, High= 84.3914, Low=
76.2457,Close= 78.1514,Volume= 919719846},
new Data{X= new DateTime(2012,11,12), Open= 79.1643, High= 79.2143, Low=
72.25,Close= 75.3825,Volume= 894382149},
new Data{X= new DateTime(2012,11,19), Open= 77.2443, High= 81.7143, Low=
77.1257,Close= 81.6428,Volume= 527416747},
new Data{X= new DateTime(2012,11,26), Open= 82.2714, High= 84.8928, Low=
81.7514,Close= 83.6114,Volume= 646467974},
new Data{X= new DateTime(2012,12,03), Open= 84.8071, High= 84.9414, Low=
74.09,Close= 76.1785,Volume= 980096264},
new Data{X= new DateTime(2012,12,10), Open= 75, High= 78.5085, Low=
72.2257,Close= 72.8277,Volume= 835016110},
new Data{X= new DateTime(2012,12,17), Open= 72.7043, High= 76.4143, Low=
71.6043,Close= 74.19,Volume= 726150329},

```

```
new Data{X= new DateTime(2012,12,24), Open= 74.3357, High= 74.8928, Low=
72.0943,Close= 72.7984,Volume= 321104733},
new Data{X= new DateTime(2012,12,31), Open= 72.9328, High= 79.2857, Low=
72.7143,Close= 75.2857,Volume= 540854882},
new Data{X= new DateTime(2013,01,07), Open= 74.5714, High= 75.9843, Low=
73.6,Close= 74.3285,Volume= 574594262},
new Data{X= new DateTime(2013,01,14), Open= 71.8114, High= 72.9643, Low=
69.0543,Close= 71.4285,Volume= 803105621},
new Data{X= new DateTime(2013,01,21), Open= 72.08, High= 73.57, Low=
62.1428,Close= 62.84,Volume= 971912560},
new Data{X= new DateTime(2013,01,28), Open= 62.5464, High= 66.0857, Low=
62.2657,Close= 64.8028,Volume= 656549587},
new Data{X= new DateTime(2013,02,04), Open= 64.8443, High= 68.4014, Low=
63.1428,Close= 67.8543,Volume= 743778993},
new Data{X= new DateTime(2013,02,11), Open= 68.0714, High= 69.2771, Low=
65.7028,Close= 65.7371,Volume= 585292366},
new Data{X= new DateTime(2013,02,18), Open= 65.8714, High= 66.1043, Low=
63.26,Close= 64.4014,Volume= 421766997},
new Data{X= new DateTime(2013,02,25), Open= 64.8357, High= 65.0171, Low=
61.4257,Close= 61.4957,Volume= 582741215},
new Data{X= new DateTime(2013,03,04), Open= 61.1143, High= 62.2043, Low=
59.8571,Close= 61.6743,Volume= 632856539},
new Data{X= new DateTime(2013,03,11), Open= 61.3928, High= 63.4614, Low=
60.7343,Close= 63.38,Volume= 572066981},
new Data{X= new DateTime(2013,03,18), Open= 63.0643, High= 66.0143, Low=
63.0286,Close= 65.9871,Volume= 552156035},
new Data{X= new DateTime(2013,03,25), Open= 66.3843, High= 67.1357, Low=
63.0886,Close= 63.2371,Volume= 390762517},
new Data{X= new DateTime(2013,04,01), Open= 63.1286, High= 63.3854, Low=
59.9543,Close= 60.4571,Volume= 505273732},
new Data{X= new DateTime(2013,04,08), Open= 60.6928, High= 62.57, Low=
60.3557,Close= 61.4,Volume= 387323550},
new Data{X= new DateTime(2013,04,15), Open= 61, High= 61.1271, Low=
55.0143,Close= 55.79,Volume= 709945604},
new Data{X= new DateTime(2013,04,22), Open= 56.0914, High= 59.8241, Low=
55.8964,Close= 59.6007,Volume= 787007506},
new Data{X= new DateTime(2013,04,29), Open= 60.0643, High= 64.7471, Low=
60,Close= 64.2828,Volume= 655020017},
new Data{X= new DateTime(2013,05,06), Open= 65.1014, High= 66.5357, Low=
64.3543,Close= 64.71,Volume= 545488533},
new Data{X= new DateTime(2013,05,13), Open= 64.5014, High= 65.4143, Low=
59.8428,Close= 61.8943,Volume= 633706550},
new Data{X= new DateTime(2013,05,20), Open= 61.7014, High= 64.05, Low=
61.4428,Close= 63.5928,Volume= 494379068},
new Data{X= new DateTime(2013,05,27), Open= 64.2714, High= 65.3, Low=
62.7714,Close= 64.2478,Volume= 362907830},
new Data{X= new DateTime(2013,06,03), Open= 64.39, High= 64.9186, Low=
61.8243,Close= 63.1158,Volume= 443249793},
new Data{X= new DateTime(2013,06,10), Open= 63.5328, High= 64.1541, Low=
61.2143,Close= 61.4357,Volume= 389680092},
new Data{X= new DateTime(2013,06,17), Open= 61.6343, High= 62.2428, Low=
58.3,Close= 59.0714,Volume= 400384818},
new Data{X= new DateTime(2013,06,24), Open= 58.2, High= 58.38, Low=
55.5528,Close= 56.6471,Volume= 519314826},
new Data{X= new DateTime(2013,07,01), Open= 57.5271, High= 60.47, Low=
57.3171,Close= 59.6314,Volume= 343878841},
```

```
new Data{X= new DateTime(2013,07,08), Open= 60.0157, High= 61.3986, Low=
58.6257,Close= 60.93,Volume= 384106977},
new Data{X= new DateTime(2013,07,15), Open= 60.7157, High= 62.1243, Low=
60.5957,Close= 60.7071,Volume= 286035513},
new Data{X= new DateTime(2013,07,22), Open= 61.3514, High= 63.5128, Low=
59.8157,Close= 62.9986,Volume= 395816827},
new Data{X= new DateTime(2013,07,29), Open= 62.9714, High= 66.1214, Low=
62.8857,Close= 66.0771,Volume= 339668858},
new Data{X= new DateTime(2013,08,12), Open= 65.2657, High= 72.0357, Low=
65.2328,Close= 71.7614,Volume= 711563584},
new Data{X= new DateTime(2013,08,19), Open= 72.0485, High= 73.3914, Low=
71.1714,Close= 71.5743,Volume= 417119660},
new Data{X= new DateTime(2013,08,26), Open= 71.5357, High= 72.8857, Low=
69.4286,Close= 69.6023,Volume= 392805888},
new Data{X= new DateTime(2013,09,02), Open= 70.4428, High= 71.7485, Low=
69.6214,Close= 71.1743,Volume= 317244380},
new Data{X= new DateTime(2013,09,09), Open= 72.1428, High= 72.56, Low=
66.3857,Close= 66.4143,Volume= 669376320},
new Data{X= new DateTime(2013,09,16), Open= 65.8571, High= 68.3643, Low=
63.8886,Close= 66.7728,Volume= 625142677},
new Data{X= new DateTime(2013,09,23), Open= 70.8714, High= 70.9871, Low=
68.6743,Close= 68.9643,Volume= 475274537},
new Data{X= new DateTime(2013,09,30), Open= 68.1786, High= 70.3357, Low=
67.773,Close= 69.0043,Volume= 368198906},
new Data{X= new DateTime(2013,10,07), Open= 69.5086, High= 70.5486, Low=
68.3257,Close= 70.4017,Volume= 361437661},
new Data{X= new DateTime(2013,10,14), Open= 69.9757, High= 72.7514, Low=
69.9071,Close= 72.6985,Volume= 342694379},
new Data{X= new DateTime(2013,10,21), Open= 73.11, High= 76.1757, Low=
72.5757,Close= 75.1368,Volume= 490458997},
new Data{X= new DateTime(2013,10,28), Open= 75.5771, High= 77.0357, Low=
73.5057,Close= 74.29,Volume= 508130174},
new Data{X= new DateTime(2013,11,04), Open= 74.4428, High= 75.555, Low=
73.1971,Close= 74.3657,Volume= 318132218},
new Data{X= new DateTime(2013,11,11), Open= 74.2843, High= 75.6114, Low=
73.4871,Close= 74.9987,Volume= 306711021},
new Data{X= new DateTime(2013,11,18), Open= 74.9985, High= 75.3128, Low=
73.3814,Close= 74.2571,Volume= 282778778},
};
}
```



Relative Strength Index (RSI)

Relative Strength Index (RSI) shows how strongly a stock is moving in its current direction. To render a RSI Indicator, set the indicator [Type](#) as [Rsi](#). An RSI Indicator will be represented by three lines - upper band, lower band and signal line. The upper band and lower band values are customized by [OverBought](#) and [OverSold](#) properties and the signal line is calculated by RSI formula.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart Title="AAPL - 2012-2017">
  <ChartPrimaryXAxis Title="Months"
    ValueType="Syncfusion.Blazor.Charts.ValueType.DateTime"
    IntervalType="IntervalType.Months">
    <ChartAxisMajorGridLines Width="0"></ChartAxisMajorGridLines>
  </ChartPrimaryXAxis>
  <ChartPrimaryYAxis Title="Price" LabelFormat="{value}M" Minimum="50"
    Maximum="170" Interval="30" RowIndex="1" OpposedPosition="true">
    <ChartAxisMajorTickLines Width="0"></ChartAxisMajorTickLines>
    <ChartAxisLineStyle Width="0"></ChartAxisLineStyle>
  </ChartPrimaryYAxis>
  <ChartAxes>
    <ChartAxis Name="secondary" OpposedPosition="true" RowIndex="0"
      Interval="60"
      Minimum="0" Maximum="120" Title="RSI">
      <ChartAxisLineStyle Width="0"></ChartAxisLineStyle>
    </ChartAxis>
  </ChartAxes>
</SfChart>
```

```

<ChartAxisMajorGridLines Width="0"></ChartAxisMajorGridLines>
<ChartAxisMajorTickLines Width="0"></ChartAxisMajorTickLines>
<ChartStriplines>
<ChartStripline ZIndex="ZIndex.Behind" Start="0" End="120" Text=""
Color="black" Visible="true" Opacity="0.03"></ChartStripline>
</ChartStriplines>
</ChartAxis>
</ChartAxes>
<ChartRows>
<ChartRow Height="40%"></ChartRow>
<ChartRow Height="60%"></ChartRow>
</ChartRows>
<ChartSeriesCollection>
<ChartSeries DataSource="@StockDetails" XName="X" YName="Y" Low="Low"
High="High" Close="Close" Volume="Volume" Open="Open"
Width="2" Name="Apple Inc" Type="ChartSeriesType.Candle">
</ChartSeries>
</ChartSeriesCollection>
<ChartIndicators>
<ChartIndicator Type="TechnicalIndicators.Rsi" SeriesName="Apple Inc"
Fill="blue" Period=3 Field="FinancialDataFields.Close"
YAxisName="secondary" ShowZones="true" OverBought="70" OverSold="30">
<ChartIndicatorUpperLine Color="red"></ChartIndicatorUpperLine>
<ChartIndicatorLowerLine Color="green"></ChartIndicatorLowerLine>
</ChartIndicator>
</ChartIndicators>
<ChartLegendSettings Visible="false" />
</SfChart>
@code{
public class Data
{
public DateTime X { get; set; }
public double Y { get; set; }
public double High { get; set; }
public double Low { get; set; }
public double Open { get; set; }
public double Close { get; set; }
public double Volume { get; set; }
}
public List<Data> StockDetails = new List<Data>
{
new Data{X= new DateTime(2012,10,15), Open= 90.3357, High= 93.2557, Low=
87.0885,Close= 87.12,Volume= 646996264},
new Data{X= new DateTime(2012,10,22), Open= 87.4885, High= 90.7685, Low=
84.4285,Close= 86.2857,Volume= 866040680 },
new Data{X= new DateTime(2012,10,29), Open= 84.9828, High= 86.1428, Low=
82.1071,Close= 82.4,Volume= 367371310},
new Data{X= new DateTime(2012,11,05), Open= 83.3593, High= 84.3914, Low=
76.2457,Close= 78.1514,Volume= 919719846},
new Data{X= new DateTime(2012,11,12), Open= 79.1643, High= 79.2143, Low=
72.25,Close= 75.3825,Volume= 894382149},
new Data{X= new DateTime(2012,11,19), Open= 77.2443, High= 81.7143, Low=
77.1257,Close= 81.6428,Volume= 527416747},
new Data{X= new DateTime(2012,11,26), Open= 82.2714, High= 84.8928, Low=
81.7514,Close= 83.6114,Volume= 646467974},
new Data{X= new DateTime(2012,12,03), Open= 84.8071, High= 84.9414, Low=
74.09,Close= 76.1785,Volume= 980096264},

```

```
new Data{X= new DateTime(2012,12,10), Open= 75, High= 78.5085, Low=
72.2257,Close= 72.8277,Volume= 835016110},
new Data{X= new DateTime(2012,12,17), Open= 72.7043, High= 76.4143, Low=
71.6043,Close= 74.19,Volume= 726150329},
new Data{X= new DateTime(2012,12,24), Open= 74.3357, High= 74.8928, Low=
72.0943,Close= 72.7984,Volume= 321104733},
new Data{X= new DateTime(2012,12,31), Open= 72.9328, High= 79.2857, Low=
72.7143,Close= 75.2857,Volume= 540854882},
new Data{X= new DateTime(2013,01,07), Open= 74.5714, High= 75.9843, Low=
73.6,Close= 74.3285,Volume= 574594262},
new Data{X= new DateTime(2013,01,14), Open= 71.8114, High= 72.9643, Low=
69.0543,Close= 71.4285,Volume= 803105621},
new Data{X= new DateTime(2013,01,21), Open= 72.08, High= 73.57, Low=
62.1428,Close= 62.84,Volume= 971912560},
new Data{X= new DateTime(2013,01,28), Open= 62.5464, High= 66.0857, Low=
62.2657,Close= 64.8028,Volume= 656549587},
new Data{X= new DateTime(2013,02,04), Open= 64.8443, High= 68.4014, Low=
63.1428,Close= 67.8543,Volume= 743778993},
new Data{X= new DateTime(2013,02,11), Open= 68.0714, High= 69.2771, Low=
65.7028,Close= 65.7371,Volume= 585292366},
new Data{X= new DateTime(2013,02,18), Open= 65.8714, High= 66.1043, Low=
63.26,Close= 64.4014,Volume= 421766997},
new Data{X= new DateTime(2013,02,25), Open= 64.8357, High= 65.0171, Low=
61.4257,Close= 61.4957,Volume= 582741215},
new Data{X= new DateTime(2013,03,04), Open= 61.1143, High= 62.2043, Low=
59.8571,Close= 61.6743,Volume= 632856539},
new Data{X= new DateTime(2013,03,11), Open= 61.3928, High= 63.4614, Low=
60.7343,Close= 63.38,Volume= 572066981},
new Data{X= new DateTime(2013,03,18), Open= 63.0643, High= 66.0143, Low=
63.0286,Close= 65.9871,Volume= 552156035},
new Data{X= new DateTime(2013,03,25), Open= 66.3843, High= 67.1357, Low=
63.0886,Close= 63.2371,Volume= 390762517},
new Data{X= new DateTime(2013,04,01), Open= 63.1286, High= 63.3854, Low=
59.9543,Close= 60.4571,Volume= 505273732},
new Data{X= new DateTime(2013,04,08), Open= 60.6928, High= 62.57, Low=
60.3557,Close= 61.4,Volume= 387323550},
new Data{X= new DateTime(2013,04,15), Open= 61, High= 61.1271, Low=
55.0143,Close= 55.79,Volume= 709945604},
new Data{X= new DateTime(2013,04,22), Open= 56.0914, High= 59.8241, Low=
55.8964,Close= 59.6007,Volume= 787007506},
new Data{X= new DateTime(2013,04,29), Open= 60.0643, High= 64.7471, Low=
60,Close= 64.2828,Volume= 655020017},
new Data{X= new DateTime(2013,05,06), Open= 65.1014, High= 66.5357, Low=
64.3543,Close= 64.71,Volume= 545488533},
new Data{X= new DateTime(2013,05,13), Open= 64.5014, High= 65.4143, Low=
59.8428,Close= 61.8943,Volume= 633706550},
new Data{X= new DateTime(2013,05,20), Open= 61.7014, High= 64.05, Low=
61.4428,Close= 63.5928,Volume= 494379068},
new Data{X= new DateTime(2013,05,27), Open= 64.2714, High= 65.3, Low=
62.7714,Close= 64.2478,Volume= 362907830},
new Data{X= new DateTime(2013,06,03), Open= 64.39, High= 64.9186, Low=
61.8243,Close= 63.1158,Volume= 443249793},
new Data{X= new DateTime(2013,06,10), Open= 63.5328, High= 64.1541, Low=
61.2143,Close= 61.4357,Volume= 389680092},
new Data{X= new DateTime(2013,06,17), Open= 61.6343, High= 62.2428, Low=
58.3,Close= 59.0714,Volume= 400384818},
```

```
new Data{X= new DateTime(2013,06,24), Open= 58.2, High= 58.38, Low=
55.5528,Close= 56.6471,Volume= 519314826},
new Data{X= new DateTime(2013,07,01), Open= 57.5271, High= 60.47, Low=
57.3171,Close= 59.6314,Volume= 343878841},
new Data{X= new DateTime(2013,07,08), Open= 60.0157, High= 61.3986, Low=
58.6257,Close= 60.93,Volume= 384106977},
new Data{X= new DateTime(2013,07,15), Open= 60.7157, High= 62.1243, Low=
60.5957,Close= 60.7071,Volume= 286035513},
new Data{X= new DateTime(2013,07,22), Open= 61.3514, High= 63.5128, Low=
59.8157,Close= 62.9986,Volume= 395816827},
new Data{X= new DateTime(2013,07,29), Open= 62.9714, High= 66.1214, Low=
62.8857,Close= 66.0771,Volume= 339668858},
new Data{X= new DateTime(2013,08,12), Open= 65.2657, High= 72.0357, Low=
65.2328,Close= 71.7614,Volume= 711563584},
new Data{X= new DateTime(2013,08,19), Open= 72.0485, High= 73.3914, Low=
71.1714,Close= 71.5743,Volume= 417119660},
new Data{X= new DateTime(2013,08,26), Open= 71.5357, High= 72.8857, Low=
69.4286,Close= 69.6023,Volume= 392805888},
new Data{X= new DateTime(2013,09,02), Open= 70.4428, High= 71.7485, Low=
69.6214,Close= 71.1743,Volume= 317244380},
new Data{X= new DateTime(2013,09,09), Open= 72.1428, High= 72.56, Low=
66.3857,Close= 66.4143,Volume= 669376320},
new Data{X= new DateTime(2013,09,16), Open= 65.8571, High= 68.3643, Low=
63.8886,Close= 66.7728,Volume= 625142677},
new Data{X= new DateTime(2013,09,23), Open= 70.8714, High= 70.9871, Low=
68.6743,Close= 68.9643,Volume= 475274537},
new Data{X= new DateTime(2013,09,30), Open= 68.1786, High= 70.3357, Low=
67.773,Close= 69.0043,Volume= 368198906},
new Data{X= new DateTime(2013,10,07), Open= 69.5086, High= 70.5486, Low=
68.3257,Close= 70.4017,Volume= 361437661},
new Data{X= new DateTime(2013,10,14), Open= 69.9757, High= 72.7514, Low=
69.9071,Close= 72.6985,Volume= 342694379},
new Data{X= new DateTime(2013,10,21), Open= 73.11, High= 76.1757, Low=
72.5757,Close= 75.1368,Volume= 490458997},
new Data{X= new DateTime(2013,10,28), Open= 75.5771, High= 77.0357, Low=
73.5057,Close= 74.29,Volume= 508130174},
new Data{X= new DateTime(2013,11,04), Open= 74.4428, High= 75.555, Low=
73.1971,Close= 74.3657,Volume= 318132218},
new Data{X= new DateTime(2013,11,11), Open= 74.2843, High= 75.6114, Low=
73.4871,Close= 74.9987,Volume= 306711021},
new Data{X= new DateTime(2013,11,18), Open= 74.9985, High= 75.3128, Low=
73.3814,Close= 74.2571,Volume= 282778778},
};
}
```



Simple Moving Average (SMA)

Moving Average Indicators are used to define the direction of the trend. To render a SMA Indicator, set the indicator [Type](#) as [Sma](#).

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart Title="AAPL - 2012-2017">
  <ChartPrimaryXAxis Title="Months"
    ValueType="Syncfusion.Blazor.Charts.ValueType.DateTime"
    IntervalType="IntervalType.Months">
    <ChartAxisMajorGridLines Width="0"></ChartAxisMajorGridLines>
  </ChartPrimaryXAxis>
  <ChartSeriesCollection>
    <ChartSeries DataSource="@StockDetails" XName="X" YName="Y" Low="Low"
      High="High" Close="Close" Volume="Volume" Open="Open"
      Width="2" Name="Apple Inc" Type="ChartSeriesType.Candle">
    </ChartSeries>
  </ChartSeriesCollection>
  <ChartIndicators>
    <ChartIndicator Type="TechnicalIndicators.Sma"
      Field="FinancialDataFields.Close" XName="X" SeriesName="Apple Inc"
      Fill="blue" Period="2">
    <ChartIndicatorUpperLine Color="red"></ChartIndicatorUpperLine>
    <ChartIndicatorLowerLine Color="green"></ChartIndicatorLowerLine>
  </ChartIndicator>
</SfChart>
```



```

</ChartIndicators>
<ChartLegendSettings Visible="false" />
</SfChart>
@code{
public class Data
{
public DateTime X { get; set; }
public double Y { get; set; }
public double High { get; set; }
public double Low { get; set; }
public double Open { get; set; }
public double Close { get; set; }
public double Volume { get; set; }
}
public List<Data> DataSource = new List<Data>
{
new Data{X= new DateTime(2012,10,15), Open= 90.3357, High= 93.2557, Low=
87.0885,Close= 87.12,Volume= 646996264},
new Data{X= new DateTime(2012,10,22), Open= 87.4885, High= 90.7685, Low=
84.4285,Close= 86.2857,Volume= 866040680 },
new Data{X= new DateTime(2012,10,29), Open= 84.9828, High= 86.1428, Low=
82.1071,Close= 82.4,Volume= 367371310},
new Data{X= new DateTime(2012,11,05), Open= 83.3593, High= 84.3914, Low=
76.2457,Close= 78.1514,Volume= 919719846},
new Data{X= new DateTime(2012,11,12), Open= 79.1643, High= 79.2143, Low=
72.25,Close= 75.3825,Volume= 894382149},
new Data{X= new DateTime(2012,11,19), Open= 77.2443, High= 81.7143, Low=
77.1257,Close= 81.6428,Volume= 527416747},
new Data{X= new DateTime(2012,11,26), Open= 82.2714, High= 84.8928, Low=
81.7514,Close= 83.6114,Volume= 646467974},
new Data{X= new DateTime(2012,12,03), Open= 84.8071, High= 84.9414, Low=
74.09,Close= 76.1785,Volume= 980096264},
new Data{X= new DateTime(2012,12,10), Open= 75, High= 78.5085, Low=
72.2257,Close= 72.8277,Volume= 835016110},
new Data{X= new DateTime(2012,12,17), Open= 72.7043, High= 76.4143, Low=
71.6043,Close= 74.19,Volume= 726150329},
new Data{X= new DateTime(2012,12,24), Open= 74.3357, High= 74.8928, Low=
72.0943,Close= 72.7984,Volume= 321104733},
new Data{X= new DateTime(2012,12,31), Open= 72.9328, High= 79.2857, Low=
72.7143,Close= 75.2857,Volume= 540854882},
new Data{X= new DateTime(2013,01,07), Open= 74.5714, High= 75.9843, Low=
73.6,Close= 74.3285,Volume= 574594262},
new Data{X= new DateTime(2013,01,14), Open= 71.8114, High= 72.9643, Low=
69.0543,Close= 71.4285,Volume= 803105621},
new Data{X= new DateTime(2013,01,21), Open= 72.08, High= 73.57, Low=
62.1428,Close= 62.84,Volume= 971912560},
new Data{X= new DateTime(2013,01,28), Open= 62.5464, High= 66.0857, Low=
62.2657,Close= 64.8028,Volume= 656549587},
new Data{X= new DateTime(2013,02,04), Open= 64.8443, High= 68.4014, Low=
63.1428,Close= 67.8543,Volume= 743778993},
new Data{X= new DateTime(2013,02,11), Open= 68.0714, High= 69.2771, Low=
65.7028,Close= 65.7371,Volume= 585292366},
new Data{X= new DateTime(2013,02,18), Open= 65.8714, High= 66.1043, Low=
63.26,Close= 64.4014,Volume= 421766997},
new Data{X= new DateTime(2013,02,25), Open= 64.8357, High= 65.0171, Low=
61.4257,Close= 61.4957,Volume= 582741215},

```

```
new Data{X= new DateTime(2013,03,04), Open= 61.1143, High= 62.2043, Low=
59.8571,Close= 61.6743,Volume= 632856539},
new Data{X= new DateTime(2013,03,11), Open= 61.3928, High= 63.4614, Low=
60.7343,Close= 63.38,Volume= 572066981},
new Data{X= new DateTime(2013,03,18), Open= 63.0643, High= 66.0143, Low=
63.0286,Close= 65.9871,Volume= 552156035},
new Data{X= new DateTime(2013,03,25), Open= 66.3843, High= 67.1357, Low=
63.0886,Close= 63.2371,Volume= 390762517},
new Data{X= new DateTime(2013,04,01), Open= 63.1286, High= 63.3854, Low=
59.9543,Close= 60.4571,Volume= 505273732},
new Data{X= new DateTime(2013,04,08), Open= 60.6928, High= 62.57, Low=
60.3557,Close= 61.4,Volume= 387323550},
new Data{X= new DateTime(2013,04,15), Open= 61, High= 61.1271, Low=
55.0143,Close= 55.79,Volume= 709945604},
new Data{X= new DateTime(2013,04,22), Open= 56.0914, High= 59.8241, Low=
55.8964,Close= 59.6007,Volume= 787007506},
new Data{X= new DateTime(2013,04,29), Open= 60.0643, High= 64.7471, Low=
60,Close= 64.2828,Volume= 655020017},
new Data{X= new DateTime(2013,05,06), Open= 65.1014, High= 66.5357, Low=
64.3543,Close= 64.71,Volume= 545488533},
new Data{X= new DateTime(2013,05,13), Open= 64.5014, High= 65.4143, Low=
59.8428,Close= 61.8943,Volume= 633706550},
new Data{X= new DateTime(2013,05,20), Open= 61.7014, High= 64.05, Low=
61.4428,Close= 63.5928,Volume= 494379068},
new Data{X= new DateTime(2013,05,27), Open= 64.2714, High= 65.3, Low=
62.7714,Close= 64.2478,Volume= 362907830},
new Data{X= new DateTime(2013,06,03), Open= 64.39, High= 64.9186, Low=
61.8243,Close= 63.1158,Volume= 443249793},
new Data{X= new DateTime(2013,06,10), Open= 63.5328, High= 64.1541, Low=
61.2143,Close= 61.4357,Volume= 389680092},
new Data{X= new DateTime(2013,06,17), Open= 61.6343, High= 62.2428, Low=
58.3,Close= 59.0714,Volume= 400384818},
new Data{X= new DateTime(2013,06,24), Open= 58.2, High= 58.38, Low=
55.5528,Close= 56.6471,Volume= 519314826},
new Data{X= new DateTime(2013,07,01), Open= 57.5271, High= 60.47, Low=
57.3171,Close= 59.6314,Volume= 343878841},
new Data{X= new DateTime(2013,07,08), Open= 60.0157, High= 61.3986, Low=
58.6257,Close= 60.93,Volume= 384106977},
new Data{X= new DateTime(2013,07,15), Open= 60.7157, High= 62.1243, Low=
60.5957,Close= 60.7071,Volume= 286035513},
new Data{X= new DateTime(2013,07,22), Open= 61.3514, High= 63.5128, Low=
59.8157,Close= 62.9986,Volume= 395816827},
new Data{X= new DateTime(2013,07,29), Open= 62.9714, High= 66.1214, Low=
62.8857,Close= 66.0771,Volume= 339668858},
new Data{X= new DateTime(2013,08,12), Open= 65.2657, High= 72.0357, Low=
65.2328,Close= 71.7614,Volume= 711563584},
new Data{X= new DateTime(2013,08,19), Open= 72.0485, High= 73.3914, Low=
71.1714,Close= 71.5743,Volume= 417119660},
new Data{X= new DateTime(2013,08,26), Open= 71.5357, High= 72.8857, Low=
69.4286,Close= 69.6023,Volume= 392805888},
new Data{X= new DateTime(2013,09,02), Open= 70.4428, High= 71.7485, Low=
69.6214,Close= 71.1743,Volume= 317244380},
new Data{X= new DateTime(2013,09,09), Open= 72.1428, High= 72.56, Low=
66.3857,Close= 66.4143,Volume= 669376320},
new Data{X= new DateTime(2013,09,16), Open= 65.8571, High= 68.3643, Low=
63.8886,Close= 66.7728,Volume= 625142677},
```

```

new Data{X= new DateTime(2013,09,23), Open= 70.8714, High= 70.9871, Low=
68.6743,Close= 68.9643,Volume= 475274537},
new Data{X= new DateTime(2013,09,30), Open= 68.1786, High= 70.3357, Low=
67.773,Close= 69.0043,Volume= 368198906},
new Data{X= new DateTime(2013,10,07), Open= 69.5086, High= 70.5486, Low=
68.3257,Close= 70.4017,Volume= 361437661},
new Data{X= new DateTime(2013,10,14), Open= 69.9757, High= 72.7514, Low=
69.9071,Close= 72.6985,Volume= 342694379},
new Data{X= new DateTime(2013,10,21), Open= 73.11, High= 76.1757, Low=
72.5757,Close= 75.1368,Volume= 490458997},
new Data{X= new DateTime(2013,10,28), Open= 75.5771, High= 77.0357, Low=
73.5057,Close= 74.29,Volume= 508130174},
new Data{X= new DateTime(2013,11,04), Open= 74.4428, High= 75.555, Low=
73.1971,Close= 74.3657,Volume= 318132218},
new Data{X= new DateTime(2013,11,11), Open= 74.2843, High= 75.6114, Low=
73.4871,Close= 74.9987,Volume= 306711021},
new Data{X= new DateTime(2013,11,18), Open= 74.9985, High= 75.3128, Low=
73.3814,Close= 74.2571,Volume= 282778778},
};
}

```

Stochastic

Stochastic Indicator is a momentum indicator that compares the particular closing price of a security to the range of its prices over a certain period of time. To render a Stochastic Indicator, set the indicator [Type](#) as [Stochastic](#). Stochastic Indicator will be represented by four lines - upper line, lower line, period line and signal line. In Stochastic Indicator, the upper band value and lower band value is customized by [OverBought](#) and [OverSold](#) properties and the period line and signal line is render based on stochastic formula.

Customization of Stochastic Indicator

The **Width** and **Color** of upper line, lower line and period line can be customized by using [UpperLine](#), [LowerLine](#), and [PeriodLine](#) properties of indicator. To find the average price [KPeriod](#) and [DPeriod](#) properties can be used.

ASPX-CS

```

@using Syncfusion.Blazor.Charts
<SfChart Title="AAPL - 2012-2017">
  <ChartPrimaryXAxis ValueType="Syncfusion.Blazor.Charts.ValueType.DateTime"
    Title="months" IntervalType="IntervalType.Months">
    <ChartAxisMajorGridLines Width="0"></ChartAxisMajorGridLines>
    <ChartAxisCrosshairTooltip Enable="true"></ChartAxisCrosshairTooltip>
  </ChartPrimaryXAxis>
  <ChartPrimaryYAxis Title="Price" LabelFormat="{value}M" Minimum="50"
    Maximum="170" Interval="30" RowIndex="1" OpposedPosition="true">
    <ChartAxisMajorTickLines Width="0"></ChartAxisMajorTickLines>
    <ChartAxisLineStyle Width="0"></ChartAxisLineStyle>
  </ChartPrimaryYAxis>
  <ChartAxes>
    <ChartAxis Name="secondary" OpposedPosition="true" RowIndex="0"
      Interval="60"
      Minimum="0" Maximum="120" Title="Stochastic">
      <ChartAxisLineStyle Width="0"></ChartAxisLineStyle>
      <ChartAxisMajorGridLines Width="0"></ChartAxisMajorGridLines>
    </ChartAxis>
  </ChartAxes>
</SfChart>

```

```

<ChartAxisMajorTickLines Width="0"></ChartAxisMajorTickLines>
<ChartStriplines>
<ChartStripline ZIndex="ZIndex.Behind" Start="0" End="120" Text=""
Color="black" Visible="true" Opacity="0.03"></ChartStripline>
</ChartStriplines>
</ChartAxis>
</ChartAxes>
<ChartRows>
<ChartRow Height="40%"></ChartRow>
<ChartRow Height="60%"></ChartRow>
</ChartRows>
<ChartSeriesCollection>
<ChartSeries DataSource="@StockDetails" XName="X" YName="Y" Low="Low"
High="High" Close="Close" Volume="Volume" Open="Open"
Width="2" Name="Apple Inc" Type="ChartSeriesType.Candle">
</ChartSeries>
</ChartSeriesCollection>
<ChartIndicators>
<ChartIndicator Type="TechnicalIndicators.Stochastic"
Field="FinancialDataFields.Close" SeriesName="Apple Inc" Fill="blue"
Period=3
YAxisName="secondary" KPeriod="2" DPeriod="3" ShowZones="true">
<ChartIndicatorPeriodLine Color="yellow"></ChartIndicatorPeriodLine>
<ChartIndicatorUpperLine Color="red"></ChartIndicatorUpperLine>
<ChartIndicatorLowerLine Color="green"></ChartIndicatorLowerLine>
</ChartIndicator>
</ChartIndicators>
<ChartLegendSettings Visible="false" />
</SfChart>
@code{
public class Data
{
public DateTime X { get; set; }
public double High { get; set; }
public double Low { get; set; }
public double Open { get; set; }
public double Close { get; set; }
public double Volume { get; set; }
}
public List<Data> StockDetails = new List<Data>
{
new Data{X= new DateTime(2012,10,15), Open= 90.3357, High= 93.2557, Low=
87.0885,Close= 87.12,Volume= 646996264},
new Data{X= new DateTime(2012,10,22), Open= 87.4885, High= 90.7685, Low=
84.4285,Close= 86.2857,Volume= 866040680 },
new Data{X= new DateTime(2012,10,29), Open= 84.9828, High= 86.1428, Low=
82.1071,Close= 82.4,Volume= 367371310},
new Data{X= new DateTime(2012,11,05), Open= 83.3593, High= 84.3914, Low=
76.2457,Close= 78.1514,Volume= 919719846},
new Data{X= new DateTime(2012,11,12), Open= 79.1643, High= 79.2143, Low=
72.25,Close= 75.3825,Volume= 894382149},
new Data{X= new DateTime(2012,11,19), Open= 77.2443, High= 81.7143, Low=
77.1257,Close= 81.6428,Volume= 527416747},
new Data{X= new DateTime(2012,11,26), Open= 82.2714, High= 84.8928, Low=
81.7514,Close= 83.6114,Volume= 646467974},
new Data{X= new DateTime(2012,12,03), Open= 84.8071, High= 84.9414, Low=
74.09,Close= 76.1785,Volume= 980096264},

```

```
new Data{X= new DateTime(2012,12,10), Open= 75, High= 78.5085, Low=
72.2257,Close= 72.8277,Volume= 835016110},
new Data{X= new DateTime(2012,12,17), Open= 72.7043, High= 76.4143, Low=
71.6043,Close= 74.19,Volume= 726150329},
new Data{X= new DateTime(2012,12,24), Open= 74.3357, High= 74.8928, Low=
72.0943,Close= 72.7984,Volume= 321104733},
new Data{X= new DateTime(2012,12,31), Open= 72.9328, High= 79.2857, Low=
72.7143,Close= 75.2857,Volume= 540854882},
new Data{X= new DateTime(2013,01,07), Open= 74.5714, High= 75.9843, Low=
73.6,Close= 74.3285,Volume= 574594262},
new Data{X= new DateTime(2013,01,14), Open= 71.8114, High= 72.9643, Low=
69.0543,Close= 71.4285,Volume= 803105621},
new Data{X= new DateTime(2013,01,21), Open= 72.08, High= 73.57, Low=
62.1428,Close= 62.84,Volume= 971912560},
new Data{X= new DateTime(2013,01,28), Open= 62.5464, High= 66.0857, Low=
62.2657,Close= 64.8028,Volume= 656549587},
new Data{X= new DateTime(2013,02,04), Open= 64.8443, High= 68.4014, Low=
63.1428,Close= 67.8543,Volume= 743778993},
new Data{X= new DateTime(2013,02,11), Open= 68.0714, High= 69.2771, Low=
65.7028,Close= 65.7371,Volume= 585292366},
new Data{X= new DateTime(2013,02,18), Open= 65.8714, High= 66.1043, Low=
63.26,Close= 64.4014,Volume= 421766997},
new Data{X= new DateTime(2013,02,25), Open= 64.8357, High= 65.0171, Low=
61.4257,Close= 61.4957,Volume= 582741215},
new Data{X= new DateTime(2013,03,04), Open= 61.1143, High= 62.2043, Low=
59.8571,Close= 61.6743,Volume= 632856539},
new Data{X= new DateTime(2013,03,11), Open= 61.3928, High= 63.4614, Low=
60.7343,Close= 63.38,Volume= 572066981},
new Data{X= new DateTime(2013,03,18), Open= 63.0643, High= 66.0143, Low=
63.0286,Close= 65.9871,Volume= 552156035},
new Data{X= new DateTime(2013,03,25), Open= 66.3843, High= 67.1357, Low=
63.0886,Close= 63.2371,Volume= 390762517},
new Data{X= new DateTime(2013,04,01), Open= 63.1286, High= 63.3854, Low=
59.9543,Close= 60.4571,Volume= 505273732},
new Data{X= new DateTime(2013,04,08), Open= 60.6928, High= 62.57, Low=
60.3557,Close= 61.4,Volume= 387323550},
new Data{X= new DateTime(2013,04,15), Open= 61, High= 61.1271, Low=
55.0143,Close= 55.79,Volume= 709945604},
new Data{X= new DateTime(2013,04,22), Open= 56.0914, High= 59.8241, Low=
55.8964,Close= 59.6007,Volume= 787007506},
new Data{X= new DateTime(2013,04,29), Open= 60.0643, High= 64.7471, Low=
60,Close= 64.2828,Volume= 655020017},
new Data{X= new DateTime(2013,05,06), Open= 65.1014, High= 66.5357, Low=
64.3543,Close= 64.71,Volume= 545488533},
new Data{X= new DateTime(2013,05,13), Open= 64.5014, High= 65.4143, Low=
59.8428,Close= 61.8943,Volume= 633706550},
new Data{X= new DateTime(2013,05,20), Open= 61.7014, High= 64.05, Low=
61.4428,Close= 63.5928,Volume= 494379068},
new Data{X= new DateTime(2013,05,27), Open= 64.2714, High= 65.3, Low=
62.7714,Close= 64.2478,Volume= 362907830},
new Data{X= new DateTime(2013,06,03), Open= 64.39, High= 64.9186, Low=
61.8243,Close= 63.1158,Volume= 443249793},
new Data{X= new DateTime(2013,06,10), Open= 63.5328, High= 64.1541, Low=
61.2143,Close= 61.4357,Volume= 389680092},
new Data{X= new DateTime(2013,06,17), Open= 61.6343, High= 62.2428, Low=
58.3,Close= 59.0714,Volume= 400384818},
```

```
new Data{X= new DateTime(2013,06,24), Open= 58.2, High= 58.38, Low=
55.5528,Close= 56.6471,Volume= 519314826},
new Data{X= new DateTime(2013,07,01), Open= 57.5271, High= 60.47, Low=
57.3171,Close= 59.6314,Volume= 343878841},
new Data{X= new DateTime(2013,07,08), Open= 60.0157, High= 61.3986, Low=
58.6257,Close= 60.93,Volume= 384106977},
new Data{X= new DateTime(2013,07,15), Open= 60.7157, High= 62.1243, Low=
60.5957,Close= 60.7071,Volume= 286035513},
new Data{X= new DateTime(2013,07,22), Open= 61.3514, High= 63.5128, Low=
59.8157,Close= 62.9986,Volume= 395816827},
new Data{X= new DateTime(2013,07,29), Open= 62.9714, High= 66.1214, Low=
62.8857,Close= 66.0771,Volume= 339668858},
new Data{X= new DateTime(2013,08,12), Open= 65.2657, High= 72.0357, Low=
65.2328,Close= 71.7614,Volume= 711563584},
new Data{X= new DateTime(2013,08,19), Open= 72.0485, High= 73.3914, Low=
71.1714,Close= 71.5743,Volume= 417119660},
new Data{X= new DateTime(2013,08,26), Open= 71.5357, High= 72.8857, Low=
69.4286,Close= 69.6023,Volume= 392805888},
new Data{X= new DateTime(2013,09,02), Open= 70.4428, High= 71.7485, Low=
69.6214,Close= 71.1743,Volume= 317244380},
new Data{X= new DateTime(2013,09,09), Open= 72.1428, High= 72.56, Low=
66.3857,Close= 66.4143,Volume= 669376320},
new Data{X= new DateTime(2013,09,16), Open= 65.8571, High= 68.3643, Low=
63.8886,Close= 66.7728,Volume= 625142677},
new Data{X= new DateTime(2013,09,23), Open= 70.8714, High= 70.9871, Low=
68.6743,Close= 68.9643,Volume= 475274537},
new Data{X= new DateTime(2013,09,30), Open= 68.1786, High= 70.3357, Low=
67.773,Close= 69.0043,Volume= 368198906},
new Data{X= new DateTime(2013,10,07), Open= 69.5086, High= 70.5486, Low=
68.3257,Close= 70.4017,Volume= 361437661},
new Data{X= new DateTime(2013,10,14), Open= 69.9757, High= 72.7514, Low=
69.9071,Close= 72.6985,Volume= 342694379},
new Data{X= new DateTime(2013,10,21), Open= 73.11, High= 76.1757, Low=
72.5757,Close= 75.1368,Volume= 490458997},
new Data{X= new DateTime(2013,10,28), Open= 75.5771, High= 77.0357, Low=
73.5057,Close= 74.29,Volume= 508130174},
new Data{X= new DateTime(2013,11,04), Open= 74.4428, High= 75.555, Low=
73.1971,Close= 74.3657,Volume= 318132218},
new Data{X= new DateTime(2013,11,11), Open= 74.2843, High= 75.6114, Low=
73.4871,Close= 74.9987,Volume= 306711021},
new Data{X= new DateTime(2013,11,18), Open= 74.9985, High= 75.3128, Low=
73.3814,Close= 74.2571,Volume= 282778778},
};
}
```



Triangular Moving Average (TMA)

Triangular Moving Average (TMA) Indicator is used to define the direction of the trend. To render a TMA Indicator, set the indicator [Type](#) as [TMA](#).

Customization of Triangular Moving Average

[Fill](#), [Width](#) and [DashArray](#) properties are used to customize the **Color**, **Width**, and **Dashes** of the signal line of the indicators. The [Period](#) property is used to predict the data forecast calculations. The [Field](#) value is used to compare the current price with previous price. It is applicable to Bollinger Bands and Moving Averages. The [ShowZones](#) property is used to show or hide the over bought and over sold regions. It is applicable for RSI and Stochastic indicators.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart Title="AAPL - 2012-2017">
  <ChartPrimaryXAxis ValueType="Syncfusion.Blazor.Charts.ValueType.DateTime"
    Title="months" IntervalType="IntervalType.Months">
    <ChartAxisMajorGridLines Width="0"></ChartAxisMajorGridLines>
    <ChartAxisCrosshairTooltip Enable="true"></ChartAxisCrosshairTooltip>
  </ChartPrimaryXAxis>
  <ChartSeriesCollection>
    <ChartSeries DataSource="@StockDetails" XName="X" YName="Y" Low="Low"
      High="High" Close="Close" Volume="Volume" Open="Open"
      Width="2" Name="Apple Inc" Type="ChartSeriesType.Candle">
    </ChartSeries>
  </ChartSeriesCollection>
</SfChart>
```

```

</ChartSeriesCollection>
<ChartIndicators>
<ChartIndicator Type="TechnicalIndicators.Tma"
Field="FinancialDataFields.Close" SeriesName="Apple Inc" Fill="red"
Period=3>
</ChartIndicator>
</ChartIndicators>
</SfChart>
@code{
public class Data
{
public DateTime X { get; set; }
public double High { get; set; }
public double Low { get; set; }
public double Open { get; set; }
public double Close { get; set; }
public double Volume { get; set; }
}
public List<Data> StockDetails = new List<Data>
{
new Data{X= new DateTime(2012,10,15), Open= 90.3357, High= 93.2557, Low=
87.0885,Close= 87.12,Volume= 646996264},
new Data{X= new DateTime(2012,10,22), Open= 87.4885, High= 90.7685, Low=
84.4285,Close= 86.2857,Volume= 866040680 },
new Data{X= new DateTime(2012,10,29), Open= 84.9828, High= 86.1428, Low=
82.1071,Close= 82.4,Volume= 367371310},
new Data{X= new DateTime(2012,11,05), Open= 83.3593, High= 84.3914, Low=
76.2457,Close= 78.1514,Volume= 919719846},
new Data{X= new DateTime(2012,11,12), Open= 79.1643, High= 79.2143, Low=
72.25,Close= 75.3825,Volume= 894382149},
new Data{X= new DateTime(2012,11,19), Open= 77.2443, High= 81.7143, Low=
77.1257,Close= 81.6428,Volume= 527416747},
new Data{X= new DateTime(2012,11,26), Open= 82.2714, High= 84.8928, Low=
81.7514,Close= 83.6114,Volume= 646467974},
new Data{X= new DateTime(2012,12,03), Open= 84.8071, High= 84.9414, Low=
74.09,Close= 76.1785,Volume= 980096264},
new Data{X= new DateTime(2012,12,10), Open= 75, High= 78.5085, Low=
72.2257,Close= 72.8277,Volume= 835016110},
new Data{X= new DateTime(2012,12,17), Open= 72.7043, High= 76.4143, Low=
71.6043,Close= 74.19,Volume= 726150329},
new Data{X= new DateTime(2012,12,24), Open= 74.3357, High= 74.8928, Low=
72.0943,Close= 72.7984,Volume= 321104733},
new Data{X= new DateTime(2012,12,31), Open= 72.9328, High= 79.2857, Low=
72.7143,Close= 75.2857,Volume= 540854882},
new Data{X= new DateTime(2013,01,07), Open= 74.5714, High= 75.9843, Low=
73.6,Close= 74.3285,Volume= 574594262},
new Data{X= new DateTime(2013,01,14), Open= 71.8114, High= 72.9643, Low=
69.0543,Close= 71.4285,Volume= 803105621},
new Data{X= new DateTime(2013,01,21), Open= 72.08, High= 73.57, Low=
62.1428,Close= 62.84,Volume= 971912560},
new Data{X= new DateTime(2013,01,28), Open= 62.5464, High= 66.0857, Low=
62.2657,Close= 64.8028,Volume= 656549587},
new Data{X= new DateTime(2013,02,04), Open= 64.8443, High= 68.4014, Low=
63.1428,Close= 67.8543,Volume= 743778993},
new Data{X= new DateTime(2013,02,11), Open= 68.0714, High= 69.2771, Low=
65.7028,Close= 65.7371,Volume= 585292366},

```



```
new Data{X= new DateTime(2013,02,18), Open= 65.8714, High= 66.1043, Low=
63.26,Close= 64.4014,Volume= 421766997},
new Data{X= new DateTime(2013,02,25), Open= 64.8357, High= 65.0171, Low=
61.4257,Close= 61.4957,Volume= 582741215},
new Data{X= new DateTime(2013,03,04), Open= 61.1143, High= 62.2043, Low=
59.8571,Close= 61.6743,Volume= 632856539},
new Data{X= new DateTime(2013,03,11), Open= 61.3928, High= 63.4614, Low=
60.7343,Close= 63.38,Volume= 572066981},
new Data{X= new DateTime(2013,03,18), Open= 63.0643, High= 66.0143, Low=
63.0286,Close= 65.9871,Volume= 552156035},
new Data{X= new DateTime(2013,03,25), Open= 66.3843, High= 67.1357, Low=
63.0886,Close= 63.2371,Volume= 390762517},
new Data{X= new DateTime(2013,04,01), Open= 63.1286, High= 63.3854, Low=
59.9543,Close= 60.4571,Volume= 505273732},
new Data{X= new DateTime(2013,04,08), Open= 60.6928, High= 62.57, Low=
60.3557,Close= 61.4,Volume= 387323550},
new Data{X= new DateTime(2013,04,15), Open= 61, High= 61.1271, Low=
55.0143,Close= 55.79,Volume= 709945604},
new Data{X= new DateTime(2013,04,22), Open= 56.0914, High= 59.8241, Low=
55.8964,Close= 59.6007,Volume= 787007506},
new Data{X= new DateTime(2013,04,29), Open= 60.0643, High= 64.7471, Low=
60,Close= 64.2828,Volume= 655020017},
new Data{X= new DateTime(2013,05,06), Open= 65.1014, High= 66.5357, Low=
64.3543,Close= 64.71,Volume= 545488533},
new Data{X= new DateTime(2013,05,13), Open= 64.5014, High= 65.4143, Low=
59.8428,Close= 61.8943,Volume= 633706550},
new Data{X= new DateTime(2013,05,20), Open= 61.7014, High= 64.05, Low=
61.4428,Close= 63.5928,Volume= 494379068},
new Data{X= new DateTime(2013,05,27), Open= 64.2714, High= 65.3, Low=
62.7714,Close= 64.2478,Volume= 362907830},
new Data{X= new DateTime(2013,06,03), Open= 64.39, High= 64.9186, Low=
61.8243,Close= 63.1158,Volume= 443249793},
new Data{X= new DateTime(2013,06,10), Open= 63.5328, High= 64.1541, Low=
61.2143,Close= 61.4357,Volume= 389680092},
new Data{X= new DateTime(2013,06,17), Open= 61.6343, High= 62.2428, Low=
58.3,Close= 59.0714,Volume= 400384818},
new Data{X= new DateTime(2013,06,24), Open= 58.2, High= 58.38, Low=
55.5528,Close= 56.6471,Volume= 519314826},
new Data{X= new DateTime(2013,07,01), Open= 57.5271, High= 60.47, Low=
57.3171,Close= 59.6314,Volume= 343878841},
new Data{X= new DateTime(2013,07,08), Open= 60.0157, High= 61.3986, Low=
58.6257,Close= 60.93,Volume= 384106977},
new Data{X= new DateTime(2013,07,15), Open= 60.7157, High= 62.1243, Low=
60.5957,Close= 60.7071,Volume= 286035513},
new Data{X= new DateTime(2013,07,22), Open= 61.3514, High= 63.5128, Low=
59.8157,Close= 62.9986,Volume= 395816827},
new Data{X= new DateTime(2013,07,29), Open= 62.9714, High= 66.1214, Low=
62.8857,Close= 66.0771,Volume= 339668858},
new Data{X= new DateTime(2013,08,12), Open= 65.2657, High= 72.0357, Low=
65.2328,Close= 71.7614,Volume= 711563584},
new Data{X= new DateTime(2013,08,19), Open= 72.0485, High= 73.3914, Low=
71.1714,Close= 71.5743,Volume= 417119660},
new Data{X= new DateTime(2013,08,26), Open= 71.5357, High= 72.8857, Low=
69.4286,Close= 69.6023,Volume= 392805888},
new Data{X= new DateTime(2013,09,02), Open= 70.4428, High= 71.7485, Low=
69.6214,Close= 71.1743,Volume= 317244380},
```

```
new Data{X= new DateTime(2013,09,09), Open= 72.1428, High= 72.56, Low=
66.3857,Close= 66.4143,Volume= 669376320},
new Data{X= new DateTime(2013,09,16), Open= 65.8571, High= 68.3643, Low=
63.8886,Close= 66.7728,Volume= 625142677},
new Data{X= new DateTime(2013,09,23), Open= 70.8714, High= 70.9871, Low=
68.6743,Close= 68.9643,Volume= 475274537},
new Data{X= new DateTime(2013,09,30), Open= 68.1786, High= 70.3357, Low=
67.773,Close= 69.0043,Volume= 368198906},
new Data{X= new DateTime(2013,10,07), Open= 69.5086, High= 70.5486, Low=
68.3257,Close= 70.4017,Volume= 361437661},
new Data{X= new DateTime(2013,10,14), Open= 69.9757, High= 72.7514, Low=
69.9071,Close= 72.6985,Volume= 342694379},
new Data{X= new DateTime(2013,10,21), Open= 73.11, High= 76.1757, Low=
72.5757,Close= 75.1368,Volume= 490458997},
new Data{X= new DateTime(2013,10,28), Open= 75.5771, High= 77.0357, Low=
73.5057,Close= 74.29,Volume= 508130174},
new Data{X= new DateTime(2013,11,04), Open= 74.4428, High= 75.555, Low=
73.1971,Close= 74.3657,Volume= 318132218},
new Data{X= new DateTime(2013,11,11), Open= 74.2843, High= 75.6114, Low=
73.4871,Close= 74.9987,Volume= 306711021},
new Data{X= new DateTime(2013,11,18), Open= 74.9985, High= 75.3128, Low=
73.3814,Close= 74.2571,Volume= 282778778},
};
}
```

AAPL - 2012-2017

**Data Source**

Usually technical indicators are added along with a financial series. The [SeriesName](#) identifies the series whose data to be analyzed using indicators.

Technical indicators can also be added without series using [DataSource](#) property of indicator.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart Title="AAPL - 2012-2017">
  <ChartAxes>
    <ChartAxis Name="secondary" OpposedPosition="true" RowIndex="0" Minimum="-7000000000" Maximum="5000000000"
    Title="Accumulation Distribution" Interval="6000000000">
    </ChartAxis>
  </ChartAxes>
  <ChartPrimaryXAxis ValueType="Syncfusion.Blazor.Charts.ValueType.DateTime"
  Title="months" IntervalType="IntervalType.Months">
  <ChartAxisMajorGridLines Width="0"></ChartAxisMajorGridLines>
  <ChartAxisCrosshairTooltip Enable="true"></ChartAxisCrosshairTooltip>
  </ChartPrimaryXAxis>
  <ChartIndicators>
    <ChartIndicator DataSource="@StockDetails"
    Type="TechnicalIndicators.AccumulationDistribution"
    Field="FinancialDataFields.Close"
    XName="X" Low="Low" High="High" Close="Close" Volume="Volume" Open="Open"
    SeriesName="Apple Inc"
    YAxisName="secondary" Fill="Blue" Period=3>
    <ChartIndicatorAnimation Enable="true"></ChartIndicatorAnimation>
  </ChartIndicator>
  </ChartIndicators>
</SfChart>

@code{
public class Data
{
  public DateTime X { get; set; }
  public double High { get; set; }
  public double Low { get; set; }
  public double Open { get; set; }
  public double Close { get; set; }
  public double Volume { get; set; }
}

public List<Data> StockDetails = new List<Data>
{
  new Data{X= new DateTime(2012,10,15), Open= 90.3357, High= 93.2557, Low= 87.0885,Close= 87.12,Volume= 646996264},
  new Data{X= new DateTime(2012,10,22), Open= 87.4885, High= 90.7685, Low= 84.4285,Close= 86.2857,Volume= 866040680 },
  new Data{X= new DateTime(2012,10,29), Open= 84.9828, High= 86.1428, Low= 82.1071,Close= 82.4,Volume= 367371310},
  new Data{X= new DateTime(2012,11,05), Open= 83.3593, High= 84.3914, Low= 76.2457,Close= 78.1514,Volume= 919719846},
  new Data{X= new DateTime(2012,11,12), Open= 79.1643, High= 79.2143, Low= 72.25,Close= 75.3825,Volume= 894382149},
  new Data{X= new DateTime(2012,11,19), Open= 77.2443, High= 81.7143, Low= 77.1257,Close= 81.6428,Volume= 527416747},
  new Data{X= new DateTime(2012,11,26), Open= 82.2714, High= 84.8928, Low= 81.7514,Close= 83.6114,Volume= 646467974},
```

```
new Data{X= new DateTime(2012,12,03), Open= 84.8071, High= 84.9414, Low=
74.09,Close= 76.1785,Volume= 980096264},
new Data{X= new DateTime(2012,12,10), Open= 75, High= 78.5085, Low=
72.2257,Close= 72.8277,Volume= 835016110},
new Data{X= new DateTime(2012,12,17), Open= 72.7043, High= 76.4143, Low=
71.6043,Close= 74.19,Volume= 726150329},
new Data{X= new DateTime(2012,12,24), Open= 74.3357, High= 74.8928, Low=
72.0943,Close= 72.7984,Volume= 321104733},
new Data{X= new DateTime(2012,12,31), Open= 72.9328, High= 79.2857, Low=
72.7143,Close= 75.2857,Volume= 540854882},
new Data{X= new DateTime(2013,01,07), Open= 74.5714, High= 75.9843, Low=
73.6,Close= 74.3285,Volume= 574594262},
new Data{X= new DateTime(2013,01,14), Open= 71.8114, High= 72.9643, Low=
69.0543,Close= 71.4285,Volume= 803105621},
new Data{X= new DateTime(2013,01,21), Open= 72.08, High= 73.57, Low=
62.1428,Close= 62.84,Volume= 971912560},
new Data{X= new DateTime(2013,01,28), Open= 62.5464, High= 66.0857, Low=
62.2657,Close= 64.8028,Volume= 656549587},
new Data{X= new DateTime(2013,02,04), Open= 64.8443, High= 68.4014, Low=
63.1428,Close= 67.8543,Volume= 743778993},
new Data{X= new DateTime(2013,02,11), Open= 68.0714, High= 69.2771, Low=
65.7028,Close= 65.7371,Volume= 585292366},
new Data{X= new DateTime(2013,02,18), Open= 65.8714, High= 66.1043, Low=
63.26,Close= 64.4014,Volume= 421766997},
new Data{X= new DateTime(2013,02,25), Open= 64.8357, High= 65.0171, Low=
61.4257,Close= 61.4957,Volume= 582741215},
new Data{X= new DateTime(2013,03,04), Open= 61.1143, High= 62.2043, Low=
59.8571,Close= 61.6743,Volume= 632856539},
new Data{X= new DateTime(2013,03,11), Open= 61.3928, High= 63.4614, Low=
60.7343,Close= 63.38,Volume= 572066981},
new Data{X= new DateTime(2013,03,18), Open= 63.0643, High= 66.0143, Low=
63.0286,Close= 65.9871,Volume= 552156035},
new Data{X= new DateTime(2013,03,25), Open= 66.3843, High= 67.1357, Low=
63.0886,Close= 63.2371,Volume= 390762517},
new Data{X= new DateTime(2013,04,01), Open= 63.1286, High= 63.3854, Low=
59.9543,Close= 60.4571,Volume= 505273732},
new Data{X= new DateTime(2013,04,08), Open= 60.6928, High= 62.57, Low=
60.3557,Close= 61.4,Volume= 387323550},
new Data{X= new DateTime(2013,04,15), Open= 61, High= 61.1271, Low=
55.0143,Close= 55.79,Volume= 709945604},
new Data{X= new DateTime(2013,04,22), Open= 56.0914, High= 59.8241, Low=
55.8964,Close= 59.6007,Volume= 787007506},
new Data{X= new DateTime(2013,04,29), Open= 60.0643, High= 64.7471, Low=
60,Close= 64.2828,Volume= 655020017},
new Data{X= new DateTime(2013,05,06), Open= 65.1014, High= 66.5357, Low=
64.3543,Close= 64.71,Volume= 545488533},
new Data{X= new DateTime(2013,05,13), Open= 64.5014, High= 65.4143, Low=
59.8428,Close= 61.8943,Volume= 633706550},
new Data{X= new DateTime(2013,05,20), Open= 61.7014, High= 64.05, Low=
61.4428,Close= 63.5928,Volume= 494379068},
new Data{X= new DateTime(2013,05,27), Open= 64.2714, High= 65.3, Low=
62.7714,Close= 64.2478,Volume= 362907830},
new Data{X= new DateTime(2013,06,03), Open= 64.39, High= 64.9186, Low=
61.8243,Close= 63.1158,Volume= 443249793},
new Data{X= new DateTime(2013,06,10), Open= 63.5328, High= 64.1541, Low=
61.2143,Close= 61.4357,Volume= 389680092},
```

```
new Data{X= new DateTime(2013,06,17), Open= 61.6343, High= 62.2428, Low=
58.3,Close= 59.0714,Volume= 400384818},
new Data{X= new DateTime(2013,06,24), Open= 58.2, High= 58.38, Low=
55.5528,Close= 56.6471,Volume= 519314826},
new Data{X= new DateTime(2013,07,01), Open= 57.5271, High= 60.47, Low=
57.3171,Close= 59.6314,Volume= 343878841},
new Data{X= new DateTime(2013,07,08), Open= 60.0157, High= 61.3986, Low=
58.6257,Close= 60.93,Volume= 384106977},
new Data{X= new DateTime(2013,07,15), Open= 60.7157, High= 62.1243, Low=
60.5957,Close= 60.7071,Volume= 286035513},
new Data{X= new DateTime(2013,07,22), Open= 61.3514, High= 63.5128, Low=
59.8157,Close= 62.9986,Volume= 395816827},
new Data{X= new DateTime(2013,07,29), Open= 62.9714, High= 66.1214, Low=
62.8857,Close= 66.0771,Volume= 339668858},
new Data{X= new DateTime(2013,08,12), Open= 65.2657, High= 72.0357, Low=
65.2328,Close= 71.7614,Volume= 711563584},
new Data{X= new DateTime(2013,08,19), Open= 72.0485, High= 73.3914, Low=
71.1714,Close= 71.5743,Volume= 417119660},
new Data{X= new DateTime(2013,08,26), Open= 71.5357, High= 72.8857, Low=
69.4286,Close= 69.6023,Volume= 392805888},
new Data{X= new DateTime(2013,09,02), Open= 70.4428, High= 71.7485, Low=
69.6214,Close= 71.1743,Volume= 317244380},
new Data{X= new DateTime(2013,09,09), Open= 72.1428, High= 72.56, Low=
66.3857,Close= 66.4143,Volume= 669376320},
new Data{X= new DateTime(2013,09,16), Open= 65.8571, High= 68.3643, Low=
63.8886,Close= 66.7728,Volume= 625142677},
new Data{X= new DateTime(2013,09,23), Open= 70.8714, High= 70.9871, Low=
68.6743,Close= 68.9643,Volume= 475274537},
new Data{X= new DateTime(2013,09,30), Open= 68.1786, High= 70.3357, Low=
67.773,Close= 69.0043,Volume= 368198906},
new Data{X= new DateTime(2013,10,07), Open= 69.5086, High= 70.5486, Low=
68.3257,Close= 70.4017,Volume= 361437661},
new Data{X= new DateTime(2013,10,14), Open= 69.9757, High= 72.7514, Low=
69.9071,Close= 72.6985,Volume= 342694379},
new Data{X= new DateTime(2013,10,21), Open= 73.11, High= 76.1757, Low=
72.5757,Close= 75.1368,Volume= 490458997},
new Data{X= new DateTime(2013,10,28), Open= 75.5771, High= 77.0357, Low=
73.5057,Close= 74.29,Volume= 508130174},
new Data{X= new DateTime(2013,11,04), Open= 74.4428, High= 75.555, Low=
73.1971,Close= 74.3657,Volume= 318132218},
new Data{X= new DateTime(2013,11,11), Open= 74.2843, High= 75.6114, Low=
73.4871,Close= 74.9987,Volume= 306711021},
new Data{X= new DateTime(2013,11,18), Open= 74.9985, High= 75.3128, Low=
73.3814,Close= 74.2571,Volume= 282778778},
};
}
```

Refer to our [Blazor Charts](#) feature tour page for its groundbreaking feature representations and also explore our [Blazor Chart Example](#) to know various chart types and how to represent time-dependent data, showing trends at equal intervals.

See Also

- [Data label](#)
- [Tooltip](#)

- [Legend](#)

<!-- markdownlint-disable MD036 -->

Trendlines in Blazor Charts Component

Trendlines are used to show the price's direction and pace. Except for bar series, trendlines can be generated for Cartesian series like Line, Column, Scatter, Area, Candle, Hilo, and so on. In addition, a series can have multiple trendlines. There are six different types of trendlines that can be used on the chart. They are as follows:

Linear

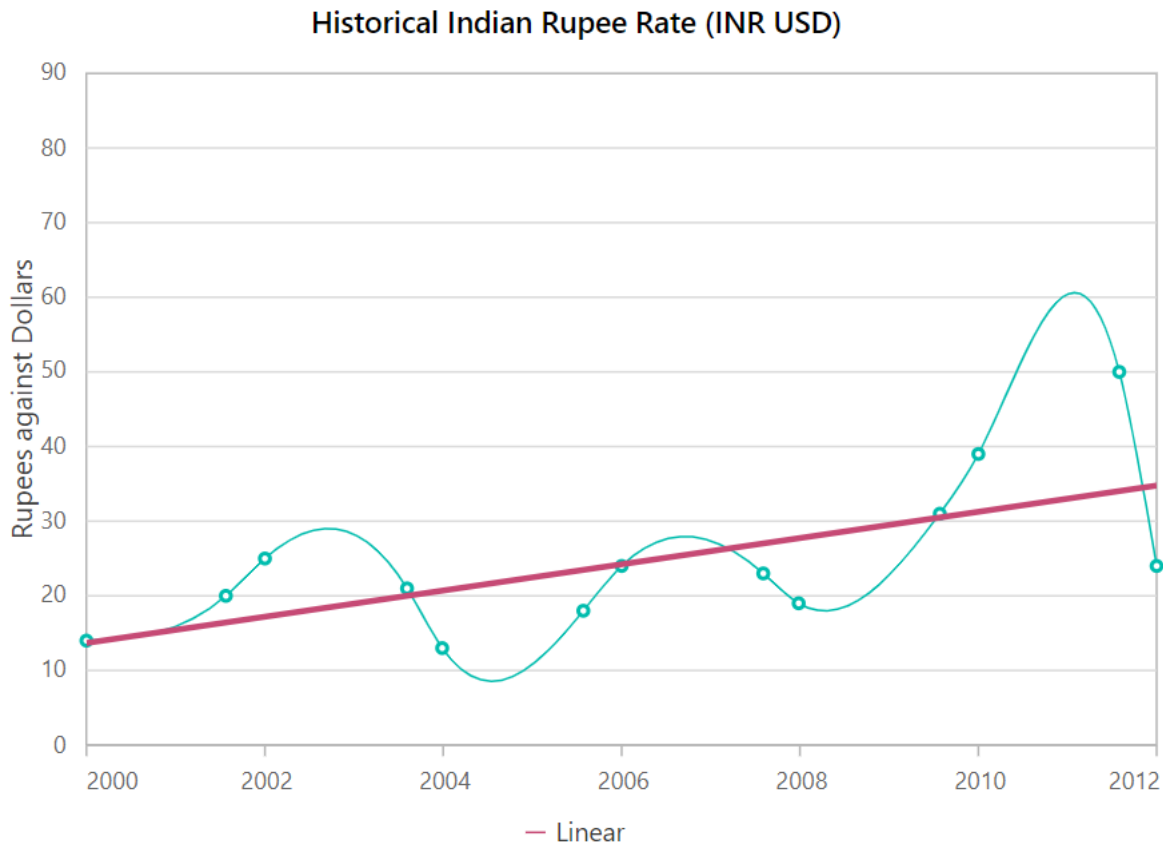
A linear trendline is a best-fit straight line used with simpler data sets. To render a linear trendline, set the [Type](#) property to [Linear](#).

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart Title="Historical Indian Rupee Rate (INR USD)">
  <ChartPrimaryXAxis LabelFormat="yyyy"
    ValueType="Syncfusion.Blazor.Charts.ValueType.DateTime"
    EdgeLabelPlacement="EdgeLabelPlacement.Shift">
  </ChartPrimaryXAxis>
  <ChartAxisMajorGridLines Width="0"></ChartAxisMajorGridLines>
  <ChartPrimaryYAxis Title="Rupees against Dollars">
  <ChartAxisLineStyle Width="0"></ChartAxisLineStyle>
  <ChartAxisMajorTickLines Width="0"></ChartAxisMajorTickLines>
  </ChartPrimaryYAxis>
  <ChartSeriesCollection>
  <ChartSeries DataSource="@Data" XName="XValue" YName="YValue"
    Type="ChartSeriesType.Spline">
  <ChartMarker Visible="true">
  </ChartMarker>
  <ChartTrendlines>
  <ChartTrendline Type="TrendlineTypes.Linear" Width="3" Name="Linear"
    Fill="#C64A75">
  </ChartTrendline>
  </ChartTrendlines>
  </ChartSeries>
  </ChartSeriesCollection>
</SfChart>

@code{
public class ChartData
{
public DateTime XValue { get; set; }
public double YValue { get; set; }
}
public List<ChartData> Data = new List<ChartData>
{
new ChartData { XValue = new DateTime(2000, 2, 11), YValue = 14 },
new ChartData { XValue = new DateTime(2001, 9, 4), YValue = 20 },
new ChartData { XValue = new DateTime(2002, 2, 11), YValue = 25 },
new ChartData { XValue = new DateTime(2003, 9, 16), YValue = 21 },
new ChartData { XValue = new DateTime(2004, 2, 7), YValue = 13},
new ChartData { XValue = new DateTime(2005, 9, 7), YValue = 18 },
new ChartData { XValue = new DateTime(2006, 2, 11), YValue = 24 },
new ChartData { XValue = new DateTime(2007, 9, 14), YValue = 23 },
```

```
new ChartData { XValue = new DateTime(2008, 2, 6), YValue = 19 },
new ChartData { XValue = new DateTime(2009, 9, 6), YValue = 31 },
new ChartData { XValue = new DateTime(2010, 2, 11), YValue = 39},
new ChartData { XValue = new DateTime(2011, 9, 11), YValue = 50 },
new ChartData { XValue = new DateTime(2012, 2, 11), YValue = 24 },
};
}
```



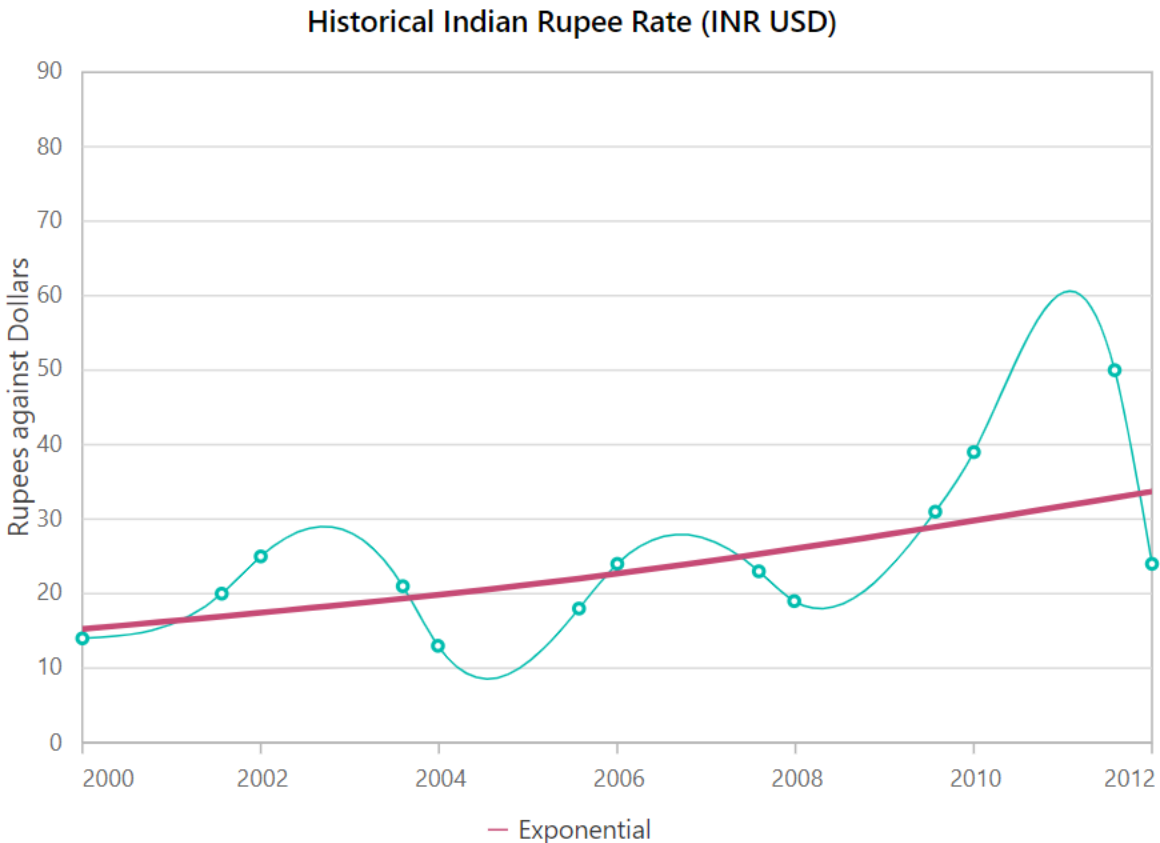
Exponential

Exponential trendline is a curved line that is most useful when data values rise or fall at increasingly higher rates. If the data contains zero or negative values, an exponential trendline cannot be created. To render an exponential trendline, set the [Type](#) property to [Exponential](#).

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart Title="Historical Indian Rupee Rate (INR USD)">
<ChartPrimaryXAxis LabelFormat="yyyy"
Value="Syncfusion.Blazor.Charts.ValueType.DateTime"
EdgeLabelPlacement="EdgeLabelPlacement.Shift">
<ChartAxisMajorGridLines Width="0"></ChartAxisMajorGridLines>
</ChartPrimaryXAxis>
<ChartPrimaryYAxis Title="Rupees against Dollars">
<ChartAxisLineStyle Width="0"></ChartAxisLineStyle>
<ChartAxisMajorTickLines Width="0"></ChartAxisMajorTickLines>
</ChartPrimaryYAxis>
```

```
<ChartSeriesCollection>
<ChartSeries DataSource="@Data" XName="XValue" YName="YValue"
Type="ChartSeriesType.Spline">
<ChartMarker Visible="true">
</ChartMarker>
<ChartTrendlines>
<ChartTrendline Type="TrendlineTypes.Exponential" Width="3"
Name="Exponential" Fill="#C64A75">
</ChartTrendline>
</ChartTrendlines>
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code{
public class ChartData
{
public DateTime XValue { get; set; }
public double YValue { get; set; }
}
public List<ChartData> Data = new List<ChartData>
{
new ChartData { XValue = new DateTime(2000, 2, 11), YValue = 14 },
new ChartData { XValue = new DateTime(2001, 9, 4), YValue = 20 },
new ChartData { XValue = new DateTime(2002, 2, 11), YValue = 25 },
new ChartData { XValue = new DateTime(2003, 9, 16), YValue = 21 },
new ChartData { XValue = new DateTime(2004, 2, 7), YValue = 13},
new ChartData { XValue = new DateTime(2005, 9, 7), YValue = 18 },
new ChartData { XValue = new DateTime(2006, 2, 11), YValue = 24 },
new ChartData { XValue = new DateTime(2007, 9, 14), YValue = 23 },
new ChartData { XValue = new DateTime(2008, 2, 6), YValue = 19 },
new ChartData { XValue = new DateTime(2009, 9, 6), YValue = 31 },
new ChartData { XValue = new DateTime(2010, 2, 11), YValue = 39},
new ChartData { XValue = new DateTime(2011, 9, 11), YValue = 50 },
new ChartData { XValue = new DateTime(2012, 2, 11), YValue = 24 },
};
}
```

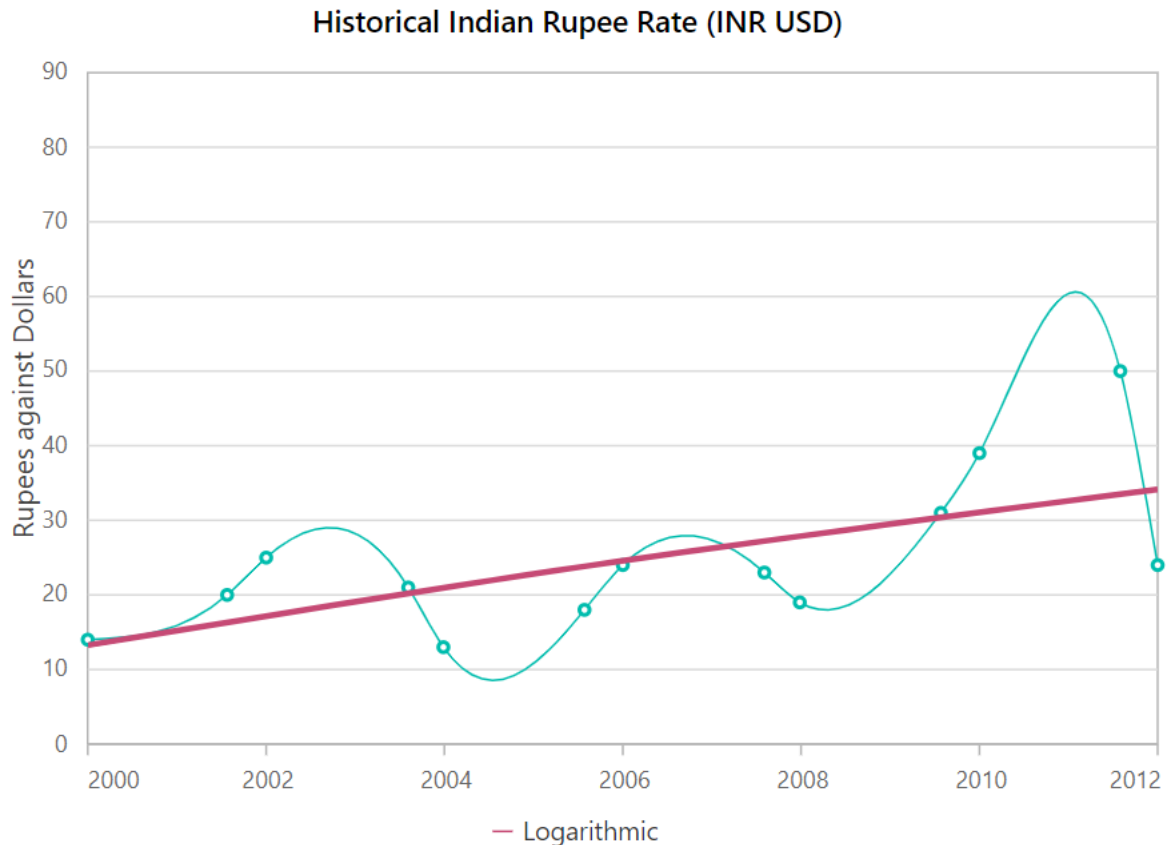
Logarithmic

A logarithmic trendline is a best-fit curved line that is most useful when the rate of change in the data increases or decreases quickly and then levels out. Negative and positive numbers can be used in a logarithmic trendline. To render a logarithmic trendline, set the [Type](#) property to [Logarithmic](#).

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart Title="Historical Indian Rupee Rate (INR USD)">
  <ChartPrimaryXAxis LabelFormat="yyyy"
    ValueType="Syncfusion.Blazor.Charts.ValueType.DateTime"
    EdgeLabelPlacement="EdgeLabelPlacement.Shift">
    <ChartAxisMajorGridLines Width="0"></ChartAxisMajorGridLines>
  </ChartPrimaryXAxis>
  <ChartPrimaryYAxis Title="Rupees against Dollars">
    <ChartAxisLineStyle Width="0"></ChartAxisLineStyle>
    <ChartAxisMajorTickLines Width="0"></ChartAxisMajorTickLines>
  </ChartPrimaryYAxis>
  <ChartSeriesCollection>
    <ChartSeries DataSource="@Data" XName="XValue" YName="YValue"
      Type="ChartSeriesType.Spline">
      <ChartMarker Visible="true">
      </ChartMarker>
    </ChartSeries>
    <ChartTrendlines>
      <ChartTrendline Type="TrendlineTypes.Logarithmic" Width="3"
        Name="Logarithmic" Fill="#C64A75">
      </ChartTrendline>
    </ChartTrendlines>
  </ChartSeriesCollection>
</SfChart>
```

```
</ChartTrendlines>
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code{
public class ChartData
{
public DateTime XValue { get; set; }
public double YValue { get; set; }
}
public List<ChartData> Data = new List<ChartData>
{
new ChartData { XValue = new DateTime(2000, 2, 11), YValue = 14 },
new ChartData { XValue = new DateTime(2001, 9, 4), YValue = 20 },
new ChartData { XValue = new DateTime(2002, 2, 11), YValue = 25 },
new ChartData { XValue = new DateTime(2003, 9, 16), YValue = 21 },
new ChartData { XValue = new DateTime(2004, 2, 7), YValue = 13},
new ChartData { XValue = new DateTime(2005, 9, 7), YValue = 18 },
new ChartData { XValue = new DateTime(2006, 2, 11), YValue = 24 },
new ChartData { XValue = new DateTime(2007, 9, 14), YValue = 23 },
new ChartData { XValue = new DateTime(2008, 2, 6), YValue = 19 },
new ChartData { XValue = new DateTime(2009, 9, 6), YValue = 31 },
new ChartData { XValue = new DateTime(2010, 2, 11), YValue = 39},
new ChartData { XValue = new DateTime(2011, 9, 11), YValue = 50 },
new ChartData { XValue = new DateTime(2012, 2, 11), YValue = 24 },
};
}
```



Polynomial

A polynomial trendline is a curved line that is used when data fluctuates. To render a polynomial trendline, set the [Type](#) property to [Polynomial](#).

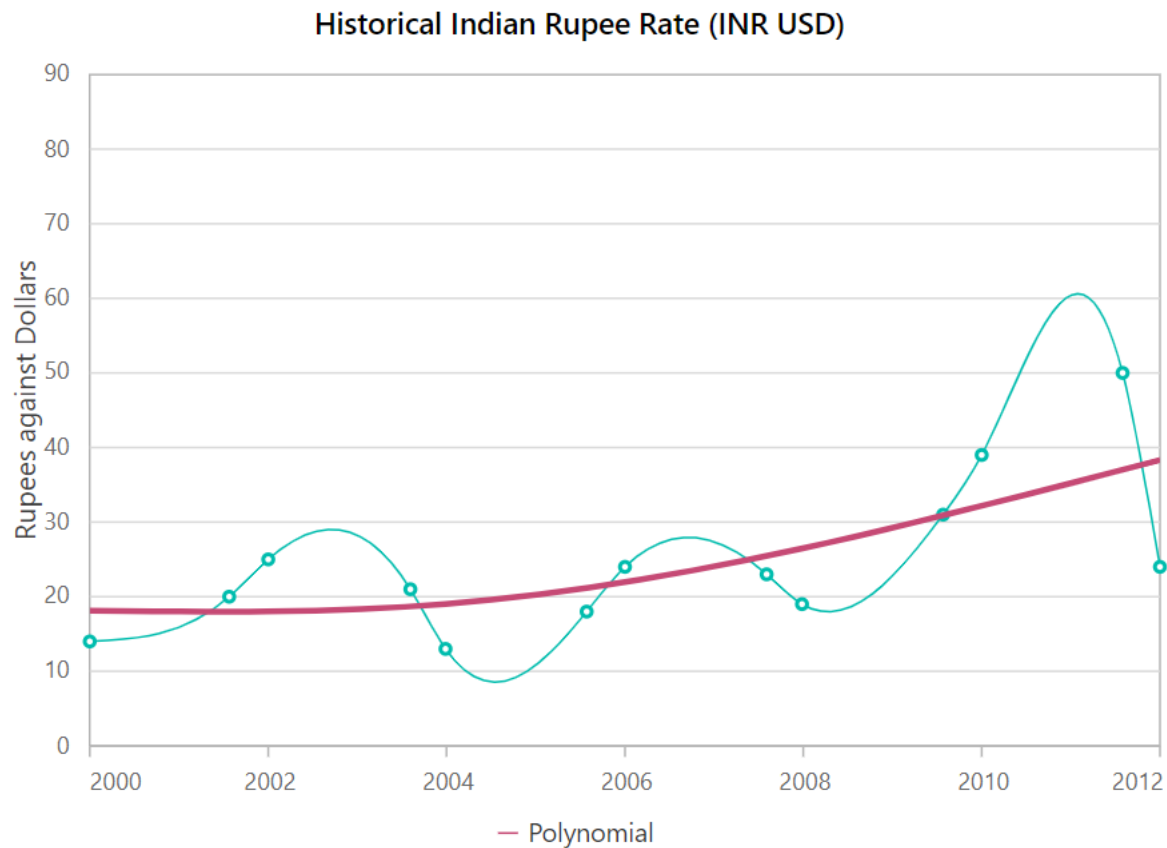
ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart Title="Historical Indian Rupee Rate (INR USD)">
  <ChartPrimaryXAxis LabelFormat="yyyy"
    ValueType="Syncfusion.Blazor.Charts.ValueType.DateTime"
    EdgeLabelPlacement="EdgeLabelPlacement.Shift">
    <ChartAxisMajorGridLines Width="0"></ChartAxisMajorGridLines>
  </ChartPrimaryXAxis>
  <ChartPrimaryYAxis Title="Rupees against Dollars">
    <ChartAxisLineStyle Width="0"></ChartAxisLineStyle>
    <ChartAxisMajorTickLines Width="0"></ChartAxisMajorTickLines>
  </ChartPrimaryYAxis>
  <ChartSeriesCollection>
    <ChartSeries DataSource="@Data" XName="XValue" YName="YValue"
      Type="ChartSeriesType.Spline">
      <ChartMarker Visible="true">
      </ChartMarker>
    </ChartSeries>
    <ChartTrendlines>
      <ChartTrendline Type="TrendlineTypes.Polynomial" Width="3" Name="Polynomial"
        Fill="#C64A75">
      </ChartTrendline>
    </ChartTrendlines>
  </ChartSeriesCollection>
</SfChart>
```

```

</ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code{
public class ChartData
{
public DateTime XValue { get; set; }
public double YValue { get; set; }
}
public List<ChartData> Data = new List<ChartData>
{
new ChartData { XValue = new DateTime(2000, 2, 11), YValue = 14 },
new ChartData { XValue = new DateTime(2001, 9, 4), YValue = 20 },
new ChartData { XValue = new DateTime(2002, 2, 11), YValue = 25 },
new ChartData { XValue = new DateTime(2003, 9, 16), YValue = 21 },
new ChartData { XValue = new DateTime(2004, 2, 7), YValue = 13},
new ChartData { XValue = new DateTime(2005, 9, 7), YValue = 18 },
new ChartData { XValue = new DateTime(2006, 2, 11), YValue = 24 },
new ChartData { XValue = new DateTime(2007, 9, 14), YValue = 23 },
new ChartData { XValue = new DateTime(2008, 2, 6), YValue = 19 },
new ChartData { XValue = new DateTime(2009, 9, 6), YValue = 31 },
new ChartData { XValue = new DateTime(2010, 2, 11), YValue = 39},
new ChartData { XValue = new DateTime(2011, 9, 11), YValue = 50 },
new ChartData { XValue = new DateTime(2012, 2, 11), YValue = 24 },
};
}

```



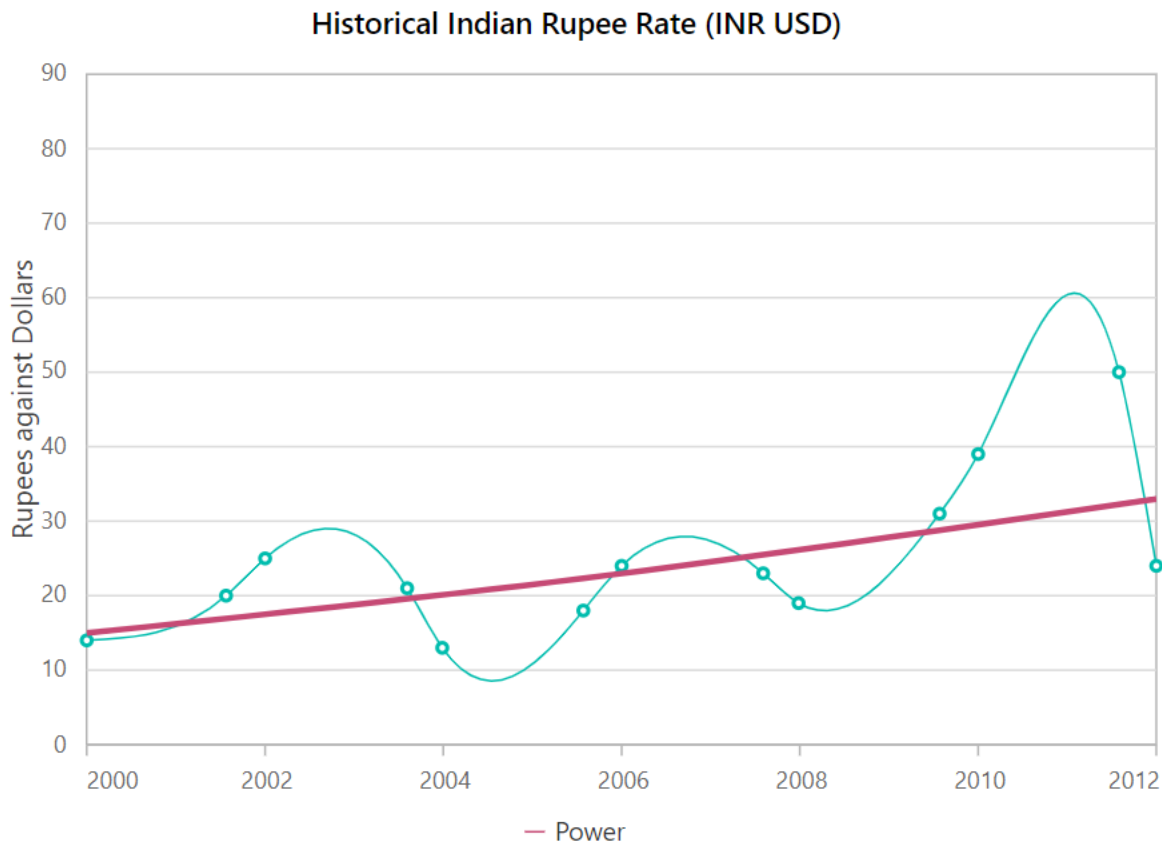
Power

A power trendline is a curved line that is best used with data sets that compare measurements that increase at a specific rate. To render a power trendline, set the [Type](#) property to [Power](#).

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart Title="Historical Indian Rupee Rate (INR USD)">
  <ChartPrimaryXAxis LabelFormat="yyyy"
  ValueType="Syncfusion.Blazor.Charts.ValueType.DateTime"
  EdgeLabelPlacement="EdgeLabelPlacement.Shift">
    <ChartAxisMajorGridLines Width="0"></ChartAxisMajorGridLines>
  </ChartPrimaryXAxis>
  <ChartPrimaryYAxis Title="Rupees against Dollars">
    <ChartAxisLineStyle Width="0"></ChartAxisLineStyle>
    <ChartAxisMajorTickLines Width="0"></ChartAxisMajorTickLines>
  </ChartPrimaryYAxis>
  <ChartSeriesCollection>
    <ChartSeries DataSource="@Data" XName="XValue" YName="YValue"
    Type="ChartSeriesType.Spline">
      <ChartMarker Visible="true">
      </ChartMarker>
      <ChartTrendlines>
        <ChartTrendline Type="TrendlineTypes.Power" Width="3" Name="Power"
        Fill="#C64A75">
        </ChartTrendline>
      </ChartTrendlines>
    </ChartSeries>
  </ChartSeriesCollection>
</SfChart>

@code{
public class ChartData
{
public DateTime XValue { get; set; }
public double YValue { get; set; }
}
public List<ChartData> Data = new List<ChartData>
{
new ChartData { XValue = new DateTime(2000, 2, 11), YValue = 14 },
new ChartData { XValue = new DateTime(2001, 9, 4), YValue = 20 },
new ChartData { XValue = new DateTime(2002, 2, 11), YValue = 25 },
new ChartData { XValue = new DateTime(2003, 9, 16), YValue = 21 },
new ChartData { XValue = new DateTime(2004, 2, 7), YValue = 13},
new ChartData { XValue = new DateTime(2005, 9, 7), YValue = 18 },
new ChartData { XValue = new DateTime(2006, 2, 11), YValue = 24 },
new ChartData { XValue = new DateTime(2007, 9, 14), YValue = 23 },
new ChartData { XValue = new DateTime(2008, 2, 6), YValue = 19 },
new ChartData { XValue = new DateTime(2009, 9, 6), YValue = 31 },
new ChartData { XValue = new DateTime(2010, 2, 11), YValue = 39},
new ChartData { XValue = new DateTime(2011, 9, 11), YValue = 50 },
new ChartData { XValue = new DateTime(2012, 2, 11), YValue = 24 },
};
}
```



Moving Average

A moving average trendline smoothen out fluctuations in data to show a pattern or trend more clearly. To render a moving average trendline, set the [Type](#) property to [MovingAverage](#). The [Period](#) property specifies how long the moving average should be calculated over.

ASPX-CS

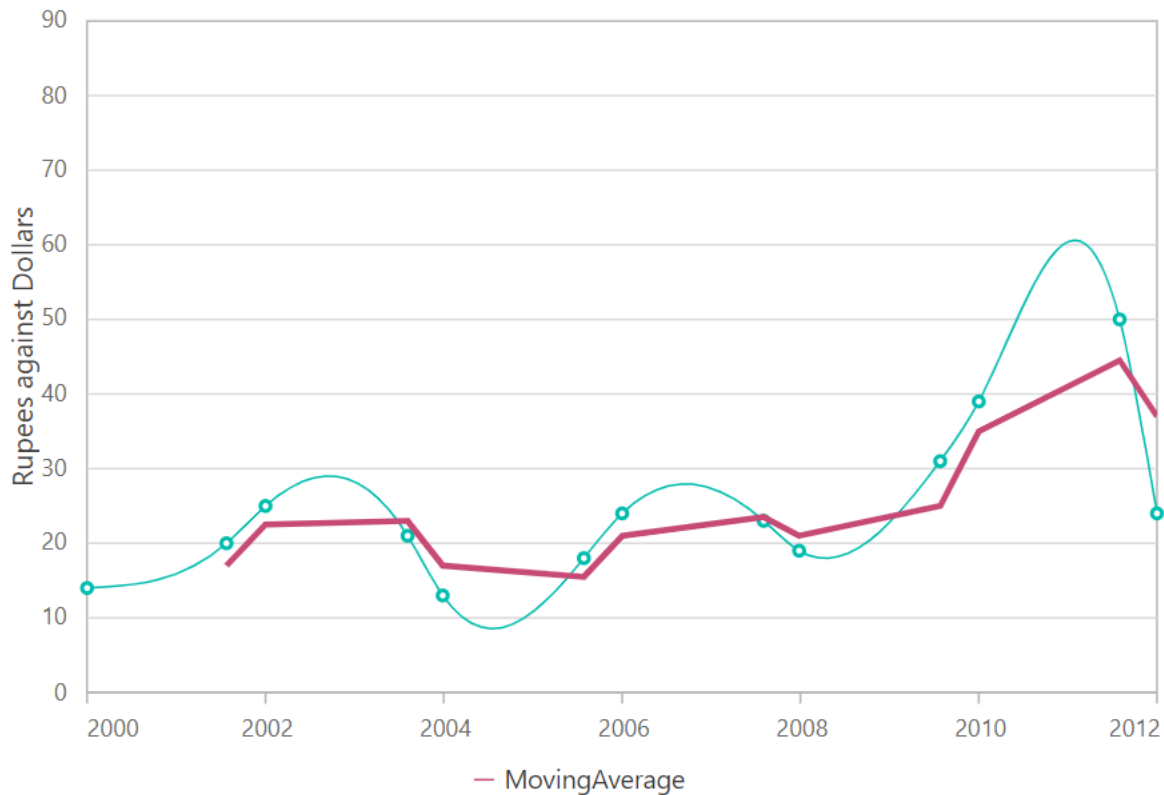
```
@using Syncfusion.Blazor.Charts
<SfChart Title="Historical Indian Rupee Rate (INR USD)">
  <ChartPrimaryXAxis LabelFormat="yyyy"
    ValueType="Syncfusion.Blazor.Charts.ValueType.DateTime"
    EdgeLabelPlacement="EdgeLabelPlacement.Shift">
    <ChartAxisMajorGridLines Width="0"></ChartAxisMajorGridLines>
  </ChartPrimaryXAxis>
  <ChartPrimaryYAxis Title="Rupees against Dollars">
    <ChartAxisLineStyle Width="0"></ChartAxisLineStyle>
    <ChartAxisMajorTickLines Width="0"></ChartAxisMajorTickLines>
  </ChartPrimaryYAxis>
  <ChartSeriesCollection>
    <ChartSeries DataSource="@Data" XName="XValue" YName="YValue"
      Type="ChartSeriesType.Spline">
      <ChartMarker Visible="true">
      </ChartMarker>
    </ChartSeries>
    <ChartTrendlines>
      <ChartTrendline Type="TrendlineTypes.MovingAverage" Width="3"
        Name="MovingAverage" Fill="#C64A75">
      </ChartTrendline>
    </ChartTrendlines>
  </ChartSeriesCollection>
</SfChart>
```

```

</ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code{
public class ChartData
{
public DateTime XValue { get; set; }
public double YValue { get; set; }
}
public List<ChartData> Data = new List<ChartData>
{
new ChartData { XValue = new DateTime(2000, 2, 11), YValue = 14 },
new ChartData { XValue = new DateTime(2001, 9, 4), YValue = 20 },
new ChartData { XValue = new DateTime(2002, 2, 11), YValue = 25 },
new ChartData { XValue = new DateTime(2003, 9, 16), YValue = 21 },
new ChartData { XValue = new DateTime(2004, 2, 7), YValue = 13},
new ChartData { XValue = new DateTime(2005, 9, 7), YValue = 18 },
new ChartData { XValue = new DateTime(2006, 2, 11), YValue = 24 },
new ChartData { XValue = new DateTime(2007, 9, 14), YValue = 23 },
new ChartData { XValue = new DateTime(2008, 2, 6), YValue = 19 },
new ChartData { XValue = new DateTime(2009, 9, 6), YValue = 31 },
new ChartData { XValue = new DateTime(2010, 2, 11), YValue = 39},
new ChartData { XValue = new DateTime(2011, 9, 11), YValue = 50 },
new ChartData { XValue = new DateTime(2012, 2, 11), YValue = 24 },
};
}

```

Historical Indian Rupee Rate (INR USD)



Forecasting

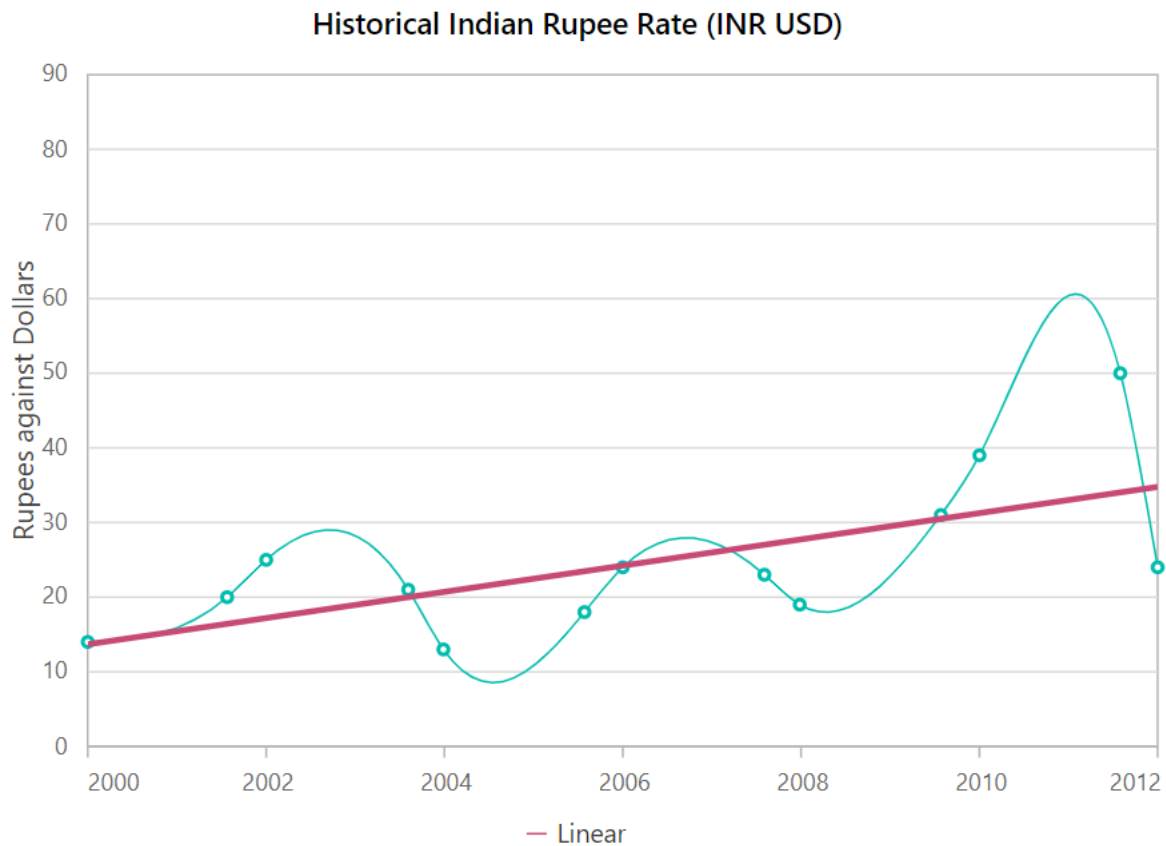
Trendlines forecasting is the prediction of future/past situations. There are two types of forecasting available: forward forecasting and backward forecasting.

Forward Forecasting

The value set to [ForwardForecast](#) property is used to calculate the distance between the current trend and the future trend.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart Title="Historical Indian Rupee Rate (INR USD)">
<ChartPrimaryXAxis LabelFormat="yyyy"
ValueType="Syncfusion.Blazor.Charts.ValueType.DateTime"
EdgeLabelPlacement="EdgeLabelPlacement.Shift">
<ChartAxisMajorGridLines Width="0"></ChartAxisMajorGridLines>
</ChartPrimaryXAxis>
<ChartPrimaryYAxis Title="Rupees against Dollars">
<ChartAxisLineStyle Width="0"></ChartAxisLineStyle>
<ChartAxisMajorTickLines Width="0"></ChartAxisMajorTickLines>
</ChartPrimaryYAxis>
<ChartSeriesCollection>
<ChartSeries DataSource="@Data" XName="XValue" YName="YValue"
Type="ChartSeriesType.Spline">
<ChartMarker Visible="true">
</ChartMarker>
<ChartTrendlines>
<ChartTrendline Type="TrendlineTypes.Linear" Width="3" Name="Linear"
Fill="#C64A75" ForwardForecast="5">
</ChartTrendline>
</ChartTrendlines>
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code{
public class ChartData
{
public DateTime XValue { get; set; }
public double YValue { get; set; }
}
public List<ChartData> Data = new List<ChartData>
{
new ChartData { XValue = new DateTime(2000, 2, 11), YValue = 14 },
new ChartData { XValue = new DateTime(2001, 9, 4), YValue = 20 },
new ChartData { XValue = new DateTime(2002, 2, 11), YValue = 25 },
new ChartData { XValue = new DateTime(2003, 9, 16), YValue = 21 },
new ChartData { XValue = new DateTime(2004, 2, 7), YValue = 13},
new ChartData { XValue = new DateTime(2005, 9, 7), YValue = 18 },
new ChartData { XValue = new DateTime(2006, 2, 11), YValue = 24 },
new ChartData { XValue = new DateTime(2007, 9, 14), YValue = 23 },
new ChartData { XValue = new DateTime(2008, 2, 6), YValue = 19 },
new ChartData { XValue = new DateTime(2009, 9, 6), YValue = 31 },
new ChartData { XValue = new DateTime(2010, 2, 11), YValue = 39},
new ChartData { XValue = new DateTime(2011, 9, 11), YValue = 50 },
new ChartData { XValue = new DateTime(2012, 2, 11), YValue = 24 },
};
}
```

Backward Forecasting

The value set to [BackwardForecast](#) property is used to determine historical trends.

ASPX-CS

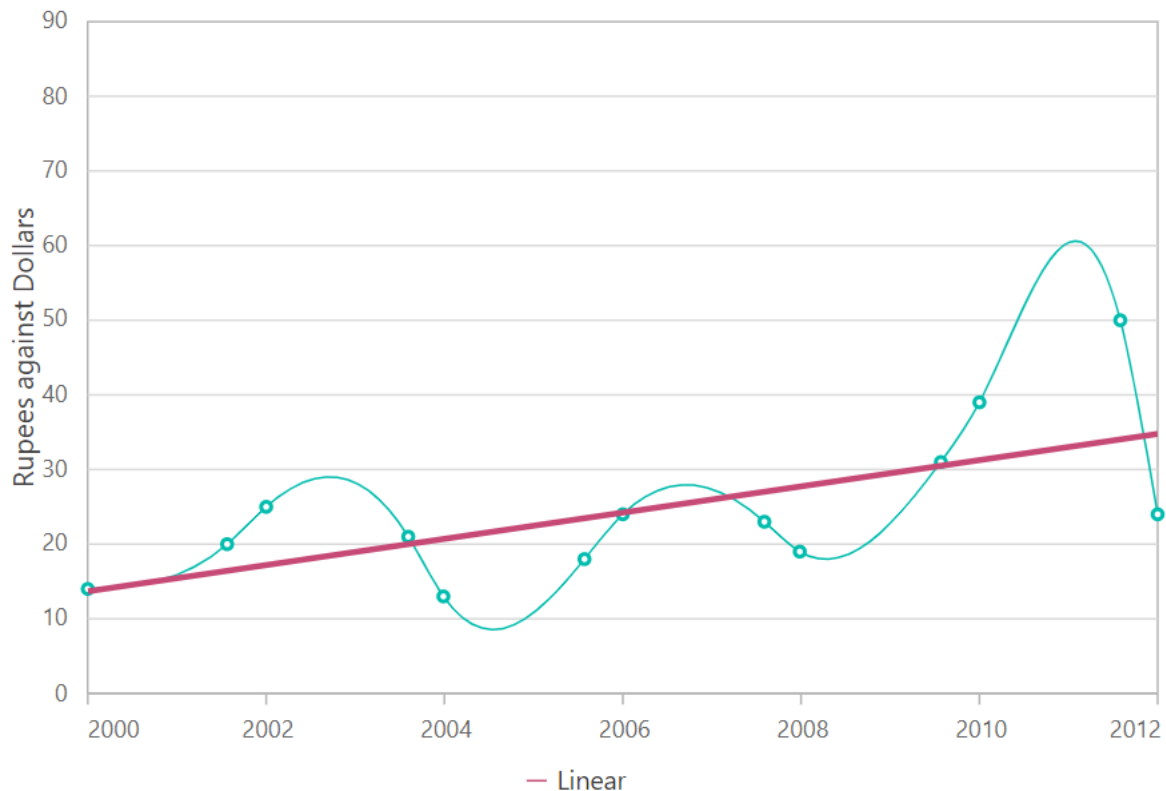
```
@using Syncfusion.Blazor.Charts
<SfChart Title="Historical Indian Rupee Rate (INR USD)">
  <ChartPrimaryXAxis LabelFormat="yyyy"
  ValueType="Syncfusion.Blazor.Charts.ValueType.DateTime"
  EdgeLabelPlacement="EdgeLabelPlacement.Shift">
    <ChartAxisMajorGridLines Width="0"></ChartAxisMajorGridLines>
  </ChartPrimaryXAxis>
  <ChartPrimaryYAxis Title="Rupees against Dollars">
    <ChartAxisLineStyle Width="0"></ChartAxisLineStyle>
    <ChartAxisMajorTickLines Width="0"></ChartAxisMajorTickLines>
  </ChartPrimaryYAxis>
  <ChartSeriesCollection>
    <ChartSeries DataSource="@Data" XName="XValue" YName="YValue"
    Type="ChartSeriesType.Spline">
      <ChartMarker Visible="true">
      </ChartMarker>
    </ChartSeries>
    <ChartTrendlines>
      <ChartTrendline Type="TrendlineTypes.Linear" Width="3" Name="Linear"
      Fill="#C64A75" BackwardForecast="5">
      </ChartTrendline>
    </ChartTrendlines>
  </ChartSeriesCollection>
</SfChart>
```

```

</ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code{
public class ChartData
{
public DateTime XValue { get; set; }
public double YValue { get; set; }
}
public List<ChartData> Data = new List<ChartData>
{
new ChartData { XValue = new DateTime(2000, 2, 11), YValue = 14 },
new ChartData { XValue = new DateTime(2001, 9, 4), YValue = 20 },
new ChartData { XValue = new DateTime(2002, 2, 11), YValue = 25 },
new ChartData { XValue = new DateTime(2003, 9, 16), YValue = 21 },
new ChartData { XValue = new DateTime(2004, 2, 7), YValue = 13},
new ChartData { XValue = new DateTime(2005, 9, 7), YValue = 18 },
new ChartData { XValue = new DateTime(2006, 2, 11), YValue = 24 },
new ChartData { XValue = new DateTime(2007, 9, 14), YValue = 23 },
new ChartData { XValue = new DateTime(2008, 2, 6), YValue = 19 },
new ChartData { XValue = new DateTime(2009, 9, 6), YValue = 31 },
new ChartData { XValue = new DateTime(2010, 2, 11), YValue = 39},
new ChartData { XValue = new DateTime(2011, 9, 11), YValue = 50 },
new ChartData { XValue = new DateTime(2012, 2, 11), YValue = 24 },
};
}

```

Historical Indian Rupee Rate (INR USD)



Trendlines Customization

The [Fill](#) and [Width](#) properties are used to customize the appearance of the trendline.

Refer to our [Blazor Charts](#) feature tour page for its groundbreaking feature representations and also explore our [Blazor Chart Example](#) to know various chart types and how to represent time-dependent data, showing trends at equal intervals.

See Also

- [Data label](#)
- [Tooltip](#)
- [Marker](#)

Internationalization in Blazor Charts Component

Internationalization is the process of designing and developing an component that can be easily adapted for, users from any culture, region, or language. Below elements in the [Chart](#) display content based on the internationalization configuration.

- Data label.
- Axis label.
- Tooltip.

<!-- markdownlint-disable MD036 -->

Globalization

[LabelFormat](#) property in axis is used to globalize the number, date and time value of the elements such as [Axis label](#), [Data label](#), and [Tooltip](#) in the [Chart](#) component.

In the below example, axis, point and tooltip labels are globalized to EUR.

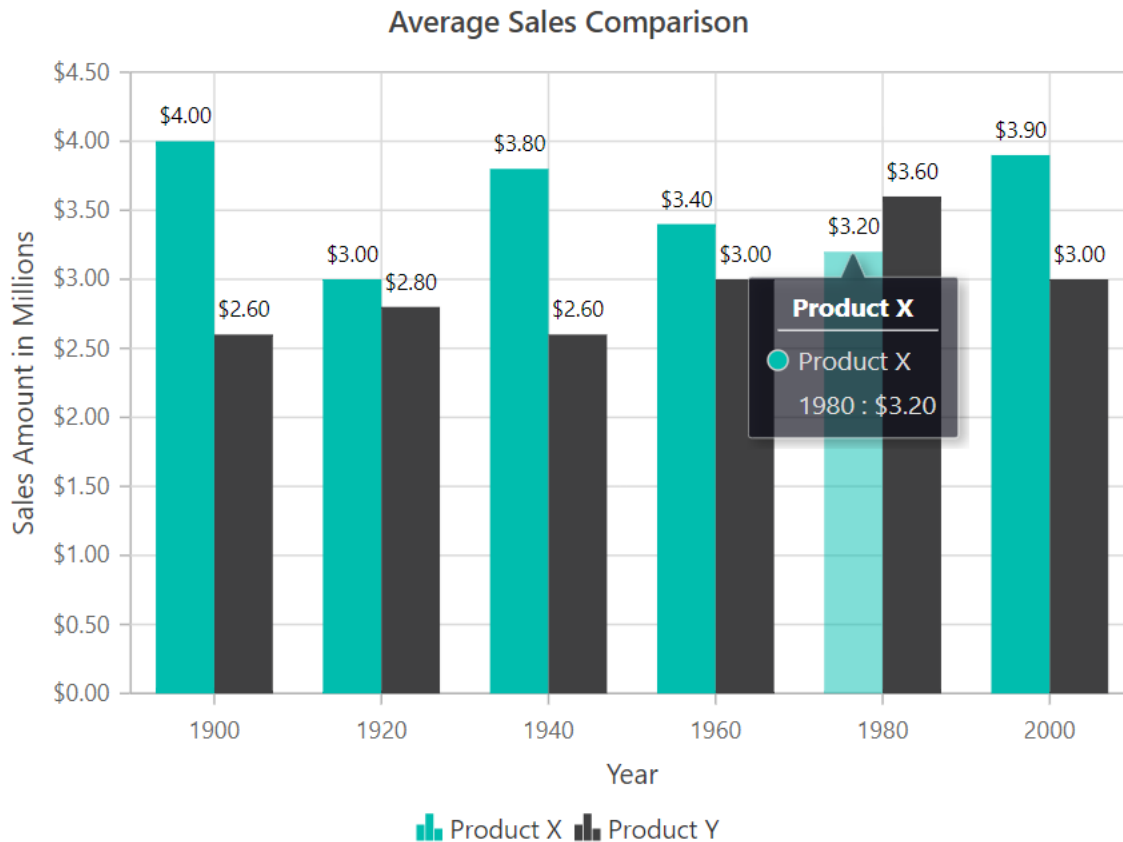
ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart Title="Average Sales Comparison">
  <ChartPrimaryXAxis Title="Year"></ChartPrimaryXAxis>
  <ChartPrimaryYAxis LabelFormat="c" Title="Sales Amount in Millions">
</ChartPrimaryYAxis>
  <ChartTooltipSettings Enable="true" Format="{series.name} <br>{point.x} :
  {point.y}">
</ChartTooltipSettings>
  <ChartSeriesCollection>
    <ChartSeries DataSource="@SalesReports" XName="X" YName="Y"
    Type="ChartSeriesType.Column" Name="Product X">
      <ChartMarker>
        <ChartDataLabel Visible="true"></ChartDataLabel>
      </ChartMarker>
    </ChartSeries>
    <ChartSeries DataSource="@SalesReports" XName="X" YName="Y1"
    Type="ChartSeriesType.Column" Name="Product Y">
      <ChartMarker>
        <ChartDataLabel Visible="true"></ChartDataLabel>
      </ChartMarker>
    </ChartSeries>
  </ChartSeriesCollection>
```

```

</SfChart>
@code{
public class ChartData
{
public double X { get; set; }
public double Y { get; set; }
public double Y1 { get; set; }
}
public List<ChartData> SalesReports = new List<ChartData>
{
new ChartData {X= 1900, Y= 4, Y1= 2.6 },
new ChartData{ X= 1920, Y= 3.0, Y1= 2.8 },
new ChartData{ X= 1940, Y= 3.8, Y1= 2.6},
new ChartData{ X= 1960, Y= 3.4, Y1= 3 },
new ChartData{ X= 1980, Y= 3.2, Y1= 3.6 },
new ChartData{ X= 2000, Y= 3.9, Y1= 3 }
};
}

```



Label Format

Learn more about axis label format in-relation to axis types from the pages below.

- [Numeric Label Format](#)
- [DateTime Label Format](#)
- [Logarithmic Label Format](#)
- [Custom Label Format](#)

Refer to our [Blazor Charts](#) feature tour page for its groundbreaking feature representations and also explore our [Blazor Chart Example](#) to know various chart types and how to represent time-dependent data, showing trends at equal intervals.

See Also

- [Data label](#)
- [Tooltip](#)
- [Legend](#)
- [Marker](#)

Localization in Blazor Charts Component

The static text of the Chart was translated using a Resource file (**.resx**).

The Resource file is an XML file that contains the strings (key and value pairs) that you want to translate. For further information on how to configure and use localization in the ASP.NET Core application framework, see the [Localization](#) link.

- Add the **.resx** file to the [Resources](#) folder and fill in the **Name** column with the key value (Locale Keywords) and the translated string in the **Value** column.

Blazor Server Side

In the server side Blazor samples, the following examples show how to activate **Localization** for charts.

- Open the **Startup.cs** file and, in the **ConfigureServices** function, add the following configuration.

CSHARP

```
using Syncfusion.Blazor;
using System.Globalization;
using Microsoft.AspNetCore.Localization;
namespace BlazorApplication
{
    public class Startup
    {
        ....
        ....
        public void ConfigureServices(IServiceCollection services)
        {
            ....
            ....
            services.AddSyncfusionBlazor();
            services.AddLocalization(options => options.ResourcesPath = "Resources");
            services.Configure<RequestLocalizationOptions>(options =>
            {
                // define the list of cultures your app will support
                var supportedCultures = new List<CultureInfo>()
                {
                    new CultureInfo("de")
                };
                // set the default culture
                options.DefaultRequestCulture = new RequestCulture("de");
                options.SupportedCultures = supportedCultures;
            });
        }
    }
}
```

```
options.SupportedUICultures = supportedCultures;
options.RequestCultureProviders = new List<IRequestCultureProvider>() {
    new QueryStringRequestCultureProvider() // Here, You can also use other
    localization provider
};
});
services.AddSingleton(typeof(ISyncfusionStringLocalizer),
    typeof(SampleLocalizer));
}
}
```

Add [UseRequestLocalization\(\)](#) middle-ware in Configure method in **Startup.cs** file to get browser Culture Information.

- Then, write a **class** by inheriting **ISyncfusionStringLocalizer** interface and override the Manager property to get the resource file details from the application end.

CSHARP

```
using Syncfusion.Blazor;
namespace BlazorServer
{
    public class SampleLocalizer : ISyncfusionStringLocalizer
    {
        public string GetText(string key)
        {
            return this.ResourceManager.GetString(key);
        }
        public System.Resources.ResourceManager ResourceManager
        {
            get
            {
                return BlazorServer.Resources.SfResources.ResourceManager;
            }
        }
    }
}
```

Blazor WebAssembly

In Client side Blazor samples, the following examples show how to activate **Localization** for charts.

- Open the **Program.cs** file and add the below configuration in the **Main** function as follows.

CSHARP

```
using Syncfusion.Blazor;
using System.Globalization;
namespace ClientApplication
{
    public class Program
    {
        public static async Task Main(string[] args)
```

```
{
var builder = WebAssemblyHostBuilder.CreateDefault(args);
builder.RootComponents.Add<App>("app");
builder.Services.AddTransient(sp => new HttpClient { BaseAddress = new
Uri(builder.HostEnvironment.BaseAddress) });
builder.Services.AddSyncfusionBlazor();
// Register the Syncfusion locale service to customize the SyncfusionBlazor
component locale culture
builder.Services.AddSingleton(typeof(ISyncfusionStringLocalizer),
typeof(SyncfusionLocalizer));
// Set the default culture of the application
CultureInfo.DefaultThreadCurrentCulture = new CultureInfo("de");
CultureInfo.DefaultThreadCurrentUICulture = new CultureInfo("de");
await builder.Build().RunAsync();
}
}
```

- Then, create a `~/Shared/SyncfusionLocalizer.cs` file and implement `ISyncfusionStringLocalizer` interface to the class and override the `ResourceManager` property to get the resource file details from the application end.

CSHARP

```
using Syncfusion.Blazor;
public class SyncfusionLocalizer : ISyncfusionStringLocalizer
{
// To get the locale key from mapped resources file
public string GetText(string key)
{
return this.ResourceManager.GetString(key);
}
// To access the resource file and get the exact value for locale key
public System.Resources.ResourceManager ResourceManager
{
get
{
// Replace the ApplicationNamespace with your application name.
return ClientApplication.Resources.SfResources.ResourceManager;
}
}
}
```

You can refer more details about localization [here](#).

Refer to our [Blazor Charts](#) feature tour page for its groundbreaking feature representations and also explore our [Blazor Chart Example](#) to know various chart types and how to represent time-dependent data, showing trends at equal intervals.

See Also

- [Data label](#)
- [Tooltip](#)

- [Legend](#)
- [Marker](#)

Events in Blazor Charts Component

In this section, we have provided a list of chart component events that will be triggered for appropriate chart actions.

The events should be provided to the chart using [ChartEvents](#) component.

From v18.4.*, we have added few additional events for the chart component

Event Name |

Resized | [SizeChanged](#)

ScrollChanged | [OnScrollChanged](#)

OnScrollEnd | [OnScrollChanged](#)

OnScrollStart | [OnScrollChanged](#)

AfterExport | [OnExportComplete](#)

OnPrint | [OnPrintComplete](#)

DragStart | [OnDataEdit](#)

DragEnd | [OnDataEditCompleted](#)

LegendClick | [OnLegendClick](#)

MultiLevelLabelClick | [OnMultiLevelLabelClick](#)

OnSelectionComplete | [OnSelectionChanged](#)

OnDragComplete | [OnSelectionChanged](#)

From v18.4.*, We have removed the following previous release events from chart component

Event Name |

OnAnimationComplete |

OnChartMouseClicked |

OnChartMouseDown |

OnChartMouseLeave |

OnChartMouseMove |

OnChartMouseUp |

PointMoved |

BeforeExport |

Load |

OnPointDoubleClick |

PointMoved |

OnZoomStart

After the zoom selection is made, the [OnZoomStart](#) event is triggered.

Arguments

Below property available in the the [ZoomingEventArgs](#).

- [AxisCollection](#) – Specifies the collection of the axis.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart>
<ChartEvents OnZoomStart="OnZoomingEvent"></ChartEvents>
<ChartPrimaryXAxis
ValueType="Syncfusion.Blazor.Charts.ValueType.Category"></ChartPrimaryXAxis>
<ChartSeriesCollection>
<ChartSeries DataSource="@Sales" XName="Month" YName="SalesValue"
Type="ChartSeriesType.Column">
</ChartSeries>
</ChartSeriesCollection>
<ChartZoomSettings EnableSelectionZooming="true"></ChartZoomSettings>
</SfChart>
@code{
public class SalesInfo
{
public string Month { get; set; }
public double SalesValue { get; set; }
}
public List<SalesInfo> Sales = new List<SalesInfo>
{
new SalesInfo { Month = "Jan", SalesValue = 35 },
new SalesInfo { Month = "Feb", SalesValue = 28 },
new SalesInfo { Month = "Mar", SalesValue = 34 },
new SalesInfo { Month = "Apr", SalesValue = 32 },
new SalesInfo { Month = "May", SalesValue = 40 },
new SalesInfo { Month = "Jun", SalesValue = 32 },
new SalesInfo { Month = "Jul", SalesValue = 35 }
};
public void OnZoomingEvent(ZoomingEventArgs args)
{
// Here you can customize your code
}
}
```

OnZoomEnd

[OnZoomEnd](#) event triggers, after the zoom selection is completed.

Arguments

Below property available in the the [ZoomingEventArgs](#).

- [AxisCollection](#) – Specifies the collection of the axis.

ASPX-CS

```

@using Syncfusion.Blazor.Charts
<SfChart>
<ChartEvents OnZoomEnd="OnZoomingEvent"></ChartEvents>
<ChartPrimaryXAxis
ValueType="Syncfusion.Blazor.Charts.ValueType.Category"></ChartPrimaryXAxis>
<ChartSeriesCollection>
<ChartSeries DataSource="@Sales" XName="Month" YName="SalesValue"
Type="ChartSeriesType.Column">
</ChartSeries>
</ChartSeriesCollection>
<ChartZoomSettings EnableSelectionZooming="true"></ChartZoomSettings>
</SfChart>
@code{
public class SalesInfo
{
public string Month { get; set; }
public double SalesValue { get; set; }
}
public List<SalesInfo> Sales = new List<SalesInfo>
{
new SalesInfo { Month = "Jan", SalesValue = 35 },
new SalesInfo { Month = "Feb", SalesValue = 28 },
new SalesInfo { Month = "Mar", SalesValue = 34 },
new SalesInfo { Month = "Apr", SalesValue = 32 },
new SalesInfo { Month = "May", SalesValue = 40 },
new SalesInfo { Month = "Jun", SalesValue = 32 },
new SalesInfo { Month = "Jul", SalesValue = 35 }
};
public void OnZoomingEvent(ZoomingEventArgs args)
{
// Here you can customize your code
}
}

```

OnZooming

[OnZooming](#) event triggers, after the zoom selection is completed.

Arguments

Below property available in the the [ZoomingEventArgs](#).

- [AxisCollection](#) – Specifies the collection of the axis.

ASPX-CS

```

@using Syncfusion.Blazor.Charts
<SfChart>
<ChartEvents OnZooming="OnZoomingEvent"></ChartEvents>
<ChartPrimaryXAxis
ValueType="Syncfusion.Blazor.Charts.ValueType.Category"></ChartPrimaryXAxis>
<ChartSeriesCollection>
<ChartSeries DataSource="@Sales" XName="Month" YName="SalesValue"
Type="ChartSeriesType.Column">
</ChartSeries>
</ChartSeriesCollection>
<ChartZoomSettings EnableSelectionZooming="true"></ChartZoomSettings>

```

```

</SfChart>
@code{
public class SalesInfo
{
public string Month { get; set; }
public double SalesValue { get; set; }
}
public List<SalesInfo> Sales = new List<SalesInfo>
{
new SalesInfo { Month = "Jan", SalesValue = 35 },
new SalesInfo { Month = "Feb", SalesValue = 28 },
new SalesInfo { Month = "Mar", SalesValue = 34 },
new SalesInfo { Month = "Apr", SalesValue = 32 },
new SalesInfo { Month = "May", SalesValue = 40 },
new SalesInfo { Month = "Jun", SalesValue = 32 },
new SalesInfo { Month = "Jul", SalesValue = 35 }
};
public void OnZoomingEvent(ZoomingEventArgs args)
{
// Here you can customize your code
}
}

```

OnLegendItemRender

[OnLegendItemRender](#) event triggers, before the legend is rendered.

Arguments

Below properties are available in the the [LegendRenderEventArgs](#).

- [Fill](#) – Specifies the fill color of the legend item's icon.
- [MarkerShape](#) – Specifies the shape of the marker.
- [Shape](#) – Specifies the shape of the legend item's icon.
- [Text](#) – Specifies the text to be displayed in the legend item.

ASPX-CS

```

@using Syncfusion.Blazor.Charts
<SfChart>
<ChartEvents OnLegendItemRender="LegendEvent"></ChartEvents>
<ChartPrimaryXAxis
ValueType="Syncfusion.Blazor.Charts.ValueType.Category"></ChartPrimaryXAxis>
<ChartSeriesCollection>
<ChartSeries DataSource="@Sales" Name="Column" XName="Month"
YName="SalesValue" Type="ChartSeriesType.Column">
</ChartSeries>
<ChartSeries DataSource="@Sales" Name="Line" XName="Month"
YName="SalesValue" Type="ChartSeriesType.Line">
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code{
public class SalesInfo
{
public string Month { get; set; }
public double SalesValue { get; set; }
}

```

```

}
public List<SalesInfo> Sales = new List<SalesInfo>
{
    new SalesInfo { Month = "Jan", SalesValue = 35 },
    new SalesInfo { Month = "Feb", SalesValue = 28 },
    new SalesInfo { Month = "Mar", SalesValue = 34 },
    new SalesInfo { Month = "Apr", SalesValue = 32 },
    new SalesInfo { Month = "May", SalesValue = 40 },
    new SalesInfo { Month = "Jun", SalesValue = 32 },
    new SalesInfo { Month = "Jul", SalesValue = 35 }
};
public void LegendEvent(LegendRenderEventArgs args)
{
    // Here you can customize your code
}
}

```

OnDataLabelRender

[OnDataLabelRender](#) event triggers, before the data label for series is rendered.

Arguments

Below properties are available in the the [TextRenderEventArgs](#).

- [Border](#) – Specifies the color and the width of the data label border.
- [Color](#) – Specifies the text color of the data label.
- [Font](#) – Specifies the font information of the data label.
- [Template](#) – Provides the information about the data point to be used in the data label template.
- [Text](#) – Specifies the text to be displayed in the data label.

ASPX-CS

```

@using Syncfusion.Blazor.Charts
<SfChart>
<ChartEvents OnDataLabelRender="DataLabelEvent"></ChartEvents>
<ChartPrimaryXAxis
Value="Syncfusion.Blazor.Charts.ValueType.Category"></ChartPrimaryXAxis>
<ChartSeriesCollection>
<ChartSeries DataSource="@Sales" XName="Month" YName="SalesValue"
Type="ChartSeriesType.Column">
<ChartMarker>
<ChartDataLabel Visible="true"></ChartDataLabel>
</ChartMarker>
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code{
public class SalesInfo
{
    public string Month { get; set; }
    public double SalesValue { get; set; }
}
public List<SalesInfo> Sales = new List<SalesInfo>
{
    new SalesInfo { Month = "Jan", SalesValue = 35 },

```

```

new SalesInfo { Month = "Feb", SalesValue = 28 },
new SalesInfo { Month = "Mar", SalesValue = 34 },
new SalesInfo { Month = "Apr", SalesValue = 32 },
new SalesInfo { Month = "May", SalesValue = 40 },
new SalesInfo { Month = "Jun", SalesValue = 32 },
new SalesInfo { Month = "Jul", SalesValue = 35 }
};
public void DataLabelEvent(TextRenderEventArgs args)
{
    // Here you can customize your code
}
}

```

OnPointRender

[OnPointRender](#) event triggers, before each points for the series is rendered.

Arguments

Below properties are available in the the [PointRenderEventArgs](#).

- [Border](#) – Specifies the color and the width of the point border.
- [Fill](#) – Specifies the fill color of the point.
- [Height](#) – Specifies the current point's height.
- [Shape](#) – Specifies the marker shape of the point.
- [Width](#) – Specifies the current point's width.

ASPX-CS

```

@using Syncfusion.Blazor.Charts
<SfChart>
<ChartEvents OnPointRender="PointRenderEvent"></ChartEvents>
<ChartPrimaryXAxis
Value="Syncfusion.Blazor.Charts.ValueType.Category"></ChartPrimaryXAxis>
<ChartSeriesCollection>
<ChartSeries DataSource="@Sales" XName="Month" YName="SalesValue"
Type="ChartSeriesType.Column">
<ChartMarker Visible="true">
</ChartMarker>
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code{
public class SalesInfo
{
    public string Month { get; set; }
    public double SalesValue { get; set; }
}
public List<SalesInfo> Sales = new List<SalesInfo>
{
    new SalesInfo { Month = "Jan", SalesValue = 35 },
    new SalesInfo { Month = "Feb", SalesValue = 28 },
    new SalesInfo { Month = "Mar", SalesValue = 34 },
    new SalesInfo { Month = "Apr", SalesValue = 32 },
    new SalesInfo { Month = "May", SalesValue = 40 },
    new SalesInfo { Month = "Jun", SalesValue = 32 },
}

```

```

new SalesInfo { Month = "Jul", SalesValue = 35 }
};
public void PointRenderEvent(PointRenderEventArgs args)
{
    // Here you can customize your code
}
}

```

OnAxisLabelRender

[OnAxisLabelRender](#) event triggers, before each axis label is rendered.

Arguments

Below properties are available in the the [AxisLabelRenderEventArgs](#).

- [LabelStyle](#) – Specifies the font information of the axis label.
- [Text](#) – Specifies the text to be displayed in the axis label.
- [Value](#) – Specifies the value of the axis label.

ASPX-CS

```

@using Syncfusion.Blazor.Charts
<SfChart>
<ChartEvents OnAxisLabelRender="AxisLabelEvent"></ChartEvents>
<ChartPrimaryXAxis
ValueType="Syncfusion.Blazor.Charts.ValueType.Category"></ChartPrimaryXAxis>
<ChartSeriesCollection>
<ChartSeries DataSource="@Sales" XName="Month" YName="SalesValue"
Type="ChartSeriesType.Column">
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code{
public class SalesInfo
{
    public string Month { get; set; }
    public double SalesValue { get; set; }
}
public List<SalesInfo> Sales = new List<SalesInfo>
{
    new SalesInfo { Month = "Jan", SalesValue = 35 },
    new SalesInfo { Month = "Feb", SalesValue = 28 },
    new SalesInfo { Month = "Mar", SalesValue = 34 },
    new SalesInfo { Month = "Apr", SalesValue = 32 },
    new SalesInfo { Month = "May", SalesValue = 40 },
    new SalesInfo { Month = "Jun", SalesValue = 32 },
    new SalesInfo { Month = "Jul", SalesValue = 35 }
};
public void AxisLabelEvent(AxisLabelRenderEventArgs args)
{
    // Here you can customize your code
}
}

```

OnAxisLabelClick

[OnAxisLabelClick](#) event triggers, when x axis label is clicked.

Arguments

Below fields are available in the the [AxisLabelClickEventArgs](#).

- [Axis](#) – Specifies the current axis.
- [Chart](#) – Specifies the chart instance.
- [Index](#) – Specifies the index of the axis label.
- [LabelID](#) – Specifies the current axis label's element id.
- [Location](#) – Specifies the location of the axis label.
- [Text](#) – Specifies the text of the axis label.
- [Value](#) – Specifies the value of the axis label.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart>
<ChartEvents OnAxisLabelClick="AxisLabelClickEvent"></ChartEvents>
<ChartPrimaryXAxis
ValueType="Syncfusion.Blazor.Charts.ValueType.Category"></ChartPrimaryXAxis>
<ChartSeriesCollection>
<ChartSeries DataSource="@Sales" XName="Month" YName="SalesValue"
Type="ChartSeriesType.Column">
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code{
public class SalesInfo
{
public string Month { get; set; }
public double SalesValue { get; set; }
}
public List<SalesInfo> Sales = new List<SalesInfo>
{
new SalesInfo { Month = "Jan", SalesValue = 35 },
new SalesInfo { Month = "Feb", SalesValue = 28 },
new SalesInfo { Month = "Mar", SalesValue = 34 },
new SalesInfo { Month = "Apr", SalesValue = 32 },
new SalesInfo { Month = "May", SalesValue = 40 },
new SalesInfo { Month = "Jun", SalesValue = 32 },
new SalesInfo { Month = "Jul", SalesValue = 35 }
};
public void AxisLabelClickEvent(AxisLabelClickEventArgs args)
{
// Here you can customize your code
}
}
```

OnAxisActualRangeCalculated

[OnAxisActualRangeCalculated](#) event triggers, before each axis range is calculated.

Arguments

Below properties are available in the the [AxisRangeCalculatedEventArgs](#).

- [Interval](#) – Specifies the current interval of the axis.
- [Maximum](#) – Specifies the current maximum value of the axis.
- [Minimum](#) – Specifies current minimum value of the axis.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart>
<ChartEvents
OnAxisActualRangeCalculated="AxisActualRangeEvent"></ChartEvents>
<ChartPrimaryXAxis
ValueType="Syncfusion.Blazor.Charts.ValueType.Category"></ChartPrimaryXAxis>
<ChartSeriesCollection>
<ChartSeries DataSource="@Sales" XName="Month" YName="SalesValue"
Type="ChartSeriesType.Column">
<ChartMarker Visible="true">
</ChartMarker>
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code{
public class SalesInfo
{
public string Month { get; set; }
public double SalesValue { get; set; }
}
public List<SalesInfo> Sales = new List<SalesInfo>
{
new SalesInfo { Month = "Jan", SalesValue = 35 },
new SalesInfo { Month = "Feb", SalesValue = 28 },
new SalesInfo { Month = "Mar", SalesValue = 34 },
new SalesInfo { Month = "Apr", SalesValue = 32 },
new SalesInfo { Month = "May", SalesValue = 40 },
new SalesInfo { Month = "Jun", SalesValue = 32 },
new SalesInfo { Month = "Jul", SalesValue = 35 }
};
public void AxisActualRangeEvent(AxisRangeCalculatedEventArgs args)
{
// Here you can customize your code
}
}
```

OnAxisMultiLevelLabelRender

[OnAxisMultiLevelLabelRender](#) event triggers, while rendering multilevel labels.

Arguments

Below properties are available in the [AxisMultiLabelRenderEventArgs](#).

- [Alignment](#) – Specifies the alignment of the axis label.
- [Text](#) – Specifies the text to be displayed in the axis label.
- [TextStyle](#) – Specifies the text style of the axis label.

ASPX-CS


```

@using Syncfusion.Blazor.Charts
<SfChart>
<ChartEvents
OnAxisMultiLevelLabelRender="AxisMultiLevelLabelEvent"></ChartEvents>
<ChartPrimaryXAxis ValueType="Syncfusion.Blazor.Charts.ValueType.Category">
<ChartMultiLevelLabels>
<ChartMultiLevelLabel>
<ChartCategories>
<ChartCategory Start="0" End="3" Text="First_Half"></ChartCategory>
<ChartCategory Start="3" End="6" Text="Second_Half"></ChartCategory>
</ChartCategories>
</ChartMultiLevelLabel>
</ChartMultiLevelLabels>
</ChartPrimaryXAxis>
<ChartSeriesCollection>
<ChartSeries DataSource="@Sales" XName="Month" YName="SalesValue"
Type="ChartSeriesType.Column">
<ChartMarker Visible="true">
</ChartMarker>
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code{
public class SalesInfo
{
public string Month { get; set; }
public double SalesValue { get; set; }
}
public List<SalesInfo> Sales = new List<SalesInfo>
{
new SalesInfo { Month = "Jan", SalesValue = 35 },
new SalesInfo { Month = "Feb", SalesValue = 28 },
new SalesInfo { Month = "Mar", SalesValue = 34 },
new SalesInfo { Month = "Apr", SalesValue = 32 },
new SalesInfo { Month = "May", SalesValue = 40 },
new SalesInfo { Month = "Jun", SalesValue = 32 },
new SalesInfo { Month = "Jul", SalesValue = 35 }
};
public void AxisMultiLevelLabelEvent(AxisMultiLevelLabelRenderEventArgs args)
{
// Here you can customize your code
}
}

```

SizeChanged

[SizeChanged](#) event triggers after resizing of the chart.

Arguments

Below fields are available in the the [ResizeEventArgs](#).

- [CurrentSize](#) – Specifies the current size of the chart.
- [PreviousSize](#) – Specifies the previous size of the chart.

ASPX-CS

```

@using Syncfusion.Blazor.Charts
<SfChart>
<ChartEvents SizeChanged="@SizeChangedEvent"></ChartEvents>
<ChartPrimaryXAxis
ValueType="Syncfusion.Blazor.Charts.ValueType.Category"></ChartPrimaryXAxis>
<ChartSeriesCollection>
<ChartSeries DataSource="@Sales" XName="Month" YName="SalesValue"
Type="ChartSeriesType.Column">
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code{
public class SalesInfo
{
public string Month { get; set; }
public double SalesValue { get; set; }
}
public List<SalesInfo> Sales = new List<SalesInfo>
{
new SalesInfo { Month = "Jan", SalesValue = 35 },
new SalesInfo { Month = "Feb", SalesValue = 28 },
new SalesInfo { Month = "Mar", SalesValue = 34 },
new SalesInfo { Month = "Apr", SalesValue = 32 },
new SalesInfo { Month = "May", SalesValue = 40 },
new SalesInfo { Month = "Jun", SalesValue = 32 },
new SalesInfo { Month = "Jul", SalesValue = 35 }
};
public void SizeChangedEvent(ResizeEventArgs args)
{
// Here you can customize your code
}
}

```

OnScrollChanged

[OnScrollChanged](#) event triggers while scrolling the chart.

Arguments

Below properties are available in the the [ScrollEventArgs](#).

- [Axis](#) – Specifies the current axis that is scrolled.
- [CurrentRange](#) – Specifies the current range of the axis.
- [PreviousAxisRange](#) – Specifies the current axis.
- [PreviousRange](#) – Specifies the previous range of the axis.
- [PreviousZoomFactor](#) – Specifies the previous zoom factor value.
- [PreviousZoomPosition](#) – Specifies the previous zoom position value.
- [Range](#) – Specifies the range of the axis.
- [ZoomFactor](#) – Specifies the current zoom factor value.
- [ZoomPosition](#) – Specifies the current zoom position value.

ASPX-CS

```

@using Syncfusion.Blazor.Charts
<SfChart>
<ChartEvents OnScrollChanged="ScrollChangeEvent"></ChartEvents>

```

```

<ChartPrimaryXAxis ValueType="Syncfusion.Blazor.Charts.ValueType.Category"
ZoomFactor="0.5" ZoomPosition="0.2">
<ChartAxisScrollbarSettings Enable="true"></ChartAxisScrollbarSettings>
</ChartPrimaryXAxis>
<ChartSeriesCollection>
<ChartSeries DataSource="@Sales" XName="Month" YName="SalesValue"
Type="ChartSeriesType.Column">
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code{
public class SalesInfo
{
public string Month { get; set; }
public double SalesValue { get; set; }
}
public List<SalesInfo> Sales = new List<SalesInfo>
{
new SalesInfo { Month = "Jan", SalesValue = 35 },
new SalesInfo { Month = "Feb", SalesValue = 28 },
new SalesInfo { Month = "Mar", SalesValue = 34 },
new SalesInfo { Month = "Apr", SalesValue = 32 },
new SalesInfo { Month = "May", SalesValue = 40 },
new SalesInfo { Month = "Jun", SalesValue = 32 },
new SalesInfo { Month = "Jul", SalesValue = 35 }
};
public void ScrollChangeEvent(ScrollEventArgs args)
{
// Here you can customize your code
}
}

```

OnExportComplete

[OnExportComplete](#) event triggers after exporting the chart.

Arguments

Below field is available in the the [ExportEventArgs](#).

- [DataUrl](#) – Specifies the DataUrl of the exported file.

ASPX-CS

```

@using Syncfusion.Blazor.Charts
<button class="btn-success" @onclick="Export">Export</button>
<SfChart @ref="chart">
<ChartEvents OnExportComplete="ExportCompleteEvent"></ChartEvents>
<ChartPrimaryXAxis ValueType="Syncfusion.Blazor.Charts.ValueType.Category">
</ChartPrimaryXAxis>
<ChartSeriesCollection>
<ChartSeries DataSource="@Sales" XName="Month" YName="SalesValue"
Type="ChartSeriesType.Column">
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code{

```

```

SfChart chart;
public class SalesInfo
{
    public string Month { get; set; }
    public double SalesValue { get; set; }
}
public List<SalesInfo> Sales = new List<SalesInfo>
{
    new SalesInfo { Month = "Jan", SalesValue = 35 },
    new SalesInfo { Month = "Feb", SalesValue = 28 },
    new SalesInfo { Month = "Mar", SalesValue = 34 },
    new SalesInfo { Month = "Apr", SalesValue = 32 },
    new SalesInfo { Month = "May", SalesValue = 40 },
    new SalesInfo { Month = "Jun", SalesValue = 32 },
    new SalesInfo { Month = "Jul", SalesValue = 35 }
};
public void Export()
{
    chart.Export(ExportType.JPEG, "Charts");
}
public void ExportCompleteEvent(ExportEventArgs args)
{
    // Here you can customize your code
}
}

```

OnDataEdit

[OnDataEdit](#) event triggers, while dragging the data point.

Arguments

Below fields are available in the the [DataEditingEventArgs](#).

- [NewValue](#) – Specifies the new value of the current point.
- [OldValue](#) – Specifies the previous value of the current point.
- [Point](#) – Specifies the current point which is being edited.
- [PointIndex](#) – Specifies the index of the current point.
- [Series](#) – Specifies the current chart series whose point is being edited.
- [SeriesIndex](#) – Specifies the index of the current series.

ASPX-CS

```

@using Syncfusion.Blazor.Charts
<SfChart>
<ChartEvents OnDataEdit="DataEditEvent"></ChartEvents>
<ChartPrimaryXAxis ValueType="Syncfusion.Blazor.Charts.ValueType.Category">
</ChartPrimaryXAxis>
<ChartSeriesCollection>
<ChartSeries DataSource="@Sales" XName="Month" YName="SalesValue"
Type="ChartSeriesType.Column">
<ChartDataEditSettings Enable="true"></ChartDataEditSettings>
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code{

```

```

public class SalesInfo
{
    public string Month { get; set; }
    public double SalesValue { get; set; }
}
public List<SalesInfo> Sales = new List<SalesInfo>
{
    new SalesInfo { Month = "Jan", SalesValue = 35 },
    new SalesInfo { Month = "Feb", SalesValue = 28 },
    new SalesInfo { Month = "Mar", SalesValue = 34 },
    new SalesInfo { Month = "Apr", SalesValue = 32 },
    new SalesInfo { Month = "May", SalesValue = 40 },
    new SalesInfo { Month = "Jun", SalesValue = 32 },
    new SalesInfo { Month = "Jul", SalesValue = 35 }
};
public void DataEditEvent(DataEditingEventArgs arg)
{
    // Here you can customize your code
}
}

```

OnDataEditCompleted

[OnDataEditCompleted](#) event triggers when the point drag completed.

Arguments

Below fields are available in the the [DataEditingEventArgs](#).

- [NewValue](#) – Specifies the new value of the current point.
- [OldValue](#) – Specifies the previous value of the current point.
- [Point](#) – Specifies the current point which is being edited.
- [PointIndex](#) – Specifies the index of the current point.
- [Series](#) – Specifies the current chart series whose point is being edited.
- [SeriesIndex](#) – Specifies the index of the current series.

ASPX-CS

```

@using Syncfusion.Blazor.Charts
<SfChart>
<ChartEvents OnDataEditCompleted="DataEditEvent"></ChartEvents>
<ChartPrimaryXAxis ValueType="Syncfusion.Blazor.Charts.ValueType.Category">
</ChartPrimaryXAxis>
<ChartSeriesCollection>
<ChartSeries DataSource="@Sales" XName="Month" YName="SalesValue"
Type="ChartSeriesType.Column">
<ChartDataEditSettings Enable="true"></ChartDataEditSettings>
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code{
public class SalesInfo
{
    public string Month { get; set; }
    public double SalesValue { get; set; }
}
}

```

```

public List<SalesInfo> Sales = new List<SalesInfo>
{
    new SalesInfo { Month = "Jan", SalesValue = 35 },
    new SalesInfo { Month = "Feb", SalesValue = 28 },
    new SalesInfo { Month = "Mar", SalesValue = 34 },
    new SalesInfo { Month = "Apr", SalesValue = 32 },
    new SalesInfo { Month = "May", SalesValue = 40 },
    new SalesInfo { Month = "Jun", SalesValue = 32 },
    new SalesInfo { Month = "Jul", SalesValue = 35 }
};
public void DataEditEvent(DataEditingEventArgs arg)
{
    // Here you can customize your code
}
}

```

OnLegendClick

[OnLegendClick](#) event triggers after legend click.

Arguments

Below properties are available in the the [LegendClickEventArgs](#).

- [LegendShape](#) – Specifies the shape of the legend item.

ASPX-CS

```

@using Syncfusion.Blazor.Charts
<SfChart>
<ChartEvents OnLegendClick="LegendClickEvent"></ChartEvents>
<ChartPrimaryXAxis
    ValueType="Syncfusion.Blazor.Charts.ValueType.Category"></ChartPrimaryXAxis>
<ChartSeriesCollection>
<ChartSeries DataSource="@Sales" Name="Column" XName="Month"
    YName="SalesValue" Type="ChartSeriesType.Column">
</ChartSeries>
<ChartSeries DataSource="@Sales" Name="Line" XName="Month"
    YName="SalesValue" Type="ChartSeriesType.Line">
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code{
public class SalesInfo
{
    public string Month { get; set; }
    public double SalesValue { get; set; }
}
public List<SalesInfo> Sales = new List<SalesInfo>
{
    new SalesInfo { Month = "Jan", SalesValue = 35 },
    new SalesInfo { Month = "Feb", SalesValue = 28 },
    new SalesInfo { Month = "Mar", SalesValue = 34 },
    new SalesInfo { Month = "Apr", SalesValue = 32 },
    new SalesInfo { Month = "May", SalesValue = 40 },
    new SalesInfo { Month = "Jun", SalesValue = 32 },
    new SalesInfo { Month = "Jul", SalesValue = 35 }
}

```

```
};
public void LegendClickEvent(LegendClickEventArgs args)
{
    // Here you can customize your code
}
}
```

OnMultiLevelLabelClick

[OnMultiLevelLabelClick](#) event triggers after click on multilevellabelclick.

Arguments

Below fields are available in the the [MultiLevelLabelClickEventArgs](#).

- [Axis](#) – Specifies the axis of the clicked label.
- [CustomAttributes](#) – Specifies the custom objects for multi level labels.
- [End](#) – Specifies the end value of the multi level labels.
- [Level](#) – Specifies the current level of the label.
- [Start](#) – Specifies the start value of the multi level labels
- [Text](#) – Specifies the text of the current label.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart>
<ChartEvents OnMultiLevelLabelClick="MultiLabelClickEvent"></ChartEvents>
<ChartPrimaryXAxis ValueType="Syncfusion.Blazor.Charts.ValueType.Category">
<ChartMultiLevelLabels>
<ChartMultiLevelLabel>
<ChartCategories>
<ChartCategory Start="0" End="3" Text="First_Half"></ChartCategory>
<ChartCategory Start="3" End="6" Text="Second_Half"></ChartCategory>
</ChartCategories>
</ChartMultiLevelLabel>
</ChartMultiLevelLabels>
</ChartPrimaryXAxis>
<ChartSeriesCollection>
<ChartSeries DataSource="@Sales" XName="Month" YName="SalesValue"
Type="ChartSeriesType.Column">
<ChartMarker Visible="true">
</ChartMarker>
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code{
public class SalesInfo
{
    public string Month { get; set; }
    public double SalesValue { get; set; }
}
public List<SalesInfo> Sales = new List<SalesInfo>
{
    new SalesInfo { Month = "Jan", SalesValue = 35 },
    new SalesInfo { Month = "Feb", SalesValue = 28 },
    new SalesInfo { Month = "Mar", SalesValue = 34 },
}
```

```

new SalesInfo { Month = "Apr", SalesValue = 32 },
new SalesInfo { Month = "May", SalesValue = 40 },
new SalesInfo { Month = "Jun", SalesValue = 32 },
new SalesInfo { Month = "Jul", SalesValue = 35 }
};
public void MultiLabelClickEvent(MultiLevelLabelClickEventArgs args)
{
    // Here you can customize your code
}
}

```

OnSelectionChanged

[OnSelectionChanged](#) event triggers after the selection is completed.

Arguments

Below property available in the the [SelectionCompleteEventArgs](#).

- [SelectedDataValues](#) – Specifies the selected Data X, Y values.

ASPX-CS

```

@using Syncfusion.Blazor.Charts
<SfChart SelectionMode="SelectionMode.Point">
<ChartEvents OnSelectionChanged="SelectionChangedEvent"></ChartEvents>
<ChartPrimaryXAxis ValueType="Syncfusion.Blazor.Charts.ValueType.Category">
</ChartPrimaryXAxis>
<ChartSeriesCollection>
<ChartSeries DataSource="@Sales" XName="Month" YName="SalesValue"
Type="ChartSeriesType.Column">
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code{
public class SalesInfo
{
    public string Month { get; set; }
    public double SalesValue { get; set; }
}
public List<SalesInfo> Sales = new List<SalesInfo>
{
    new SalesInfo { Month = "Jan", SalesValue = 35 },
    new SalesInfo { Month = "Feb", SalesValue = 28 },
    new SalesInfo { Month = "Mar", SalesValue = 34 },
    new SalesInfo { Month = "Apr", SalesValue = 32 },
    new SalesInfo { Month = "May", SalesValue = 40 },
    new SalesInfo { Month = "Jun", SalesValue = 32 },
    new SalesInfo { Month = "Jul", SalesValue = 35 }
};
public void SelectionChangedEvent(SelectionCompleteEventArgs args)
{
    // Here you can customize your code
}
}

```


Loaded

Loaded event triggers after chart load.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart>
<ChartEvents Loaded="LoadedEvent"></ChartEvents>
<ChartPrimaryXAxis
Value="Syncfusion.Blazor.Charts.ValueType.Category"></ChartPrimaryXAxis>
<ChartSeriesCollection>
<ChartSeries DataSource="@Sales" XName="Month" YName="SalesValue"
Type="ChartSeriesType.Column">
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code{
public class SalesInfo
{
public string Month { get; set; }
public double SalesValue { get; set; }
}
public List<SalesInfo> Sales = new List<SalesInfo>
{
new SalesInfo { Month = "Jan", SalesValue = 35 },
new SalesInfo { Month = "Feb", SalesValue = 28 },
new SalesInfo { Month = "Mar", SalesValue = 34 },
new SalesInfo { Month = "Apr", SalesValue = 32 },
new SalesInfo { Month = "May", SalesValue = 40 },
new SalesInfo { Month = "Jun", SalesValue = 32 },
new SalesInfo { Month = "Jul", SalesValue = 35 }
};
public void LoadedEvent(LoadedEventArgs args)
{
// Here you can customize your code
}
}
```

OnPointClick

OnPointClick event triggers on point click.

Arguments

Below fields are available in the the [PointEventArgs](#).

- [Chart](#) – Specifies the current chart instance.
- [PageX](#) – Specifies the current window page x location.
- [PageY](#) – Specifies the current window page y location.
- [Point](#) – Specifies the current point which is clicked.
- [PointIndex](#) – Specifies the index of the current point.
- [Series](#) – Specifies the current series.
- [SeriesIndex](#) – Specifies the current series index.
- [X](#) – Specifies the x coordinate of the current mouse click.
- [Y](#) – Specifies the y coordinate of the current mouse click.

ASPX-CS

```

@using Syncfusion.Blazor.Charts
<SfChart>
<ChartEvents OnPointClick="PointClickEvent"></ChartEvents>
<ChartPrimaryXAxis
Value="Syncfusion.Blazor.Charts.ValueType.Category"></ChartPrimaryXAxis>
<ChartSeriesCollection>
<ChartSeries DataSource="@Sales" XName="Month" YName="SalesValue"
Type="ChartSeriesType.Column">
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
@code{
public class SalesInfo
{
public string Month { get; set; }
public double SalesValue { get; set; }
}
public List<SalesInfo> Sales = new List<SalesInfo>
{
new SalesInfo { Month = "Jan", SalesValue = 35 },
new SalesInfo { Month = "Feb", SalesValue = 28 },
new SalesInfo { Month = "Mar", SalesValue = 34 },
new SalesInfo { Month = "Apr", SalesValue = 32 },
new SalesInfo { Month = "May", SalesValue = 40 },
new SalesInfo { Month = "Jun", SalesValue = 32 },
new SalesInfo { Month = "Jul", SalesValue = 35 }
};
public void PointClickEvent(PointEventArgs args)
{
// Here you can customize your code
}
}

```

TooltipRender

[TooltipRender](#) event triggers, before the tooltip for series is rendered.

Arguments

Below property available in the the [TooltipRenderEventArgs](#).

- [HeaderText](#) – Specifies the header text for the tooltip.
- [Text](#) – Specifies the text for the tooltip.

ASPX-CS

```

@using Syncfusion.Blazor.Charts
<SfChart>
<ChartEvents TooltipRender="TooltipEvent"></ChartEvents>
<ChartPrimaryXAxis
Value="Syncfusion.Blazor.Charts.ValueType.Category"></ChartPrimaryXAxis>
<ChartSeriesCollection>
<ChartSeries DataSource="@Sales" XName="Month" YName="SalesValue"
Type="ChartSeriesType.Column">
</ChartSeries>

```

```

</ChartSeriesCollection>
<ChartTooltipSettings Enable="true"></ChartTooltipSettings>
</SfChart>
@code{
public class SalesInfo
{
public string Month { get; set; }
public double SalesValue { get; set; }
}
public List<SalesInfo> Sales = new List<SalesInfo>
{
new SalesInfo { Month = "Jan", SalesValue = 35 },
new SalesInfo { Month = "Feb", SalesValue = 28 },
new SalesInfo { Month = "Mar", SalesValue = 34 },
new SalesInfo { Month = "Apr", SalesValue = 32 },
new SalesInfo { Month = "May", SalesValue = 40 },
new SalesInfo { Month = "Jun", SalesValue = 32 },
new SalesInfo { Month = "Jul", SalesValue = 35 }
};
public void TooltipEvent(TooltipRenderEventArgs args)
{
// Here you can customize your code
}
}

```

SharedTooltipRender

[SharedTooltipRender](#) event triggers, before the sharedtooltip for series is rendered.

Arguments

Below property available in the the [SharedTooltipRenderEventArgs](#).

- [HeaderText](#) – Specifies the header text for the tooltip.
- [Text](#) – Specifies the text for the tooltip.

ASPX-CS

```

@using Syncfusion.Blazor.Charts
<SfChart>
<ChartEvents SharedTooltipRender="SharedTooltipEvent"></ChartEvents>
<ChartPrimaryXAxis
ValueType="Syncfusion.Blazor.Charts.ValueType.Category"></ChartPrimaryXAxis>
<ChartSeriesCollection>
<ChartSeries DataSource="@Sales" XName="Month" YName="SalesValue"
Type="ChartSeriesType.Column">
</ChartSeries>
<ChartSeries DataSource="@Sales" XName="Month" YName="SalesValue"
Type="ChartSeriesType.Column">
</ChartSeries>
</ChartSeriesCollection>
<ChartTooltipSettings Enable="true" Shared="true"></ChartTooltipSettings>
</SfChart>
@code{
public class SalesInfo
{
public string Month { get; set; }
}
}

```

```

public double SalesValue { get; set; }
}
public List<SalesInfo> Sales = new List<SalesInfo>
{
    new SalesInfo { Month = "Jan", SalesValue = 35 },
    new SalesInfo { Month = "Feb", SalesValue = 28 },
    new SalesInfo { Month = "Mar", SalesValue = 34 },
    new SalesInfo { Month = "Apr", SalesValue = 32 },
    new SalesInfo { Month = "May", SalesValue = 40 },
    new SalesInfo { Month = "Jun", SalesValue = 32 },
    new SalesInfo { Month = "Jul", SalesValue = 35 }
};
public void SharedTooltipEvent(SharedTooltipRenderEventArgs args)
{
    // Here you can customize your code
}
}

```

Refer to our [Blazor Charts](#) feature tour page for its groundbreaking feature representations and also explore our [Blazor Chart Example](#) to know various chart types and how to represent time-dependent data, showing trends at equal intervals.

How To

<!-- markdownlint-disable MD036 -->

Add or Remove Series in Blazor Charts Component

The chart series can be dynamically added or removed by adding and removing a series to the [ChartSeriesCollection](#). Follow the steps below to dynamically add or remove a series.

Step 1:

Render a series using [ChartSeriesCollection](#) class of the chart.

ASPX-CS

```

<SfChart>
  <ChartSeriesCollection>
    @foreach (SeriesData series in SeriesCollection)
    {
      <ChartSeries XName=@series.XName YName=@series.YName
        DataSource=@series.Data>
      </ChartSeries>
    }
  </ChartSeriesCollection>
</SfChart>

```

Step 2:

Create buttons to call add and remove methods, which will add and remove a series from the chart respectively.

ASPX-CS

```

<SfButton @onclick="AddChartSeries">Add Chart Series</SfButton>
<SfButton @onclick="RemoveChartSeries">Remove Chart Series</SfButton>

```

Step 3:

To add a new series to the chart dynamically use the code below in the **AddChartSeries** method. This code adds a new series data to the series list named **SeriesCollection** mapped to the [ChartSeriesCollection](#).

C#

```
public void AddChartSeries ()
{
    SeriesCollection.Add(new SeriesData
    {
        XName = nameof(LineChartData.XValue),
        YName = nameof(LineChartData.YValue),
        Data = GetChartData()
    });
}
```

Step 4:

To remove a series from the chart dynamically use the code below in the **RemoveChartSeries** method. This code removes a series data from the series list named **SeriesCollection** mapped to the [ChartSeriesCollection](#).

C#

```
public void RemoveChartSeries ()
{
    if (SeriesCollection.Count > 0)
    {
        SeriesCollection.Remove(SeriesCollection[SeriesCollection.Count - 1]);
    }
}
```

Action:

By clicking the **Add Chart Series** button a new series will be added to the chart and similarly by clicking the **Remove Chart Series** button the last series in the chart series collection will be removed from the chart. The complete code snippet for the preceding steps is available below.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
@using Syncfusion.Blazor.Buttons
<SfButton @onclick="AddChartSeries">Add Chart Series</SfButton>
<SfButton @onclick="RemoveChartSeries">Remove Chart Series</SfButton>
<div class="container">
<SfChart>
<ChartSeriesCollection>
@foreach (SeriesData series in SeriesCollection)
{
<ChartSeries XName=@series.XName YName=@series.YName
DataSource=@series.Data>
</ChartSeries>
}
</ChartSeriesCollection>
```

```
</SfChart>
</div>
@code{
List<SeriesData> SeriesCollection;
// Here we have added the chart series by adding series data to the
"SeriesCollection" list.
public void AddChartSeries()
{
SeriesCollection.Add(new SeriesData
{
XName = nameof(LineChartData.XValue),
YName = nameof(LineChartData.YValue),
Data = GetChartData()
});
}
// Here we have removed the chart series by removing the series data in the
"SeriesCollection" list.
public void RemoveChartSeries()
{
if (SeriesCollection.Count > 0)
{
SeriesCollection.Remove(SeriesCollection[SeriesCollection.Count - 1]);
}
}
protected override void OnInitialized()
{
base.OnInitialized();
SeriesCollection = new List<SeriesData>() {
new SeriesData {
XName = nameof(LineChartData.XValue),
YName = nameof(LineChartData.YValue),
Data = GetChartData()
}
};
}
private List<LineChartData> GetChartData()
{
int count = 20;
double value = 0;
List<LineChartData> data = new List<LineChartData>();
Random random = new Random();
for (int i = 0; i < count; i++)
{
if (i % 3 == 0)
{
value = value - (random.Next(0, 100) / 3) * 4;
}
else if (i % 2 == 0)
{
value = value + (random.Next(0, 100) / 3) * 4;
}
data.Add(new LineChartData()
{
XValue = i,
YValue = value
});
}
}
```

```
return data;
}
public class SeriesData
{
    public string XName
    {
        get;
        set;
    }
    public string YName
    {
        get;
        set;
    }
    public List<LineChartData> Data
    {
        get;
        set;
    }
}
public class LineChartData
{
    public double XValue
    {
        get;
        set;
    }
    public double YValue
    {
        get;
        set;
    }
}
```

Refer to our [Blazor Charts](#) feature tour page for its groundbreaking feature representations and also explore our [Blazor Chart Example](#) to know various chart types and how to represent time-dependent data, showing trends at equal intervals.

Lazy Loading in Blazor Charts Component

The lazy loading loads data for the chart on demand. The [OnScrollChanged](#) event will be fired by the chart, allowing us to get the minimum and maximum ranges of the axes and then upload the data to the chart.

ASPX-CS

```
<ChartEvents OnScrollChanged="@ScrollChange"></ChartEvents>
private void ScrollChange(ScrollEventArgs e)
{
    this.dataSource = GetRangeData(Convert.ToInt32(e.CurrentRange.Minimum),
    Convert.ToInt32(e.CurrentRange.Maximum));
    this.StateHasChanged();
}
```

The complete code snippet is available below.

ASPX-CS

```

@using Syncfusion.Blazor.Charts
@using System.Collections.ObjectModel
@if (dataSource != null)
{
    <SfChart Title="Lazy Loading Chart">
    <ChartEvents OnScrollChanged="@ScrollChange"></ChartEvents>
    <ChartPrimaryXAxis>
    <ChartAxisScrollbarSettings Enable="true"
PointsLength="1000"></ChartAxisScrollbarSettings>
    </ChartPrimaryXAxis>
    <ChartSeriesCollection>
    <ChartSeries DataSource="@dataSource" Fill="blue" XName="x" YName="y"
Type="ChartSeriesType.Line">
    </ChartSeries>
    </ChartSeriesCollection>
    </SfChart>
}
else
{
    <p>Chart Loading</p>
}
@code {
int count = 1;
Random random = new Random();
public ObservableCollection < ColumnChartData > dataSource;
protected override void OnInitialized() {
dataSource = this.GetData();
}
public void ScrollChange(ScrollEventArgs e) {
this.dataSource = GetRangeData(Convert.ToInt32(e.CurrentRange.Minimum),
Convert.ToInt32(e.CurrentRange.Maximum));
}
public ObservableCollection < ColumnChartData > GetRangeData(int min, int
max) {
ObservableCollection < ColumnChartData > data = new ObservableCollection <
ColumnChartData > ();
for (; min <= max; min++) {
data.Add(new ColumnChartData {
x = min, y = random.Next(10, 100)
});
}
return data;
}
public ObservableCollection < ColumnChartData > GetData() {
ObservableCollection < ColumnChartData > data = new ObservableCollection <
ColumnChartData > ();
for (; count <= 100; count++) {
data.Add(new ColumnChartData {
x = count++, y = random.Next(10, 100)
});
}
return data;
}
public class ColumnChartData {
public double x {

```



```

get;
set;
}
public double y {
get;
set;
}
}
}

```

Refer to our [Blazor Charts](#) feature tour page for its groundbreaking feature representations and also explore our [Blazor Chart Example](#) to know various chart types and how to represent time-dependent data, showing trends at equal intervals.

Table in Tooltip in Blazor Charts Component

A table type tooltip can be created using the [Template](#) property in [ChartTooltipSettings](#). Follow the steps below to display a table inside the tooltip.

Step 1:

Render a chart with the required series using [ChartSeriesCollection](#).

ASPX-CS

```

<SfChart Title="Weather condition JPN vs DEU">
<ChartPrimaryXAxis ValueType="Syncfusion.Blazor.Charts.ValueType.Category"
/>
<ChartSeriesCollection>
<ChartSeries DataSource="@WeatherReports" XName="X" YName="Y"
Type="ChartSeriesType.Column" />
</ChartSeriesCollection>
</SfChart>

```

Step 2:

The tooltip can be enabled using the [Enable](#) property as **true** in [ChartTooltipSettings](#).

ASPX-CS

```

...
<ChartTooltipSettings Enable="true">
...

```

Step 3:

Construct a HTML table as per the requirement and place the implicit named parameter context to access the aggregate values within the template. To retrieve aggregate values inside the template, type cast the context as [ChartTooltipInfo](#).

ASPX-CS

```

...
<ChartTooltipSettings Enable="true">
<Template>
@{
var data = context as ChartTooltipInfo;

```

```

<table border="5" bgcolor="lightblue">
<tr style="border: 1px solid black">
<td style="border: 1px solid black">Month: </td>
<td style="border: 1px solid black">@data.X</td>
</tr>
<tr style="border: 1px solid black">
<td style="border: 1px solid black">Value: </td>
<td style="border: 1px solid black">@data.Y</td>
</tr>
</table>
}
</Template>
</ChartTooltipSettings>
...

```

Action

When the mouse is moved over the chart series points, the tooltip is displayed in table format. The complete code snippet for the preceding steps is available below.

ASPX-CS

```

@using Syncfusion.Blazor.Charts
<SfChart Title="Weather condition JPN vs DEU">
<ChartPrimaryXAxis ValueTypes="Syncfusion.Blazor.Charts.ValueType.Category"
/>
<ChartSeriesCollection>
<ChartSeries DataSource="@WeatherReports" XName="X" YName="Y"
Type="ChartSeriesType.Column" />
</ChartSeriesCollection>
<ChartTooltipSettings Enable="true">
<Template>
@{
var data = context as ChartTooltipInfo;
<table border="2" bgcolor="lightblue" cellpadding="5">
<tr style="border: .1px solid black">
<td style="border: 1px solid black">Month: </td>
<td style="border: 1px solid black"> @data.X</td>
</tr>
<tr style="border: .1px solid black">
<td style="border: 1px solid black">Value: </td>
<td style="border: 1px solid black"> @data.Y</td>
</tr>
</table>
}
</Template>
</ChartTooltipSettings>
</SfChart>
@code{
public class ChartData
{
public string X { get; set; }
public double Y { get; set; }
}
public List<ChartData> WeatherReports = new List<ChartData>
{

```

```

new ChartData{ X= "Jan", Y= 15 },
new ChartData{ X= "Feb", Y= 20 },
new ChartData{ X= "Mar", Y= 35 },
new ChartData{ X= "Apr", Y= 40 },
new ChartData{ X= "May", Y= 80 },
new ChartData{ X= "Jun", Y= 70 },
new ChartData{ X= "Jul", Y= 65 },
new ChartData{ X= "Aug", Y= 55 },
new ChartData{ X= "Sep", Y= 50 },
new ChartData{ X= "Oct", Y= 30 },
new ChartData{ X= "Nov", Y= 35 },
new ChartData{ X= "Dec", Y= 35 }
};
}

```

Refer to our [Blazor Charts](#) feature tour page for its groundbreaking feature representations and also explore our [Blazor Chart Example](#) to know various chart types and how to represent time-dependent data, showing trends at equal intervals.

Visualize grid data in chart in Blazor Charts Component

Use the chart's [OnSelectionChanged](#) event to get the list of selected data from the chart.

ASPX-CS

```

<ChartEvents OnSelectionChanged="@ShowSelectedData"></ChartEvents>
public void ShowSelectedData(IDragCompleteEventArgs args)
{
    object data = args.SelectedDataValues;
    List<List<SelectionData>> data1 =
    Newtonsoft.Json.JsonConvert.DeserializeObject<List<List<SelectionData>>>(dat
a.ToString());
    this.SelectedData = new List<SelectionData>();
    this.SelectedData2 = new List<SelectionData>();
    for (int i = 0; i < data1[0].Count; i++)
    {
        this.SelectedData.Add(new SelectionData { x = data1[0][i].x, y =
data1[0][i].y });
    }
    for (int i = 0; i < data1[1].Count; i++)
    {
        this.SelectedData2.Add(new SelectionData { x = data1[1][i].x, y =
data1[1][i].y });
    }
    this.StateHasChanged();
}

```

By using the grid's [DataSource](#) property, chart's selected data can be listed in the grid.

ASPX-CS

```

<SfGrid DataSource="@SelectedData2">
<GridColumns>
<GridColumn Field=@nameof(SelectionData.x) HeaderText="X Value"
TextAlign="TextAlign.Right" Width="50%"></GridColumn>
<GridColumn Field=@nameof(SelectionData.y) HeaderText="Y value"
Width="50%"></GridColumn>

```

```
</GridColumns>
</SfGrid>
```

The complete code snippet is available below.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
@using Syncfusion.Blazor.Grids
@using System.Collections.ObjectModel

<div class="row">
<div class="col-md-8">
<SfChart Title="Profit Comparison of A and B"
SelectionMode="Syncfusion.Blazor.Charts.SelectionMode.DragXY">
<ChartEvents OnSelectionChanged="@ShowSelectedData"></ChartEvents>
<ChartPrimaryXAxis Minimum="1970" Maximum="2016">
<ChartAxisMajorGridLines Width="0"></ChartAxisMajorGridLines>
</ChartPrimaryXAxis>
<ChartPrimaryYAxis Minimum="0" Maximum="100" Interval="25" Title="Sales"
LabelFormat="{value}%">
<ChartAxisMajorGridLines Width="0"></ChartAxisMajorGridLines>
</ChartPrimaryYAxis>
<ChartSeriesCollection>
<ChartSeries DataSource="@dataSource" Name="Product A" XName="x" Opacity="1"
YName="y1" Type="ChartSeriesType.Scatter">
<ChartMarker Height="10" Width="10"
Shape="ChartShape.Triangle"></ChartMarker>
</ChartSeries>
<ChartSeries DataSource="@dataSource" Name="Product B" XName="x" Opacity="1"
YName="y2" Type="ChartSeriesType.Scatter">
<ChartMarker Height="10" Width="10"
Shape="ChartShape.Pentagon"></ChartMarker>
</ChartSeries>
</ChartSeriesCollection>
<ChartArea>
<ChartAreaBorder Width="0"></ChartAreaBorder>
</ChartArea>
<ChartLegendSettings Visible="true"
ToggleVisibility="false"></ChartLegendSettings>
</SfChart>
</div>
<div class="col-md-2">
Series 1
<SfGrid DataSource="@SelectedData">
<GridColumns>
<GridColumn Field=@nameof(SelectionData.x) HeaderText="X Value"
TextAlign="TextAlign.Right" Width="50%"></GridColumn>
<GridColumn Field=@nameof(SelectionData.y) HeaderText="Y value"
Width="50%"></GridColumn>
</GridColumns>
</SfGrid>
</div>
<div class="col-md-2">
Series 2
<SfGrid DataSource="@SelectedData2">
<GridColumns>
```

```

<GridColumn Field=@nameof(SelectionData.x) HeaderText="X Value"
TextAlign="TextAlign.Right" Width="50%"></GridColumn>
<GridColumn Field=@nameof(SelectionData.y) HeaderText="Y value"
Width="50%"></GridColumn>
</GridColumn>
</SfGrid>
</div>
</div>
@code{
public class RangeSelectionData
{
public double x { get; set; }
public double y1 { get; set; }
public double y2 { get; set; }
}
public class SelectionData
{
public double x { get; set; }
public double y { get; set; }
}
private Random rnd = new Random();
public List<RangeSelectionData> dataSource = new List<RangeSelectionData>();
public ObservableCollection<SelectionData> SelectedData = new
ObservableCollection<SelectionData>();
public ObservableCollection<SelectionData> SelectedData2 = new
ObservableCollection<SelectionData>();
protected override void OnInitialized()
{
for (int i = 0; i < 48; i++)
{
dataSource.Add(new RangeSelectionData { x = 1971 + i, y1 = rnd.Next(10,
100), y2 = rnd.Next(10, 100) });
}
}
public void ShowSelectedData(SelectionCompleteEventArgs Args)
{
object data = Args.SelectedDataValues;
List<List<SelectionData>> data1 =
Newtonsoft.Json.JsonConvert.DeserializeObject<List<List<SelectionData>>>(dat
a.ToString());
this.SelectedData = new ObservableCollection<SelectionData>();
this.SelectedData2 = new ObservableCollection<SelectionData>();
for (int i = 0; i < data1[0].Count; i++)
{
this.SelectedData.Add(new SelectionData { x = data1[0][i].x, y =
data1[0][i].y });
}
for (int i = 0; i < data1[1].Count; i++)
{
this.SelectedData2.Add(new SelectionData { x = data1[1][i].x, y =
data1[1][i].y });
}
}
}

```

Refer to our [Blazor Charts](#) feature tour page for its groundbreaking feature representations and also explore our [Blazor Chart Example](#) to know various chart types and how to represent time-dependent data, showing trends at equal intervals.

<!-- markdownlint-disable MD036 -->

Threshold in Chart in Blazor Charts Component

The threshold level can be indicated in the chart using the [ChartStripline](#). Follow the steps below to add a threshold to the chart.

Step 1:

Render a chart with the required series using [ChartSeriesCollection](#).

ASPX-CS

```
<SfChart Title="Weather condition JPN vs DEU">
<ChartPrimaryXAxis ValueType="Syncfusion.Blazor.Charts.ValueType.Category"
/>
<ChartSeriesCollection>
<ChartSeries DataSource="@WeatherReports" XName="X" YName="Y"
Type="ChartSeriesType.Column" />
</ChartSeriesCollection>
</SfChart>
```

Step 2:

Since this threshold must be added to the measure axis, the [ChartStripline](#) setting will be set to [ChartPrimaryYAxis](#).

Using the [ChartStriplines](#) collection property of the axis, multiple thresholds can be added to a chart.

ASPX-CS

```
...
<ChartPrimaryYAxis>
<ChartStriplines>
<ChartStripline Start="30" Size="1" ></ChartStripline>
</ChartStriplines>
</ChartPrimaryYAxis>
...
```

Step 3:

To represent the severity of the threshold, a color can be set to the stripline using the [Color](#) property of the [ChartStripline](#).

ASPX-CS

```
...
<ChartPrimaryYAxis>
<ChartStriplines>
<ChartStripline Start="30" Size="1" Color="red" ></ChartStripline>
</ChartStriplines>
</ChartPrimaryYAxis>
...
```

Step 4:

The stripline's order, which determines whether it is rendered behind or above the series elements, can be customized by [ZIndex](#) property of the [ChartStripline](#).

ASPX-CS

```
...
<ChartPrimaryYAxis>
<ChartStriplines>
<ChartStripline Start="30" Size="1" Color="red"
ZIndex="ZIndex.Over"></ChartStripline>
</ChartStriplines>
</ChartPrimaryYAxis>
...
```

The complete code snippet for the preceding steps is available below.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart Title="Weather condition JPN vs DEU">
<ChartPrimaryXAxis ValueType="Syncfusion.Blazor.Charts.ValueType.Category"
/>
<ChartPrimaryYAxis>
<ChartStriplines>
<ChartStripline Start="30" Size="1" ZIndex="ZIndex.Over"
Color="red"></ChartStripline>
</ChartStriplines>
</ChartPrimaryYAxis>
<ChartSeriesCollection>
<ChartSeries DataSource="@WeatherReports" XName="X" YName="Y"
Type="ChartSeriesType.Column" />
</ChartSeriesCollection>
</SfChart>
@code{
public class ChartData
{
public string X { get; set; }
public double Y { get; set; }
}
public List<ChartData> WeatherReports = new List<ChartData>
{
new ChartData{ X= "Jan", Y= 15 },
new ChartData{ X= "Feb", Y= 20 },
new ChartData{ X= "Mar", Y= 35 },
new ChartData{ X= "Apr", Y= 40 },
new ChartData{ X= "May", Y= 80 },
new ChartData{ X= "Jun", Y= 70 },
new ChartData{ X= "Jul", Y= 65 },
new ChartData{ X= "Aug", Y= 55 },
new ChartData{ X= "Sep", Y= 50 },
new ChartData{ X= "Oct", Y= 30 },
new ChartData{ X= "Nov", Y= 35 },
new ChartData{ X= "Dec", Y= 35 }
};
}
```

Refer to our [Blazor Charts](#) feature tour page for its groundbreaking feature representations and also explore our [Blazor Chart Example](#) to know various chart types and how to represent time-dependent data, showing trends at equal intervals.

```
<!-- markdownlint-disable MD036 -->
```

Live Chart in Blazor Charts Component

Live update in a chart can be achieved using the timer to update the datasource with real-time data and refresh the chart. Follow the steps below to update a chart with real-time data.

Step 1:

Render a chart with the required series using [ChartSeriesCollection](#).

ASPX-CS

```
<SfChart @ref="liveChart" Title="CPU Usage" Width="100%" >
  <ChartPrimaryXAxis ValueType="Syncfusion.Blazor.Charts.ValueType.DateTime"
    Title="Time (s)">
    <ChartAxisMajorGridLines Width="0"></ChartAxisMajorGridLines>
  </ChartPrimaryXAxis>
  <ChartPrimaryYAxis Title="Usage" Minimum="0" Interval="20" Maximum="100" >
    <ChartAxisLineStyle Width="0" Color="transparent"></ChartAxisLineStyle>
  </ChartPrimaryYAxis>
  <ChartSeriesCollection>
    <ChartSeries Type="ChartSeriesType.Line" Width="2" DataSource="@DataPoints"
      XName="Time" YName="CPU_Usage">
      <ChartSeriesAnimation Enable="false"></ChartSeriesAnimation>
    </ChartSeries>
  </ChartSeriesCollection>
</SfChart>
```

Step 2:

Labels of the axes can be formatted based on our need using the [LabelFormat](#) property of the [ChartAxis](#). Since the chart will be updated in seconds, the [LabelFormat](#) has been set as **mm:ss** for the [ChartPrimaryXAxis](#) to display minutes and second in the axis labels. Similarly [ChartPrimaryYAxis](#) labels can also be formatted as shown below using its [LabelFormat](#) property.

ASPX-CS

```
<SfChart @ref="liveChart" Title="CPU Usage" Width="100%" >
  <ChartPrimaryXAxis ValueType="Syncfusion.Blazor.Charts.ValueType.DateTime"
    LabelFormat="mm:ss" Title="Time (s)">
    <ChartAxisMajorGridLines Width="0"></ChartAxisMajorGridLines>
  </ChartPrimaryXAxis>
  <ChartPrimaryYAxis Title="Usage" Minimum="0" Interval="20" Maximum="100"
    LabelFormat="{value}%">
    <ChartAxisLineStyle Width="0" Color="transparent"></ChartAxisLineStyle>
  </ChartPrimaryYAxis>
  <ChartSeriesCollection>
    <ChartSeries Type="ChartSeriesType.Line" Width="2" DataSource="@DataPoints"
      XName="Time" YName="CPU_Usage">
      <ChartSeriesAnimation Enable="false"></ChartSeriesAnimation>
    </ChartSeries>
  </ChartSeriesCollection>
</SfChart>
```



```
</ChartSeriesCollection>
</SfChart>
```

Step 3:

Add a timer to automatically update the chart every 500 milliseconds by calling the **AddData** function, which removes the first data from the data points collection, which is of the **ObservableCollection** type, and adds a new data to it. Since this **ObservableCollection** type is used as the data source, it is triggered whenever there is a data update within it.

ASPX-CS

```
protected override void OnInitialized()
{
    // Provide the chart with initial data during page load.
    DataPoints = new ObservableCollection<ChartDataPoint>();
    for (int i = 0; i < dataLength; i++)
        DataPoints.Add(new ChartDataPoint
        {
            Time = DateTime.Now.AddSeconds(i + 10),
            CPU_Usage = randomNum.Next(30, 80)
        });
    // Starting live update in the chart.
    timer = new Timer(500);
    timer.Elapsed += AddData;
    timer.AutoReset = true;
    timer.Enabled = true;
}

private void AddData(Object source, ElapsedEventArgs e)
{
    dataLength++;
    DataPoints.RemoveAt(0);
    DataPoints.Add(new ChartDataPoint
    {
        Time = DateTime.Now.AddSeconds(dataLength + 10),
        CPU_Usage = randomNum.Next(30, 80)
    });
}
```

The complete code snippet for the preceding steps is available below.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
@using System.Collections.ObjectModel
@using System.Timers

<div class="control-section" align="center">
    <SfChart @ref="liveChart" Title="CPU Usage" Width="100%" >
        <ChartPrimaryXAxis Value="Syncfusion.Blazor.Charts.ValueType.DateTime"
            LabelFormat="mm:ss" Title="Time (s)">
            <ChartAxisMajorGridLines Width="0"></ChartAxisMajorGridLines>
        </ChartPrimaryXAxis>
        <ChartPrimaryYAxis Title="Usage" Minimum="0" Interval="20" Maximum="100"
            LabelFormat="{value}%">
            <ChartAxisLineStyle Width="0" Color="transparent"></ChartAxisLineStyle>
        </ChartPrimaryYAxis>
```

```

<ChartSeriesCollection>
<ChartSeries Type="ChartSeriesType.Line" Width="2" DataSource="@DataPoints"
XName="@nameof (ChartDataPoint.Time)"
YName="@nameof (ChartDataPoint.CPU_Usage)">
<ChartSeriesAnimation Enable="false"></ChartSeriesAnimation>
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
</div>
@code{
private static Timer timer;
private SfChart liveChart;
private double dataLength = 100;
private Random randomNum = new Random();
public ObservableCollection<ChartDataPoint> DataPoints;
protected override void OnInitialized()
{
// Provide the chart with initial data during page load.
DataPoints = new ObservableCollection<ChartDataPoint>();
for (int i = 0; i < dataLength; i++)
DataPoints.Add(new ChartDataPoint
{
Time = DateTime.Now.AddSeconds(i + 10),
CPU_Usage = randomNum.Next(30, 80)
});
// Starting live update in the chart.
timer = new Timer(500);
timer.Elapsed += AddData;
timer.AutoReset = true;
timer.Enabled = true;
}
private void AddData(Object source, ElapsedEventArgs e)
{
if (liveChart == null)
return;
dataLength++;
DataPoints.RemoveAt(0);
DataPoints.Add(new ChartDataPoint
{
Time = DateTime.Now.AddSeconds(dataLength + 10),
CPU_Usage = randomNum.Next(30, 80)
});
}
public class ChartDataPoint
{
public DateTime Time { get; set; }
public double CPU_Usage { get; set; }
}
}

```

Refer to our [Blazor Charts](#) feature tour page for its groundbreaking feature representations and also explore our [Blazor Chart Example](#) to know various chart types and how to represent time-dependent data, showing trends at equal intervals.

<!-- markdownlint-disable MD036 -->

Hiding Axis in Blazor Charts Component

The axis associated with the series can be hidden by toggling the legend item of the relevant series. Follow the steps below to hide the measure axis associated with the series.

Step 1:

Render a chart with three series and three axis using [ChartSeriesCollection](#) and [ChartAxes](#) properties of the chart. Map each series to a measure axis using [YAxisName](#) property of the [ChartSeries](#).

ASPX-CS

```
<SfChart>
<ChartAxes>
<ChartAxis Name="yAxis1" />
<ChartAxis OpposedPosition="true" Name="yAxis2" />
<ChartAxis OpposedPosition="true" Name="yAxis3" />
</ChartAxes>
<ChartSeriesCollection>
<ChartSeries DataSource="@DataPoints" XName="XValue" YName="YValue1"
Name="England" YAxisName="yAxis1" Type="ChartSeriesType.Column">
</ChartSeries>
<ChartSeries DataSource="@DataPoints" XName="XValue" YName="YValue2"
Name="US" Fill="green" YAxisName="yAxis2" Type="ChartSeriesType.Column">
</ChartSeries>
<ChartSeries DataSource="@DataPoints" XName="XValue" YName="YValue3"
Name="West Indies" Fill="#b22222" YAxisName="yAxis3"
Type="ChartSeriesType.Column">
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
```

Step 2:

In the [ChartLegendSettings](#), configure the [Visible](#) property to display the legend and the [ToggleVisibility](#) property to enable legend item toggling to control the visibility of the associated series.

ASPX-CS

```
<SfChart>
...
<ChartLegendSettings Visible="true" ToggleVisibility="true" />
...
</SfChart>
```

Action:

By clicking the legend items, one can now toggle the visibility of the associated series. Once the series has been removed, the associated axis will be removed from the chart. If the legend item is toggled again to show the series, the associated axis with the series will be added to the chart. The complete code snippet for the preceding steps is available below.

ASPX-CS

```
@using Syncfusion.Blazor.Charts
<SfChart>
<ChartAxes>
```

```

<ChartAxis Name="yAxis1" />
<ChartAxis OpposedPosition="true" Name="yAxis2" />
<ChartAxis OpposedPosition="true" Name="yAxis3" />
</ChartAxes>
<ChartSeriesCollection>
<ChartSeries DataSource="@DataPoints" XName="XValue" YName="YValue1"
Name="England" YAxisName="yAxis1" Type="ChartSeriesType.Column">
</ChartSeries>
<ChartSeries DataSource="@DataPoints" XName="XValue" YName="YValue2"
Name="US" Fill="green" YAxisName="yAxis2" Type="ChartSeriesType.Column">
</ChartSeries>
<ChartSeries DataSource="@DataPoints" XName="XValue" YName="YValue3"
Name="West Indies" Fill="#b22222" YAxisName="yAxis3"
Type="ChartSeriesType.Column">
</ChartSeries>
</ChartSeriesCollection>
<ChartLegendSettings Visible="true" ToggleVisibility="true" />
</SfChart>
@code{
public List<ChartData> DataPoints = new List<ChartData>
{
new ChartData { XValue = 16, YValue1 = 2, YValue2 = 12, YValue3 = 7 },
new ChartData { XValue = 17, YValue1 = 14, YValue2 = 10, YValue3 = 7 },
new ChartData { XValue = 18, YValue1 = 7, YValue2 = 17, YValue3 = 11 },
new ChartData { XValue = 19, YValue1 = 7, YValue2 = 20, YValue3 = 8 },
new ChartData { XValue = 20, YValue1 = 10, YValue2 = 16, YValue3 = 24 },
};
public class ChartData
{
public double XValue { get; set; }
public double YValue1 { get; set; }
public double YValue2 { get; set; }
public double YValue3 { get; set; }
}
}

```

Refer to our [Blazor Charts](#) feature tour page for its groundbreaking feature representations and also explore our [Blazor Chart Example](#) to know various chart types and how to represent time-dependent data, showing trends at equal intervals.

<!-- markdownlint-disable MD036 -->

Convert millisecond to date time in Blazor Charts Component

Chart converts the datetime to milliseconds to calculate the bounds, so all events for datetime axis returns the value in milliseconds. For example, after zoom completion, the ranges for axis will be in the milliseconds, by using the [OnZoomEnd](#) event, you can convert millisecond value to date time format.

To convert millisecond value to date time format, follow the given steps:

Step 1:

Using `OnZoomEnd` event, you can get the axis range in milliseconds, by using below code, you can get the equivalent date value.

ASPX-CS

```
<ChartEvents OnZoomEnd="RangeSelectionCompleted"></ChartEvents>
public void RangeSelectionCompleted(ZoomingEventArgs args)
{
    var zoomData = args?.AxisCollection?.FirstOrDefault();
    Console.WriteLine(new DateTime(1970, 1,
    1).AddMilliseconds(zoomData.AxisRange.Min));
    Console.WriteLine(new DateTime(1970, 1,
    1).AddMilliseconds(zoomData.AxisRange.Max));
}
```

CheckBox

<!-- markdownlint-disable MD024 -->

Getting Started with Blazor CheckBox Component

This section briefly explains about how to include [Checkbox](#) Component in your Blazor server-side application. You can refer [Getting Started with Syncfusion Blazor for Server-side in Visual Studio](#) page for the introduction and configuring the common specifications.

To get started quickly with the CheckBox Component using Blazor, you can check out this video:

{% youtube

"youtube:https://www.youtube.com/watch?v=a6HR1QGAvLo"%}

Importing Syncfusion Blazor component in the application

1. Install the [Syncfusion.Blazor](#) NuGet package to the application by using the **NuGet Package Manager**.
2. You can add the client-side style resources through [CDN](#) or from [NuGet](#) package in the element of the `~/Pages/_Host.cshtml` page.

HTML

```
<head>
<link href="_content/Syncfusion.Blazor/styles/bootstrap4.css"
rel="stylesheet" />
@*<link href="https://cdn.syncfusion.com/blazor/{ site.blazorversion
}/styles/bootstrap4.css" rel="stylesheet" />*@
</head>
```

For Internet Explorer 11 kindly refer the polyfills. Refer the [documentation](#) for more information.

HTML

```
<head>
<link href="_content/Syncfusion.Blazor/styles/bootstrap4.css"
rel="stylesheet" />
<script
src="https://github.com/Daddoon/Blazor.Polyfill/releases/download/3.0.1/blaz
or.polyfill.min.js"></script>
</head>
```

Adding component package to the application

Open `/_Imports.razor` file and import the **Syncfusion.Blazor.Buttons** package.

CSHARP

```
@using Syncfusion.Blazor.Buttons
```

Add SyncfusionBlazor service in Startup.cs

Open the **Startup.cs** file and add services required by Syncfusion components.

Add **services.AddSyncfusionBlazor()** method in the `ConfigureServices` function as follows.

CSHARP

```
namespace BlazorApplication
{
    public class Startup
    {
        ....
        ....
        public void ConfigureServices(IServiceCollection services)
        {
            ....
            ....
            services.AddSyncfusionBlazor();
        }
    }
}
```

To enable custom client side resource loading from CRG or CDN, you need to disable resource loading by **AddSyncfusionBlazor(true)** and load the scripts in the HEAD element of the `~/Pages/_Host.cshtml` page.

HTML

```
<head>
<environment include="Development">
<script src="https://cdn.syncfusion.com/blazor/{{ site.blazorversion
}}/syncfusion-blazor.min.js">
</script>
</environment>
</head>
```

Adding component package to the application

Open `/_Imports.razor` file and import the **Syncfusion.Blazor.Buttons** packages. Otherwise, import these packages in the individual `razor` pages.

ASPX-CS

```
@using Syncfusion.Blazor
@using Syncfusion.Blazor.Buttons
```

Adding Checkbox component to the application

Now, add the Syncfusion Blazor Checkbox component in `razor` page in the `Pages` folder. For example, the Checkbox component is added in the `~/Pages/Index.razor` page.

ASPX-CS

```
<SfCheckBox Label="Default" @bind-Checked="isChecked"></SfCheckBox>
@code
{
    private bool isChecked = true;
}
```

Run the application

After successful compilation of your application, simply press F5 to run the application. The Blazor Checkbox component will render in the web browser as shown below



See Also

- [Getting Started with Syncfusion Blazor for Client-Side in .NET Core CLI](#)
- [Getting Started with Syncfusion Blazor for Server-Side in Visual Studio](#)
- [Getting Started with Syncfusion Blazor for Server-Side in .NET Core CLI](#)

Native Events in Blazor CheckBox Component

You can define the native event using an **event** attribute in component. The value of attribute is treated as an event handler. The event specific data will be available in event arguments.

The different event argument types for each event are,

- Focus Events - UIFocusEventArgs
- Mouse Events - UIMouseEventArgs
- Keyboard Events - UIKeyboardEventArgs
- Touch Events – UITouchEventArgs

List of Native events supported

We have provided the following native event support to the Checkbox component:

```
| List of Native events | | | | |
|---|---|---|---|---|
| onchange | oninput | onblur | onfocusout | onfocusin |
| onfocus | onclick | onkeydown | onkeyup | onkeypress |
```

How to bind onchange event to Checkbox

The **onchange** attribute is used to bind the onchange event for Checkbox. Here, we have explained about the sample code snippets of Checkbox.

ASPX-CS

```
@using Syncfusion.Blazor.Buttons
<SfCheckBox @bind-Checked="isChecked" Label="Change"
@onchange="onChange"></SfCheckBox>
@code {
```

```
private bool isChecked = true;
private void onChange(Microsoft.AspNetCore.Components.ChangeEventArgs args)
{
    //onChange Event triggered
}
```

Label and Size in Blazor CheckBox Component

This section explains the different sizes and labels.

Label

The Checkbox caption can be defined by using the [Label](#) property. This reduces the manual addition of label for Checkbox. You can customize the label position before or after the Checkbox through the [LabelPosition](#) property.

ASPX-CS

```
@using Syncfusion.Blazor.Buttons
<SfCheckBox Label="Left Side Label" LabelPosition="LabelPosition.Before"
@bind-Checked="isLeftChecked"></SfCheckBox><br />
<SfCheckBox Label="Right Side Label" LabelPosition="LabelPosition.After"
@bind-Checked="isRightChecked"></SfCheckBox>
@code {
    private bool isLeftChecked = true;
    private bool isRightChecked = true;
}
```

Output will be as follows

Left Side Label ☒

☒ Right Side Label

Size

The different Checkbox sizes available are default and small. To reduce the size of default Checkbox to small, set the [CssClass](#) property to `e-small`.

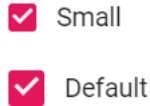
ASPX-CS

```
@using Syncfusion.Blazor.Buttons
<SfCheckBox @bind-Checked="isSmallChecked" Label="Small" CssClass="e-small"></SfCheckBox><br />
<SfCheckBox @bind-Checked="isDefaultChecked" Label="Default"></SfCheckBox>
@code {
    private bool isSmallChecked = true;
    private bool isDefaultChecked = true;
}
<style>
.e-checkbox-wrapper {
    margin-top: 18px;
}
```



```
</style>
```

Output will be as follows



See Also

- [Checkbox customization](#)

Accessibility in Blazor CheckBox Component

The web accessibility makes web content and web applications more accessible for people with disabilities. It especially helps in dynamic content change and development of advanced user interface controls with AJAX, HTML, JavaScript, and related technologies.

Checkbox provides built-in compliance with WAI-ARIA specifications. WAI-ARIA support is achieved through the attributes like `aria-checked` and `aria-disabled`. It helps the people with disabilities by providing information about the widget for assistive technology in the screen readers. Checkbox component contains the `checkbox` role.

| Properties | Functionality |

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

| role | Indicates the type of input element. |

| `aria-checked` | Indicates whether the input is checked, unchecked, or represents mixture of checked and unchecked values. |

| `aria-disabled` | Indicates that the element is perceivable but disabled, so it is not editable or otherwise operable. |

Keyboard interaction

<!-- markdownlint-disable MD033 -->

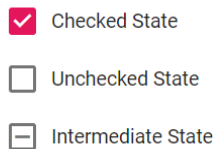
Keyboard shortcuts	Actions
Space	When the Checkbox has focus, pressing the Space key changes the state of the Checkbox.

ASPX-CS

```
@using Syncfusion.Blazor.Buttons
<SfCheckBox @bind-Checked="isChecked" Label="Checked State"></SfCheckBox><br />
```

```
<SfCheckBox @bind-Checked="isChecked" Label="Unchecked
State"></SfCheckBox><br />
<SfCheckBox @bind-Checked="isMediateChecked" Indeterminate="true"
Label="Intermediate State"></SfCheckBox>
@code {
private bool isChecked = true;
private bool isChecked = false;
private bool isMediateChecked = false;
}
```

Output will be as follows



Styles and Appearances in Blazor CheckBox Component

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

CSS Class	Purpose of Class
----- -----	
.e-checkbox-wrapper .e-frame	To customize the checkbox frame.
.e-checkbox-wrapper:hover .e-frame	To customize the checkbox frame on hover.
.e-checkbox-wrapper .e-label	To customize the checkbox label.
.e-checkbox-wrapper:hover .e-label	To customize the checkbox label on hover.
.e-checkbox-wrapper .e-frame.e-check	To customize the checked checkbox.
.e-checkbox-wrapper:hover .e-frame.e-check	To customize the checked checkbox when hover.

How To

Customized Checkbox in Blazor CheckBox Component

Customize Checkbox Appearance

You can customize the appearance of the Checkbox component using the CSS rules. Define own CSS rules according to your requirement and assign the class name to the

[CssClass](#) property.

The background and border color of the Checkbox is customized through the custom classes to create primary, success, warning, and danger info type of checkbox.

ASPX-CS

```
@using Syncfusion.Blazor.Buttons
```

```

<SfCheckBox @bind-Checked="isPrimaryChecked" Label="Primary" CssClass="e-
primary"></SfCheckBox><br />
<SfCheckBox @bind-Checked="isSuccessChecked" Label="Success" CssClass="e-
success"></SfCheckBox><br />
<SfCheckBox @bind-Checked="isInfoChecked" Label="Info" CssClass="e-
info"></SfCheckBox><br />
<SfCheckBox @bind-Checked="isWarningChecked" Label="Warning" CssClass="e-
warning"></SfCheckBox><br />
<SfCheckBox @bind-Checked="isDangerChecked" Label="Danger" CssClass="e-
danger"></SfCheckBox>
@code {
private bool isPrimaryChecked = true;
private bool isSuccessChecked = true;
private bool isInfoChecked = true;
private bool isWarningChecked = true;
private bool isDangerChecked = true;
}
<style>
.e-checkbox-wrapper.e-primary:hover .e-frame.e-check { /* csslint allow:
adjoining-classes */
background-color: #e03872;
}
.e-checkbox-wrapper.e-success .e-frame.e-check,
.e-checkbox-wrapper.e-success .e-checkbox:focus + .e-frame.e-check { /*
csslint allow: adjoining-classes */
background-color: #689f38;
}
.e-checkbox-wrapper.e-success:hover .e-frame.e-check { /* csslint allow:
adjoining-classes */
background-color: #449d44;
}
.e-checkbox-wrapper.e-info .e-frame.e-check,
.e-checkbox-wrapper.e-info .e-checkbox:focus + .e-frame.e-check { /* csslint
allow: adjoining-classes */
background-color: #2196f3;
}
.e-checkbox-wrapper.e-info:hover .e-frame.e-check { /* csslint allow:
adjoining-classes */
background-color: #0b7dda;
}
.e-checkbox-wrapper.e-warning .e-frame.e-check,
.e-checkbox-wrapper.e-warning .e-checkbox:focus + .e-frame.e-check { /*
csslint allow: adjoining-classes */
background-color: #ef6c00;
}
.e-checkbox-wrapper.e-warning:hover .e-frame.e-check { /* csslint allow:
adjoining-classes */
background-color: #cc5c00;
}
.e-checkbox-wrapper.e-danger .e-frame.e-check,
.e-checkbox-wrapper.e-danger .e-checkbox:focus + .e-frame.e-check { /*
csslint allow: adjoining-classes */
background-color: #d84315;
}
.e-checkbox-wrapper.e-danger:hover .e-frame.e-check { /* csslint allow:
adjoining-classes */
background-color: #ba3912;
}

```

```
}  
</style>
```

Output will be as follows

- ☒ Primary
- ☒ Success
- ☒ Info
- ☒ Warning
- ☒ Danger

Customize width and height

The height and width of the Checkbox component can be customized by setting **height** and **width** properties in **styles**

The following section explains about how to customize the height and width of the Checkbox component.

ASPX-CS

```
@using Syncfusion.Blazor.Buttons  
<SfCheckBox CssClass="e-customsize" Label="Default" @bind-  
Checked="isChecked"></SfCheckBox>  
@code {  
private bool isChecked = true;  
}  
<style>  
.e-customsize.e-checkbox-wrapper .e-frame {  
height: 30px;  
width: 30px;  
padding: 8px 0;  
}  
.e-customsize.e-checkbox-wrapper .e-check {  
font-size: 20px;  
}  
.e-customsize.e-checkbox-wrapper .e-ripple-container {  
height: 52px;  
top: -11px;  
width: 47px;  
}  
.e-customsize.e-checkbox-wrapper .e-label {  
line-height: 30px;  
font-size: 20px;  
}  
</style>
```

Output will be as follows



Default

Custom Frame

Checkbox frame can be customized as per the requirement by adding CSS rules.

In the following example, to-do list is displayed with round checkbox by changing `border-radius` as 100% by adding `e-custom` class.

ASPX-CS

```
@using Syncfusion.Blazor.Buttons
<SfCheckBox Label="Buy Groceries" @bind-Checked="isChecked" CssClass="e-custom"></SfCheckBox><br />
<SfCheckBox Label="Pay Rent" @bind-Checked="isRentChecked" CssClass="e-custom"></SfCheckBox><br />
<SfCheckBox Label="Make Dinner" @bind-Checked="isDinnerChecked" CssClass="e-custom"></SfCheckBox><br />
<SfCheckBox Label="Finish To-do List Article" @bind-Checked="isArticleChecked" CssClass="e-custom"></SfCheckBox>
@code {
private bool isChecked = true;
private bool isRentChecked = true;
private bool isDinnerChecked = false;
private bool isArticleChecked = false;
}
<style>
.e-custom .e-frame {
border-radius: 100%;
}
.e-checkicon.e-checkbox-wrapper .e-frame.e-check::before {
content: '\e77d';
}
.e-checkicon.e-checkbox-wrapper .e-check {
font-size: 8.5px;
}
.e-checkicon.e-checkbox-wrapper .e-frame.e-check {
background-color: white;
border-color: grey;
color: grey;
}
.e-checkicon.e-checkbox-wrapper:hover .e-frame.e-check {
background-color: white;
border-color: grey;
color: grey;
}
.e-checkicon.e-checkbox-wrapper .e-checkbox:focus + .e-frame.e-check {
background-color: white;
border-color: grey;
box-shadow: none;
color: grey;
}
</style>
```

Output will be as follows

- ☒ Buy Groceries
- ☒ Pay Rent
- ☐ Make Dinner
- ☐ Finish To-do List Article

Custom Check Icon

Checkbox check icon can be customized as per the requirement by adding CSS rules.

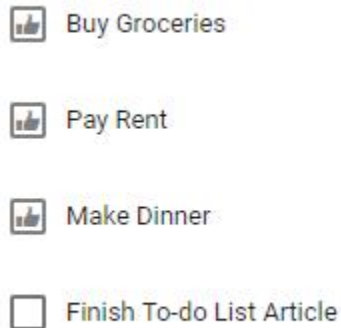
In the following example, the check icon can be customized by changing check icon content, background and border color in focus and hovered states by adding `e-checkicon` class.

ASPX-CS

```
@using Syncfusion.Blazor.Buttons
<SfCheckBox Label="Buy Groceries" @bind-Checked="isChecked" CssClass="e-checkicon"></SfCheckBox><br />
<SfCheckBox Label="Pay Rent" @bind-Checked="isRentChecked" CssClass="e-checkicon"></SfCheckBox><br />
<SfCheckBox Label="Make Dinner" @bind-Checked="isDinnerChecked" CssClass="e-checkicon"></SfCheckBox><br />
<SfCheckBox Label="Finish To-do List Article" @bind-Checked="isArticleChecked" CssClass="e-checkicon"></SfCheckBox>
@code {
private bool isChecked = true;
private bool isRentChecked = true;
private bool isDinnerChecked = true;
private bool isArticleChecked = false;
}
<style>
.e-checkicon.e-checkbox-wrapper .e-frame.e-check::before {
content: '\e682';
}
.e-checkicon.e-checkbox-wrapper .e-check {
font-size: 12px;
}
.e-checkicon.e-checkbox-wrapper .e-frame.e-check,
.e-checkicon.e-checkbox-wrapper:hover .e-frame.e-check {
background-color: white;
border-color: grey;
color: grey;
}
.e-checkicon.e-checkbox-wrapper .e-checkbox:focus + .e-frame.e-check {
background-color: white;
border-color: grey;
box-shadow: none;
color: grey;
}
.e-checkicon.e-checkbox-wrapper .e-ripple-element {
background: grey;
}
```

```
}
</style>
```

Output will be as follows



Model Binding in Blazor CheckBox Component

This section demonstrates the strongly typed extension support in Checkbox. The view that can bind with any model is called as strongly typed view. You can bind any class as model to view. You can access model properties on that view. You can use data associated with model to render the component.

In this sample, first check the option and click the submit button to post the selected value in the Checkbox. When the value is not checked, validation error message will be shown below the Checkbox.

CSHARP

```
<EditForm Model="Annotate">
<DataAnnotationsValidator></DataAnnotationsValidator>
<div class="form-group">
<SfCheckBox Label="Option 1" @bind-Checked="@Annotate.Check"></SfCheckBox>
<ValidationMessage For="@(() => Annotate.Check)" />
</div>
<SfButton HtmlAttributes="@Submit" Content="Submit"></SfButton>
</EditForm>
@code {
public Annotation Annotate = new Annotation();
public class Annotation
{
[Range(typeof(bool), "true", "true", ErrorMessage = "You need to agree to
the Terms and Conditions")]
public bool Check { get; set; }
}
public Dictionary<string, object> Submit = new Dictionary<string, object>()
{
{ "type", "submit"}
};
}
```

Output be like

☐ Option 1

You need to agree to the Terms and Conditions

SUBMIT

Right-To-Left in Blazor CheckBox Component

Checkbox 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 Checkbox component.

ASPX-CS

```
@using Syncfusion.Blazor.Buttons
<SfCheckBox Label="Default" @bind-Checked="isChecked"
EnableRtl="true"></SfCheckBox>
@code {
private bool isChecked = true;
}
```

Output will be as follows

Default ☒

Chip

Getting Started with Blazor Chip Component

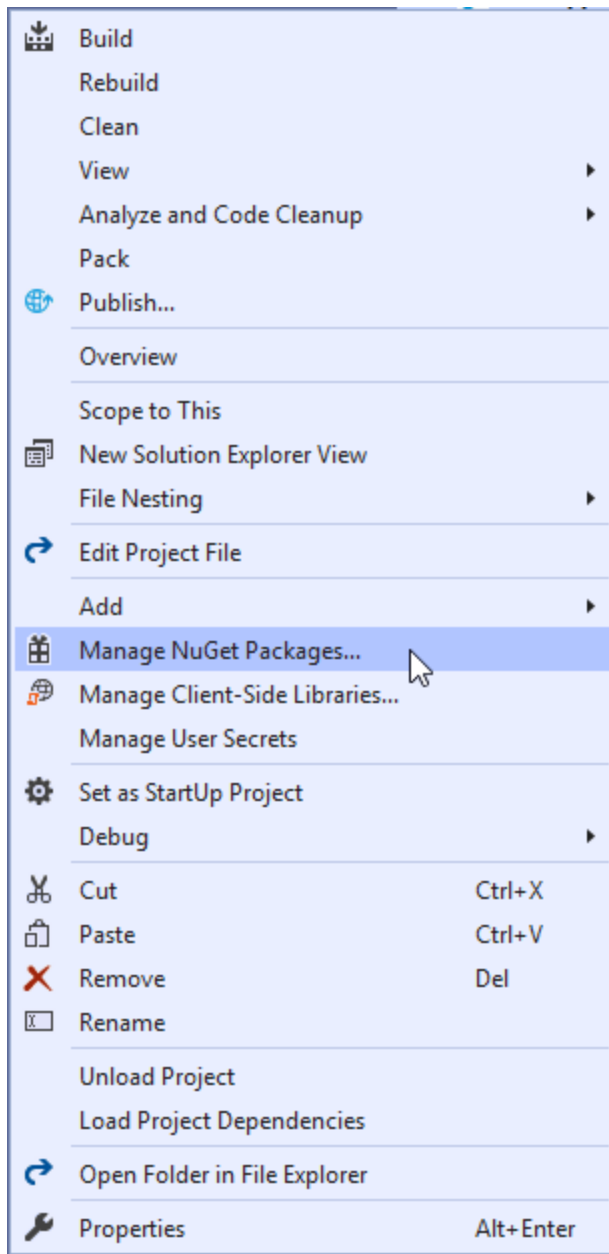
This section briefly explains how to include a [Chip](#) in your Blazor Server-Side application. Refer to the [Getting Started with Syncfusion Blazor for Server-Side in Visual Studio page](#) for the introduction and configuring the common specifications.

Importing Syncfusion Blazor component in the application

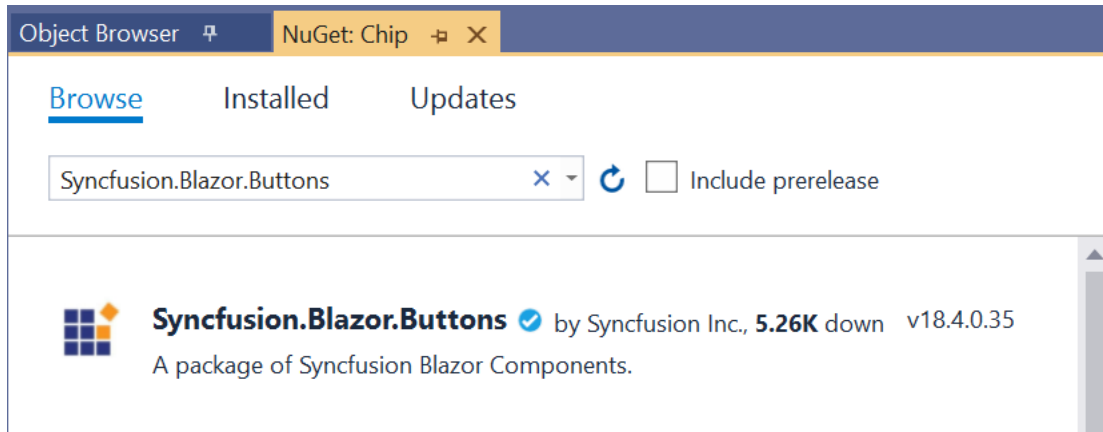
Using Syncfusion.Blazor NuGet Package [New standard]

1. Install [Syncfusion.Blazor.Buttons](#) NuGet package to the application by using the [NuGet Package Manager](#).

Refer to the Individual NuGet Packages section for the available NuGet packages.



2. Search Syncfusion.Blazor.Buttons keyword in the Browse tab and install Syncfusion.Blazor.Buttons NuGet package in the application.



3. Once the installation process is completed, the Syncfusion Blazor Buttons package will be installed in the project.

Warning: Syncfusion.Blazor package should not be installed along with [individual NuGet packages](#). Hence, you have to add the below Syncfusion.Blazor.Themes static web assets (styles) in the application.

You can add the client-side style resources through [CDN](#) or from [NuGet](#) package to the <head> element of the ~/wwwroot/index.html page in Blazor WebAssembly app or ~/Pages/_Host.cshtml page in Blazor Server app.

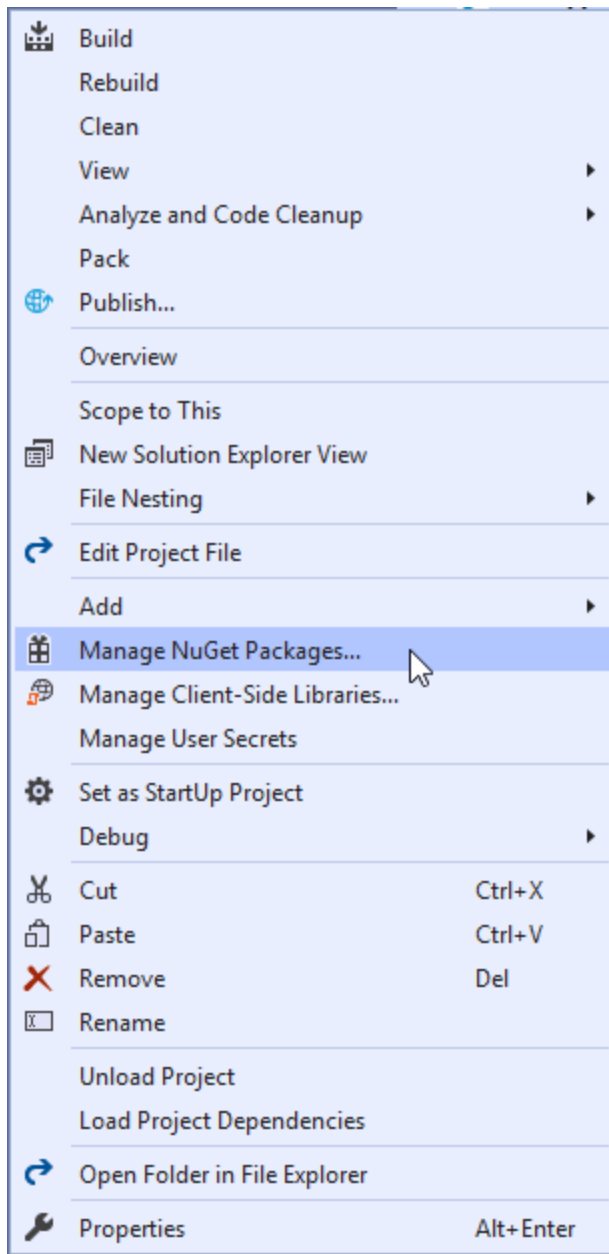
HTML

```
<head>
....
....
<link href="_content/Syncfusion.Blazor.Themes/bootstrap4.css"
rel="stylesheet" />
</head>
```

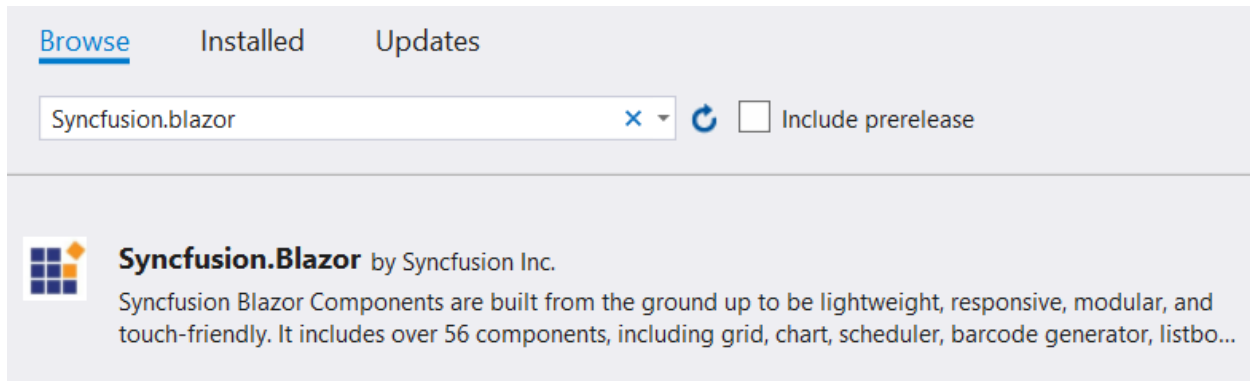
Warning: If you prefer the above new standard (individual NuGet packages), then skip this section. Using both old and new standards in the same application will throw ambiguous compilation errors.

Using Syncfusion.Blazor NuGet Package [Old standard]

1. Install **Syncfusion.Blazor** NuGet package to the application by using the **NuGet Package Manager**. Right-click the project and then select Manage NuGet Packages.



2. Search Syncfusion.Blazor keyword in the Browse tab and install Syncfusion.Blazor NuGet package in the application.



- Once the installation process is completed, the Syncfusion Blazor package will be installed in the project. You can add the client-side style resources using NuGet package to the `element` of the `~/wwwroot/index.html` page in Blazor WebAssembly app or `~/Pages/_Host.cshtml` page in Blazor Server app.

HTML

```
<head>
....
....
<link href="_content/Syncfusion.Blazor/styles/bootstrap4.css"
rel="stylesheet" />
</head>
```

HTML

```
<head>
<link href="https://cdn.syncfusion.com/blazor/{{ site.blazorversion
}}/styles/bootstrap4.css" rel="stylesheet" />
</head>
```

For Internet Explorer 11 kindly refer the polyfills. Refer the [documentation](#) for more information.

HTML

```
<head>
<link href="_content/Syncfusion.Blazor/styles/bootstrap4.css"
rel="stylesheet" />
<script
src="https://github.com/Daddoon/Blazor.Polyfill/releases/download/3.0.1/blaz
or.polyfill.min.js"></script>
</head>
```

Add Syncfusion Blazor service in Startup.cs (Server-side application)

Open the **Startup.cs** file and add services required by Syncfusion components using `services.AddSyncfusionBlazor()` method. Add this method in the `ConfigureServices` function as follows.

CSHARP

```
using Syncfusion.Blazor;
namespace BlazorApplication
{
    public class Startup
    {
        ....
        ....
        public void ConfigureServices(IServiceCollection services)
        {
            ....
            ....
            services.AddSyncfusionBlazor();
        }
    }
}
```

Add Syncfusion Blazor service in Program.cs (Client-side application)

Open the **Program.cs** file and add services required by Syncfusion components using `builder.services.AddSyncfusionBlazor()` method. Add this method in the **Main** function as follows.

C#

```
using Syncfusion.Blazor;
namespace BlazorApplication
{
    public class Program
    {
        public static async Task Main(string[] args)
        {
            ....
            ....
            builder.Services.AddSyncfusionBlazor();
        }
    }
}
```

To enable custom client side resource loading from CRG or CDN. You need to disable resource loading by **AddSyncfusionBlazor(true)** and load the scripts to the `<head>` element of the `~/wwwroot/index.html` page in Blazor WebAssembly app or `~/Pages/_Host.cshtml` page in Blazor Server app.

HTML

```
<head>
<script src="https://cdn.syncfusion.com/blazor/{{ site.blazorversion
}}/syncfusion-blazor.min.js"></script>
</head>
```

Adding component package to the application

Open `~/_Imports.razor` file and import the `Syncfusion.Blazor.Buttons` package.

ASPX-CS

```
@using Syncfusion.Blazor
@using Syncfusion.Blazor.Buttons
```

Adding Chip component to the application

Now, add the Syncfusion Blazor Chip component in any web page razor in the Pages folder. For example, the Chip component is added in the ~/Pages/Index.razor page.

ASPX-CS

```
@using Syncfusion.Blazor.Buttons
<SfChip>
<ChipItems>
<ChipItem Text="Janet Leverling"></ChipItem>
</ChipItems>
</SfChip>
```

Run the application

After successful compilation of your application, simply press F5 to run the application.

Output be like the below.



See Also

[Getting Started with Syncfusion Blazor for Client-Side in .NET Core CLI](#)

[Getting Started with Syncfusion Blazor for Client-Side in Visual Studio 2019](#)

[Getting Started with Syncfusion Blazor for Server-Side in .NET Core CLI](#)

Types in Blazor Chip Component

The Chip control has the following types.

- Input Chip
- Choice Chip
- Filter Chip
- Action Chip

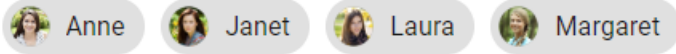
Input Chip

Input Chip holds information in compact form. It converts user input into chips.

ASPX-CS

```
@using Syncfusion.Blazor.Buttons
<SfChip>
<ChipItems>
<ChipItem Text="Anne" LeadingIconUrl="./anne.png"></ChipItem>
<ChipItem Text="Janet" LeadingIconUrl="./janet.png"></ChipItem>
<ChipItem Text="Laura" LeadingIconUrl="./laura.png"></ChipItem>
<ChipItem Text="Margaret" LeadingIconUrl="./margaret.png"></ChipItem>
</ChipItems>
</SfChip>
```

Output be like the below.



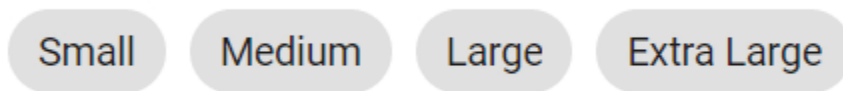
Choice Chip

Choice Chip allows you to select a single chip from the set of Chip/ChipItems. It can be enabled by setting the `Selection` property to `Single`.

ASPX-CS

```
@using Syncfusion.Blazor.Buttons
<SfChip Selection="SelectionType.Single">
  <ChipItems>
    <ChipItem Text="Small"></ChipItem>
    <ChipItem Text="Medium"></ChipItem>
    <ChipItem Text="Large"></ChipItem>
    <ChipItem Text="Extra Large"></ChipItem>
  </ChipItems>
</SfChip>
```

Output be like the below.



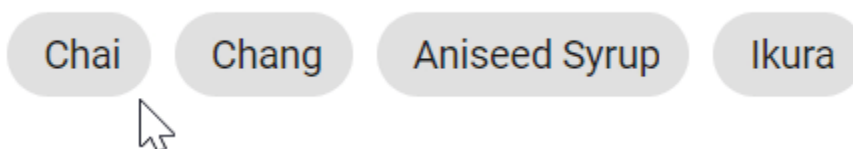
Filter Chip

Filter Chip allows you to select a multiple chip from the set of Chip/ChipItems. It can be enabled by setting the `Selection` property to `Multiple`.

ASPX-CS

```
@using Syncfusion.Blazor.Buttons
<SfChip Selection="SelectionType.Multiple">
  <ChipItems>
    <ChipItem Text="Chai"></ChipItem>
    <ChipItem Text="Chang"></ChipItem>
    <ChipItem Text="Aniseed Syrup"></ChipItem>
    <ChipItem Text="Ikura"></ChipItem>
  </ChipItems>
</SfChip>
```

Output be like the below.




Action Chip

The Action Chip triggers the event like click or delete, which helps doing action based on the event.

ASPX-CS

```
@using Syncfusion.Blazor.Buttons
<SfChip>
<ChipEvents OnClick="@OnClick"></ChipEvents>
<ChipItems>
<ChipItem Text="Sent a text"></ChipItem>
<ChipItem Text="Set a remainder"></ChipItem>
<ChipItem Text="Read my emails"></ChipItem>
<ChipItem Text="Set alarm"></ChipItem>
</ChipItems>
</SfChip>
<div>@ChipText</div>
@code
{
public string ChipText = "";
private void OnClick(Syncfusion.Blazor.Buttons.ChipEventArgs args)
{
ChipText = args.Text;
this.StateHasChanged();
}
}
```

Output be like the below.

Sent a text

Set a remainder

Read my emails

Set alarm

Deletable Chip

Deletable Chip allows you to delete a chip from Chip/ChipItems. It can be enabled by setting the `EnableDelete` property to `true`.

ASPX-CS

```
@using Syncfusion.Blazor.Buttons
<SfChip EnableDelete="true">
<ChipItems>
<ChipItem Text="Sent a text"></ChipItem>
<ChipItem Text="Set a remainder"></ChipItem>
<ChipItem Text="Read my emails"></ChipItem>
<ChipItem Text="Set alarm"></ChipItem>
</ChipItems>
</SfChip>
```

Output be like the below.

Sent a text ✕

Set a remainder ✕

Read my emails ✕

Set alarm ✕

Customization in Blazor Chip Component

This section explains the customization of styles, leading icons, avatar, and trailing icons in Chip control.

Styles

The Chip control has the following predefined styles that can be defined using the `CssClass` property.

Class	Description
-----	-----
e-primary	Represents a primary chip.
e-success	Represents a positive chip.
e-info	Represents an informative chip.
e-warning	Represents a chip with caution.
e-danger	Represents a negative chip.

ASPX-CS

```
@using Syncfusion.Blazor.Buttons
<SfChip>
<ChipItems>
<ChipItem Text="Default"></ChipItem>
<ChipItem Text="Primary" CssClass="e-primary"></ChipItem>
<ChipItem Text="Success" CssClass="e-success"></ChipItem>
<ChipItem Text="Info" CssClass="e-info"></ChipItem>
<ChipItem Text="Warning" CssClass="e-warning"></ChipItem>
<ChipItem Text="Danger" CssClass="e-danger"></ChipItem>
</ChipItems>
</SfChip>
```

Output will be like the below.

Default

Primary

Success

Info

Warning

Danger

Leading Icon

You can add and customize the leading icon of chip using the `LeadingIconCss` property.

ASPX-CS

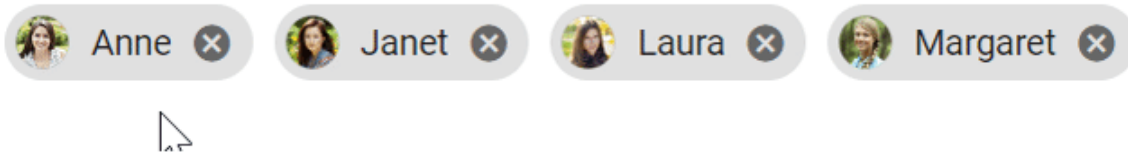
```
@using Syncfusion.Blazor.Buttons
<SfChip ID="chip-avatar" EnableDelete="true">
<ChipItems>
<ChipItem Text="Anne" LeadingIconCss="anne"></ChipItem>
<ChipItem Text="Janet" LeadingIconCss="janet"></ChipItem>
<ChipItem Text="Laura" LeadingIconCss="laura"></ChipItem>
<ChipItem Text="Margaret" LeadingIconCss="margaret"></ChipItem>
```

```

</ChipItems>
</SfChip>
<style>
#chip-avatar .anne {
background-image: url('./anne.png')
}
#chip-avatar .margaret {
background-image: url('./margaret.png')
}
#chip-avatar .laura {
background-image: url('./laura.png')
}
#chip-avatar .janet {
background-image: url('./janet.png')
}
</style>

```

Output will be like the below.



Avatar

You can add and customize the avatar of chip using the `LeadingIconCss` property.

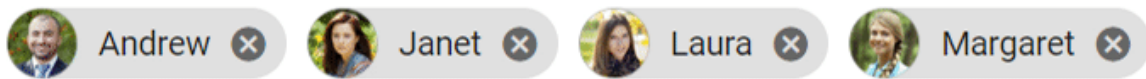
CSHARP

```

<SfChip EnableDelete="true" CssClass="e-leading-avatar">
<ChipItems>
<ChipItem Text="Andrew" LeadingIconCss="andrew"></ChipItem>
<ChipItem Text="Janet" LeadingIconCss="janet"></ChipItem>
<ChipItem Text="Laura" LeadingIconCss="laura"></ChipItem>
<ChipItem Text="Margaret" LeadingIconCss="margaret"></ChipItem>
</ChipItems>
</SfChip>
<style>
.chip-avatar .andrew {
background-image: url('./andrew.png')
}
.chip-avatar .margaret {
background-image: url('./margaret.png')
}
.chip-avatar .laura {
background-image: url('./laura.png')
}
.chip-avatar .janet {
background-image: url('./janet.png')
}
</style>

```

Output will be like the below.



Leading Content

You can add and customize the avatar content of chip using the `LeadingText` property.

ASPX-CS

```
@using Syncfusion.Blazor.Buttons
<SfChip EnableDelete="true" CssClass="e-leading-avatar">
  <ChipItems>
    <ChipItem Text="Andrew" LeadingText="A"></ChipItem>
    <ChipItem Text="Janet" LeadingText="J"></ChipItem>
    <ChipItem Text="Laura" LeadingText="L"></ChipItem>
    <ChipItem Text="Margaret" LeadingText="M"></ChipItem>
  </ChipItems>
</SfChip>
```

Output will be like the below.



Trailing Icon

You can add and customize the trailing icon of chip using the `TrailingIconCss` property.

ASPX-CS

```
@using Syncfusion.Blazor.Buttons
<SfChip>
  <ChipItems>
    <ChipItem Text="Andrew" TrailingIconCss="e-dlt-btn"></ChipItem>
    <ChipItem Text="Janet" TrailingIconCss="e-dlt-btn"></ChipItem>
    <ChipItem Text="Laura" TrailingIconCss="e-dlt-btn"></ChipItem>
    <ChipItem Text="Margaret" TrailingIconCss="e-dlt-btn"></ChipItem>
  </ChipItems>
</SfChip>
```

Output will be like the below.



Outline Chip

Outline chip has the border with the background transparent. It can be set using the `CssClass` property.

ASPX-CS

```
@using Syncfusion.Blazor.Buttons
<SfChip EnableDelete="true">
  <ChipItems>
    <ChipItem CssClass="e-outline" Text="Andrew"></ChipItem>
    <ChipItem CssClass="e-outline" Text="Janet"></ChipItem>
    <ChipItem CssClass="e-outline" Text="Laura"></ChipItem>
    <ChipItem CssClass="e-outline" Text="Margaret"></ChipItem>
  </ChipItems>
</SfChip>
```

Output will be like the below.



CSS Structure in Blazor Chip 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 chip text

Use the following CSS to customize the chip text properties.

CSS

```
.e-chip .e-chip-text {
font-size: 20px;
color: black;
font-weight: normal;
}
```

Customizing the chip icon

Use the following CSS to customize the chip icon properties.

CSS

```
.e-chip .e-icon {
background-image:
url('https://ej2.syncfusion.com/demos/src/chips/images/laura.png');
opacity: 0.8;
}
```

Customizing the chip delete button

Use the following CSS to customize the chip delete button.

CSS

```
.e-chip-list .e-chip .e-chip-delete.e-dlt-btn {
color: #e3165b;
font-size: 12px;
}
```

Customizing the chip outline

Use the following CSS to customize the chip outline.

CSS

```
.e-chip-list .e-chip.e-outline {
border-color: #e3165b;
border-width: 3px;
}
```

Customizing the chip on selection

Use the following CSS to customize the chip on selection.

CSS

```
/* To customize single chip on selection */
.e-chip-list.e-selection .e-chip.e-active {
background-color: #ffcalc;
color: #e3165b;
}
/* To customize multiple chip on selection */
.e-chip-list .e-chip.e-active {
background-color: #e3165b;
color: white;
}
```

Customizing the chip avatar text

Use the following CSS to customize the chip avatar text properties.

CSS

```
.e-chip-list .e-chip .e-chip-avatar {
background-color: #d51a1a;
color: #fafafa;
}
```

Accessibility in Blazor Chip Component

Keyboard Interaction

The following shortcut keys are used to access the Chip control without any interruption.

| Keyboard shortcuts | Actions |

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

| Enter | Selects the targeted chip from the Chip/ChipItems. |

| Delete | Deletes the targeted chip from the Chip/ChipItems. |

ASPX-CS

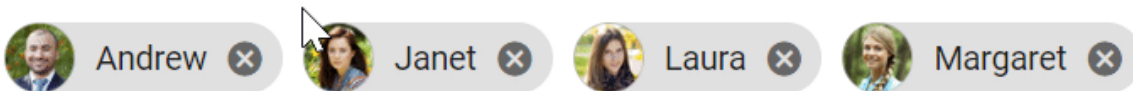
```
@using Syncfusion.Blazor.Buttons
<SfChip ID="chip-avatar" EnableDelete="true" CssClass="e-chip-avatar"
Selection="SelectionType.Single">
<ChipItems>
```

```

<ChipItem Text="Andrew" LeadingIconCss='andrew'></ChipItem>
<ChipItem Text="Janet" LeadingIconCss='janet'></ChipItem>
<ChipItem Text="Laura" LeadingIconCss='laura'></ChipItem>
<ChipItem Text="Margaret" LeadingIconCss='margaret'></ChipItem>
</ChipItems>
</SfChip>
<style>
#chip-avatar .andrew {
background-image: url('./andrew.png')
}
#chip-avatar .margaret {
background-image: url('./margaret.png')
}
#chip-avatar .laura {
background-image: url('./laura.png')
}
#chip-avatar .janet {
background-image: url('./janet.png')
}
</style>

```

Output will be like the below.



Circular Gauge

Getting Started with Blazor Circular Gauge Component

This section briefly explains how to include a Circular Gauge component in your Blazor server-side application. You can refer to our [Getting Started with Syncfusion Blazor for server-side in Visual Studio](#) page for introduction and configuring common specifications.

Importing Syncfusion Blazor Circular Gauge component in the application

1. Install **Syncfusion.Blazor.CircularGauge** NuGet package in the application by using the **NuGet Package Manager**.
2. You can add the client-side resources through [CDN](#) or from [NuGet](#) package in the element of the `~/Pages/_Host.cshtml` page.

HTML

```

<head>
<link href="_content/Syncfusion.Blazor.Themes/bootstrap4.css"
rel="stylesheet" />
<!--CDN-->
@*<link href="https://cdn.syncfusion.com/blazor/{ site.blazorversion
}/styles/bootstrap4.css" rel="stylesheet" />*@
</head>

```

For Internet Explorer 11, kindly refer the polyfills. Refer the [documentation](#) for more information.

HTML

```
<head>
<link href="https://cdn.syncfusion.com/blazor/{{ site.blazorversion
}}/styles/bootstrap4.css" rel="stylesheet" />
<script
src="https://github.com/Daddoon/Blazor.Polyfill/releases/download/3.0.1/blaz
or.polyfill.min.js"></script>
</head>
```

Adding component package to an application

Open the `~/_Imports.razor` file and include the **Syncfusion.Blazor.CircularGauge** namespace.

ASPX-CS

```
@using Syncfusion.Blazor.CircularGauge
```

Adding SyncfusionBlazor Service in Startup.cs

Open the **Startup.cs** file and add services required by Syncfusion components using **services.AddSyncfusionBlazor()** method. Add this method in the **ConfigureServices** function as follows.

CSHARP

```
using Syncfusion.Blazor;
namespace BlazorApplication
{
    public class Startup
    {
        ....
        ....
        public void ConfigureServices(IServiceCollection services)
        {
            ....
            ....
            services.AddSyncfusionBlazor();
        }
    }
}
```

To enable custom client-side source loading from CRG or CDN, please refer to the section about [custom resources in Blazor application](#).

Adding Circular Gauge component

The Syncfusion Circular Gauge component can be initialized in any razor page inside the `~/Pages` folder. For example, the Circular Gauge component is added to the `~/Pages/Index.razor` page. In a new application, if **Index.razor** page has any default content template, then those content can be completely removed and the following code can be added.

ASPX-CS

```
@page "/"
<SfCircularGauge>
<CircularGaugeAxes>
```

```
<CircularGaugeAxis>  
<CircularGaugePointers>  
<CircularGaugePointer></CircularGaugePointer>  
</CircularGaugePointers>  
</CircularGaugeAxis>  
</CircularGaugeAxes>  
</SfCircularGauge>
```

On successful compilation of your application, the Syncfusion Blazor Circular Gauge component will render in the web browser as shown below.



Set pointer value

Pointers are used to indicate values on an axis. You can change the pointer value using the [Value](#) property in [CircularGaugePointer](#).

In Circular Gauge, you can configure multiple axes. On each axis, you can add a pointer.

ASPX-CS

```
<SfCircularGauge>  
<CircularGaugeAxes>  
<CircularGaugeAxis>  
<CircularGaugePointers>  
<CircularGaugePointer Value="35">  
</CircularGaugePointer>  
</CircularGaugePointers>  
</CircularGaugeAxis>  
</CircularGaugeAxes>  
</SfCircularGauge>
```

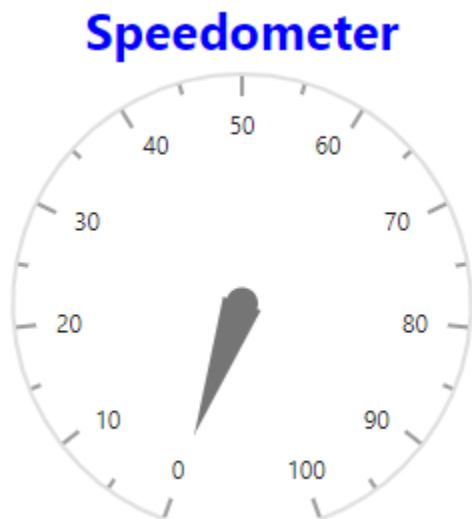


Adding title for Circular Gauge

Title can be added to the circular gauge to provide a quick information to the users about the context of the rendered circular gauge. You can add a title using [Title](#) property in [SfCircularGauge](#).

ASPX-CS

```
<SfCircularGauge Title="Speedometer">
  <CircularGaugeTitleStyle Color="blue" FontWeight="bold"
    Size="25"></CircularGaugeTitleStyle>
  <CircularGaugeAxes>
    <CircularGaugeAxis>
      <CircularGaugePointers>
        <CircularGaugePointer>
        </CircularGaugePointer>
      </CircularGaugePointers>
    </CircularGaugeAxis>
  </CircularGaugeAxes>
</SfCircularGauge>
```



Adding ranges in the Circular Gauge

Range is used to specify a group of scale values in the gauge. We can set the range start and end using [Start](#) and [End](#) properties in the [CircularGaugeRange](#).

ASPX-CS

```
<SfCircularGauge>
  <CircularGaugeAxes>
    <CircularGaugeAxis>
      <CircularGaugeRanges>
        <CircularGaugeRange Start="40" End="80">
        </CircularGaugeRange>
      </CircularGaugeRanges>
    </CircularGaugeAxis>
  </CircularGaugeAxes>
</SfCircularGauge>
```



See also

- [Getting Started with Syncfusion Blazor for Client-Side in .NET Core CLI](#)
- [Getting Started with Syncfusion Blazor for Server-Side in Visual Studio](#)
- [Getting Started with Syncfusion Blazor for Server-Side in .NET Core CLI](#)

Dimensions in Blazor Circular Gauge Component

Size for Circular Gauge

You can set size for the Circular Gauge directly using the [Width](#) and [Height](#) properties.

In pixel

You can set the size of the Circular Gauge in pixel as demonstrated below.

ASPX-CS

```
@using Syncfusion.Blazor.CircularGauge
<SfCircularGauge Width= "200px" Height= "200px"></SfCircularGauge>
```



In percentage

By setting value in percentage, gauge gets its dimension with respect to its container. For example, when

the height is '50%', gauge is rendered to half of the container height.

ASPX-CS

```
@using Syncfusion.Blazor.CircularGauge
<div style="height:450px; width:450px">
<SfCircularGauge Width="50%" Height="50%"></SfCircularGauge>
</div>
```



When you do not specify the size, it takes 450 pixels as the height and takes window size as its width.

Axes in Blazor Circular Gauge Component

By default, the Circular Gauge will be displayed with an axis. Each axis contains its own ranges, pointers, and annotations.

Axis customization

You can customize the [Width](#) and [Color](#) of an axis line using the [CircularGaugeAxisLineStyle](#) tag. The background for an axis can be customized using the [Background](#) property.

ASPX-CS

```
@using Syncfusion.Blazor.CircularGauge
<SfCircularGauge>
  <CircularGaugeAxes>
    <CircularGaugeAxis Background= "rgba(0, 128, 128, 0.3)">
      <CircularGaugeAxisLineStyle Width="2" Color="red">
      </CircularGaugeAxisLineStyle>
    </CircularGaugeAxis>
  </CircularGaugeAxes>
</SfCircularGauge>
```



Minimum and maximum

The [Minimum](#) and [Maximum](#) properties enables you to customize the start and end values of an axis.

ASPX-CS

```
@using Syncfusion.Blazor.CircularGauge
<SfCircularGauge>
  <CircularGaugeAxes>
    <CircularGaugeAxis Minimum= "50" Maximum= "250">
    </CircularGaugeAxis>
  </CircularGaugeAxes>
</SfCircularGauge>
```

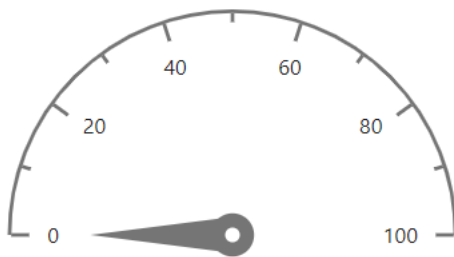


Start and end angle

You can sweep the Circular Gauge axis from 0 to 360 degrees. By default, the start angle of an axis is 200 degrees and end angle of an axis is 160 degrees. You can customize this option using the [StartAngle](#) and [EndAngle](#) properties.

ASPX-CS

```
@using Syncfusion.Blazor.CircularGauge
<SfCircularGauge>
  <CircularGaugeAxes>
    <CircularGaugeAxis StartAngle= "270" EndAngle= "90">
    </CircularGaugeAxis>
  </CircularGaugeAxes>
</SfCircularGauge>
```



Axis radius

By default, the radius of an axis is calculated based on the available size. You can customize the axis radius using the [Radius](#) property. It takes values either in [percentage](#) or in [pixel](#).

In pixel

You can set the radius of the Circular Gauge in pixel as shown below.

ASPX-CS

```
@using Syncfusion.Blazor.CircularGauge
<SfCircularGauge>
  <CircularGaugeAxes>
    <CircularGaugeAxis Radius="150px">
    </CircularGaugeAxis>
  </CircularGaugeAxes>
</SfCircularGauge>
```

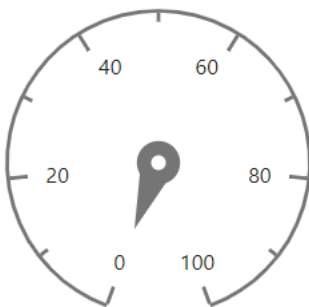


In percentage

By setting value in percentage, Circular Gauge gets its dimension with respect to its available size. For example, when the [Radius](#) is '50%', gauge renders to the half of the available size.

ASPX-CS

```
@using Syncfusion.Blazor.CircularGauge
<SfCircularGauge>
  <CircularGaugeAxes>
    <CircularGaugeAxis Radius="50%"></CircularGaugeAxis>
  </CircularGaugeAxes>
</SfCircularGauge>
```



Ticks

You can customize the [Height](#), [Color](#), and [Width](#) of major ticks and minor ticks using the [CircularGaugeAxisMajorTicks](#) and [CircularGaugeAxisMinorTicks](#) tags.

By default, [Interval](#) for [CircularGaugeAxisMajorTicks](#) will be calculated automatically. You can customize the interval for major ticks and minor ticks using the [Interval](#) property.

ASPX-CS

```
@using Syncfusion.Blazor.CircularGauge
<SfCircularGauge>
  <CircularGaugeAxes>
    <CircularGaugeAxis>
      <CircularGaugeAxisMajorTicks Interval="10" Color="red" Height="10"
        Width="3">
      </CircularGaugeAxisMajorTicks>
      <CircularGaugeAxisMinorTicks Interval="5" Color="green" Height="5"
        Width="2">
      </CircularGaugeAxisMinorTicks>
    </CircularGaugeAxis>
  </CircularGaugeAxes>
```

```
</SfCircularGauge>
```



Tick position

The minor ticks and major ticks can be positioned using the [Offset](#) and [Position](#) properties.

- The [Offset](#) defines the distance between the axis and ticks. By default, offset value is 0.
- The [Position](#) will place the ticks on the axis. By default, ticks will be placed inside the axis. Its possible values are 'Position.Inside', 'Position.Outside', and 'Position.Cross'.

ASPX-CS

```
@using Syncfusion.Blazor.CircularGauge
<SfCircularGauge>
  <CircularGaugeAxes>
    <CircularGaugeAxis>
      <CircularGaugeAxisMajorTicks Color="red"
      Height="10" Width="3"
      Position = "Position.Inside"
      Offset="5">
    </CircularGaugeAxisMajorTicks>
    <CircularGaugeAxisMinorTicks Color="green"
      Height="5" Width="2"
      Position = "Position.Inside"
      Offset="5">
    </CircularGaugeAxisMinorTicks>
    </CircularGaugeAxis>
  </CircularGaugeAxes>
</SfCircularGauge>
```



Labels

The labels of an axis can be customized using the [CircularGaugeAxisLabelFont](#) tag in [CircularGaugeAxisLabelStyle](#) option.

ASPX-CS

```
@using Syncfusion.Blazor.CircularGauge
<SfCircularGauge>
  <CircularGaugeAxes>
    <CircularGaugeAxis>
      <CircularGaugeAxisLabelStyle>
        <CircularGaugeAxisLabelFont Color="red" Size="20px" FontWeight="Bold">
        </CircularGaugeAxisLabelFont>
      </CircularGaugeAxisLabelStyle>
    </CircularGaugeAxis>
  </CircularGaugeAxes>
</SfCircularGauge>
```



Label position

The labels can be moved using the [Offset](#) or [Position](#) property.

- The [Offset](#) defines the distance between the labels and ticks. By default, offset value is 0.
- The [Position](#) specifies the label position. By default, the labels will be placed inside the axis. Its possible values are 'Position.Inside', 'Position.Outside' and 'Position.Cross'.

ASPX-CS

```
@using Syncfusion.Blazor.CircularGauge
<SfCircularGauge>
  <CircularGaugeAxes>
    <CircularGaugeAxis>
      <CircularGaugeAxisLabelStyle Position="Position.Outside"
      Offset="5">
      </CircularGaugeAxisLabelStyle>
    </CircularGaugeAxis>
  </CircularGaugeAxes>
</SfCircularGauge>
```



Auto angle

The labels can be swept along the axis angle by enabling the [AutoAngle](#) property.

ASPX-CS

```
@using Syncfusion.Blazor.CircularGauge
<SfCircularGauge>
  <CircularGaugeAxes>
    <CircularGaugeAxis>
      <CircularGaugeAxisLabelStyle AutoAngle="true"></CircularGaugeAxisLabelStyle>
    </CircularGaugeAxis>
  </CircularGaugeAxes>
</SfCircularGauge>
```



Smart labels

When an axis makes a complete circle, then the first and last labels of the axis will overlap with each other. So, you need to either hide first or last label using the [HiddenLabel](#) property. When 'HiddenLabel' value is [First](#), the first label will be hidden and when the 'HiddenLabel' value is [Last](#), the last label will be hidden.

ASPX-CS

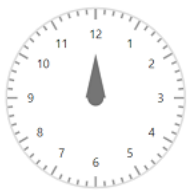
```
@using Syncfusion.Blazor.CircularGauge
<SfCircularGauge>
  <CircularGaugeAxes>
    <CircularGaugeAxis Minimum="0" HiddenLabel="First"></CircularGaugeAxis>
  </CircularGaugeAxes>
</SfCircularGauge>
```



```

Maximum="12"
EndAngle="360"
StartAngle="0">
<CircularGaugeAxisLabelStyle Position="Position.Inside"
HiddenLabel="HiddenLabel.First">
</CircularGaugeAxisLabelStyle>
<CircularGaugeAxisMajorTicks Interval="1"
Height="10"
Position="Position.Inside">
</CircularGaugeAxisMajorTicks>
<CircularGaugeAxisMinorTicks Interval="0.2"
Height="5"
Position="Position.Inside">
</CircularGaugeAxisMinorTicks>
</CircularGaugeAxis>
</CircularGaugeAxes>
</SfCircularGauge>

```



Label format

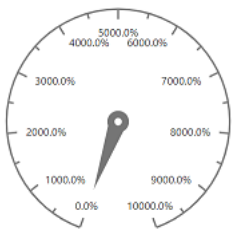
The axis labels can be formatted using the [Format](#) property in [CircularGaugeAxisLabelStyle](#), it supports all globalize format.

ASPX-CS

```

@using Syncfusion.Blazor.CircularGauge
<SfCircularGauge>
<CircularGaugeAxes>
<CircularGaugeAxis>
<CircularGaugeAxisLabelStyle Format="p1"></CircularGaugeAxisLabelStyle>
</CircularGaugeAxis>
</CircularGaugeAxes>
</SfCircularGauge>

```



The following table describes the result of applying some commonly used label formats on numeric values.

<!-- markdownlint-disable MD033 -->

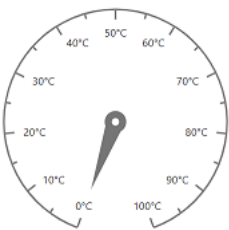
Label Value	Label Format property value	Result	Description
1000	n1	1000.0	The number is rounded to 1 decimal place.
1000	n2	1000.00	The number is rounded to 2 decimal places.
1000	n3	1000.000	The number is rounded to 3 decimal places.
0.01	p1	1.0%	The number is converted to percentage with 1 decimal place.
0.01	p2	1.00%	The number is converted to percentage with 2 decimal places.
0.01	p3	1.000%	The number is converted to percentage with 3 decimal places.
1000	c1	\$1,000.0	The currency symbol is appended to number, and the number is rounded to 1 decimal place.
1000	c2	\$1,000.00	The currency symbol is appended to number, and the number is rounded to 2 decimal places.

Custom label format

Axis labels support custom label format using placeholder like {value}°C, in which the value represent the axis label. e.g., 20°C.

ASPX-CS

```
@using Syncfusion.Blazor.CircularGauge
<SfCircularGauge>
  <CircularGaugeAxes>
    <CircularGaugeAxis>
      <CircularGaugeAxisLabelStyle
        Format="{value}°C"></CircularGaugeAxisLabelStyle>
    </CircularGaugeAxis>
  </CircularGaugeAxes>
</SfCircularGauge>
```



Show last label

If the maximum value does not enter the interval of major ticks, the last label will be hidden by default. If you want to display the last label, set the property value [ShowLastLabel](#) to true.

ASPX-CS

```
@using Syncfusion.Blazor.CircularGauge
<SfCircularGauge>
  <CircularGaugeAxes>
    <CircularGaugeAxis Maximum="100"
      ShowLastLabel="true">
      <CircularGaugeAxisMajorTicks Interval="30"></CircularGaugeAxisMajorTicks>
    </CircularGaugeAxis>
  </CircularGaugeAxes>
</SfCircularGauge>
```

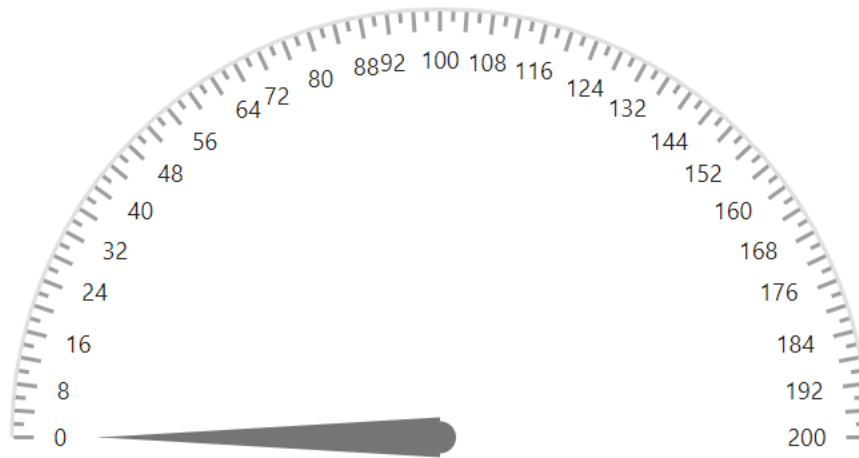


Hide intersecting axis labels

When the axis labels overlap with each other, you can hide the intersected labels by setting the `HideIntersectingLabel` property to true in the axis.

ASPX-CS

```
<SfCircularGauge>
  <CircularGaugeAxes>
    <CircularGaugeAxis Maximum="200" StartAngle="270" EndAngle="90" Minimum="0"
      HideIntersectingLabel="true">
      <CircularGaugeAxisMajorTicks Interval="4"></CircularGaugeAxisMajorTicks>
      <CircularGaugeAxisMinorTicks Interval="2"></CircularGaugeAxisMinorTicks>
    </CircularGaugeAxis>
  </CircularGaugeAxes>
</SfCircularGauge>
```



Axis direction

You can change the axis direction of the circular gauge using [Direction](#) property. Following axis directions are available in the circular gauge.

- ClockWise
- AntiClockWise

ASPX-CS

```
@using Syncfusion.Blazor.CircularGauge
<SfCircularGauge>
  <CircularGaugeAxes>
    <CircularGaugeAxis Direction="GaugeDirection.AntiClockWise">
    </CircularGaugeAxis>
  </CircularGaugeAxes>
</SfCircularGauge>
```



Multiple axes

In addition to the default axis, you can add a number of axis to a gauge. Each axis will have its own ranges, pointers, annotations, and customization options.

ASPX-CS

```
@using Syncfusion.Blazor.CircularGauge
<SfCircularGauge>
  <CircularGaugeAxes>
    <CircularGaugeAxis Minimum="0"
Maximum="140">
      <CircularGaugePointers>
        <CircularGaugePointer Type="PointerType.Needle">
        </CircularGaugePointer>
      </CircularGaugePointers>
    </CircularGaugeAxis>
    <CircularGaugeAxis Minimum="-20"
Maximum="60">
      <CircularGaugePointers>
        <CircularGaugePointer Type="PointerType.Marker"
MarkerShape="GaugeShape.InvertedTriangle"
MarkerHeight="20"
MarkerWidth="20">
        </CircularGaugePointer>
      </CircularGaugePointers>
      <CircularGaugeAxisMajorTicks Position="Position.Outside">
      </CircularGaugeAxisMajorTicks>
      <CircularGaugeAxisMinorTicks Position="Position.Outside">
      </CircularGaugeAxisMinorTicks>
      <CircularGaugeAxisLabelStyle Position="Position.Outside">
      </CircularGaugeAxisLabelStyle>
    </CircularGaugeAxis>
  </CircularGaugeAxes>
</SfCircularGauge>
```



Ranges in Blazor Circular Gauge Component

You can categorize certain interval on Circular Gauge axis using the [CircularGaugeRanges](#) tag.

Range start and end

The start and end values of a range in an axis can be customized using the [Start](#) and [End](#) properties.

ASPX-CS

```
@using Syncfusion.Blazor.CircularGauge
<SfCircularGauge>
  <CircularGaugeAxes>
    <CircularGaugeAxis>
      <CircularGaugeRanges>
        <CircularGaugeRange Start="40" End="80">
        </CircularGaugeRange>
      </CircularGaugeRanges>
    </CircularGaugeAxis>
  </CircularGaugeAxes>
</SfCircularGauge>
```



Start width and end width

Using [StartWidth](#) and [EndWidth](#) properties, you can customize the start width and end width of the range.

ASPX-CS

```
@using Syncfusion.Blazor.CircularGauge
```

```
<SfCircularGauge>
  <CircularGaugeAxes>
    <CircularGaugeAxis>
      <CircularGaugeRanges>
        <CircularGaugeRange Start="40"
End="80"
StartWidth="2"
EndWidth="20">
      </CircularGaugeRange>
    </CircularGaugeRanges>
  </CircularGaugeAxis>
</CircularGaugeAxes>
</SfCircularGauge>
```



Changing color

The color of a range can be customized using the [Color](#) and [Opacity](#) properties.

ASPX-CS

```
@using Syncfusion.Blazor.CircularGauge
<SfCircularGauge>
  <CircularGaugeAxes>
    <CircularGaugeAxis>
      <CircularGaugeRanges>
        <CircularGaugeRange Start="40"
End="80"
StartWidth="2"
EndWidth="20"
Color="blue"
Opacity="0.2">
      </CircularGaugeRange>
    </CircularGaugeRanges>
  </CircularGaugeAxis>
</CircularGaugeAxes>
</SfCircularGauge>
```



Range Position

The ranges can be placed either inside, outside or center of the axis using the [Position`] property in [CircularGaugeRange](#). Its possible values are 'PointerRangePosition.Inside', 'PointerRangePosition.Outside' and 'PointerRangePosition.Cross'.

ASPX-CS

```
@using Syncfusion.Blazor.CircularGauge
<SfCircularGauge>
  <CircularGaugeAxes>
    <CircularGaugeAxis>
      <CircularGaugeRanges>
        <CircularGaugeRange Start="40" End="80" StartWidth="15" EndWidth="15"
          Color="#ff5985" Position="PointerRangePosition.Cross">
        </CircularGaugeRange>
      </CircularGaugeRanges>
    </CircularGaugeAxis>
  </CircularGaugeAxes>
</SfCircularGauge>
```



Rounded corners

You can customize the corner radius using the [RoundedCornerRadius](#) property in [CircularGaugeRange](#).

ASPX-CS

```
@using Syncfusion.Blazor.CircularGauge
<SfCircularGauge>
```



```

<CircularGaugeAxes>
<CircularGaugeAxis>
<CircularGaugeRanges>
<CircularGaugeRange Start="40" End="80" RoundedCornerRadius="5">
</CircularGaugeRange>
</CircularGaugeRanges>
</CircularGaugeAxis>
</CircularGaugeAxes>
</SfCircularGauge>

```



Radius

You can place a range inside or outside the axis using the [Radius](#) property. The radius of a range takes value either in percentage or in pixels. By default, a range take 100% of the axis radius.

In pixel

You can set a radius of the range in pixel as demonstrated below.

ASPX-CS

```

@using Syncfusion.Blazor.CircularGauge
<SfCircularGauge>
<CircularGaugeAxes>
<CircularGaugeAxis>
<CircularGaugeRanges>
<CircularGaugeRange Start="40" End="80" Radius="100px">
</CircularGaugeRange>
</CircularGaugeRanges>
</CircularGaugeAxis>
</CircularGaugeAxes>
</SfCircularGauge>

```



In percentage

By setting value in percentage, a range gets its dimension with respect to its axis radius. For example, when the radius is '50%', the range is rendered to half of the axis radius.

ASPX-CS

```
@using Syncfusion.Blazor.CircularGauge
<SfCircularGauge>
  <CircularGaugeAxes>
    <CircularGaugeAxis>
      <CircularGaugeRanges>
        <CircularGaugeRange Start="40" End="80" Radius="50%">
        </CircularGaugeRange>
      </CircularGaugeRanges>
    </CircularGaugeAxis>
  </CircularGaugeAxes>
</SfCircularGauge>
```



<!-- markdownlint-disable MD010 -->

Dragging ranges

The ranges can be dragged over the axis line by clicking and dragging the same. To enable or disable the range drag, use the [EnableRangeDrag](#) property.

ASPX-CS

```
@using Syncfusion.Blazor.CircularGauge
<SfCircularGauge EnableRangeDrag="true" Height="250px" Width="250px">
  <CircularGaugeAxes>
    <CircularGaugeAxis>
```

```

<CircularGaugePointers>
<CircularGaugePointer Value="50">
</CircularGaugePointer>
</CircularGaugePointers>
<CircularGaugeRanges>
<CircularGaugeRange Start="0" End="100" Radius="108%" Color="#30B32D"
StartWidth="8" EndWidth="8">
</CircularGaugeRange>
</CircularGaugeRanges>
</CircularGaugeAxis>
</CircularGaugeAxes>
</SfCircularGauge>

```

Multiple ranges

You can add multiple ranges to an axis with the above customization as demonstrated below.

You can set the range color to ticks and labels of an axis by enabling the [UseRangeColor](#) property in [CircularGaugeAxisMajorTicks](#), [CircularGaugeAxisMinorTicks](#) and [CircularGaugeAxisLabelStyle](#) tags.

ASPX-CS

```

@using Syncfusion.Blazor.CircularGauge
<SfCircularGauge>
<CircularGaugeAxes>
<CircularGaugeAxis>
<CircularGaugeRanges>
<CircularGaugeRange Start="0" End="25" Radius="108%">
</CircularGaugeRange>
<CircularGaugeRange Start="25" End="50" Radius="70%">
</CircularGaugeRange>
<CircularGaugeRange Start="50" End="75" Radius="70%">
</CircularGaugeRange>
<CircularGaugeRange Start="75" End="100" Radius="108%">
</CircularGaugeRange>
</CircularGaugeRanges>
<CircularGaugeAxisLabelStyle UseRangeColor="true">
</CircularGaugeAxisLabelStyle>
<CircularGaugeAxisMinorTicks
UseRangeColor="true"></CircularGaugeAxisMinorTicks>
</CircularGaugeAxis>
</CircularGaugeAxes>
</SfCircularGauge>

```



Gradient Color

Gradient support allows to add multiple colors in the range and pointer of the circular gauge. The following gradient types are supported in the circular gauge.

- Linear Gradient
- Radial Gradient

Linear Gradient

Using linear gradient, colors will be applied in a linear progression. The start value of the linear gradient can be set using the [StartValue](#) property. The end value of the linear gradient will be set using the [EndValue](#) property. The color stop values such as color, opacity and offset are set using [ColorStop](#) property.

To apply linear gradient to the range, follow the below code sample.

ASPX-CS

```
@using Syncfusion.Blazor.CircularGauge
<SfCircularGauge CenterY="57%" Title="Shot Put Distance" Height="750">
<CircularGaugeAxes>
<CircularGaugeAxis StartAngle="200" EndAngle="130" Minimum="0" Maximum="14"
Radius="80%">
<CircularGaugeAxisLineStyle Width="0.001" />
<CircularGaugeAxisMajorTicks Width="0.01" />
<CircularGaugeAxisMinorTicks Width="0.01" />
<CircularGaugeAxisLabelStyle>
<CircularGaugeAxisLabelFont Size="0px" />
</CircularGaugeAxisLabelStyle>
<CircularGaugePointers>
<CircularGaugePointer Type="PointerType.Marker" Value="12"
MarkerShape="GaugeShape.Image" ImageUrl="src/circular-
gauge/images/football.png" Radius="100%" MarkerWidth="28" MarkerHeight="28">
<CircularGaugePointerAnimation Enable="true" Duration="1500" />
</CircularGaugePointer>
<CircularGaugePointer Type="PointerType.Marker" Value="11"
MarkerShape="GaugeShape.Image" ImageUrl="src/circular-
gauge/images/basketball.png" Radius="70%" MarkerWidth="28"
MarkerHeight="28">
<CircularGaugePointerAnimation Enable="true" Duration="1200" />
</CircularGaugePointer>
<CircularGaugePointer Type="PointerType.Marker" Value="10"
MarkerShape="GaugeShape.Image" ImageUrl="src/circular-
gauge/images/golfball.png" Radius="40%" MarkerWidth="28" MarkerHeight="28">
<CircularGaugePointerAnimation Enable="true" Duration="900" />
</CircularGaugePointer>
<CircularGaugePointer Type="PointerType.Marker" Value="12"
MarkerShape="GaugeShape.Image" ImageUrl="src/circular-
gauge/images/athletics.png" Radius="0%" MarkerWidth="90" MarkerHeight="90">
<CircularGaugePointerAnimation Enable="true" Duration="0" />
</CircularGaugePointer>
<CircularGaugePointer Type="PointerType.Marker" Value="0"
MarkerShape="GaugeShape.Image" ImageUrl="src/circular-
gauge/images/girl1.png" Radius="100%" MarkerWidth="28" MarkerHeight="28">
<CircularGaugePointerAnimation Enable="true" Duration="1500" />
</CircularGaugePointer>
<CircularGaugePointer Type="PointerType.Marker" Value="0"
MarkerShape="GaugeShape.Image" ImageUrl="src/circular-gauge/images/man1.png"
Radius="70%" MarkerWidth="28" MarkerHeight="28">
<CircularGaugePointerAnimation Enable="true" Duration="1500" />
</CircularGaugePointer>
```

```

<CircularGaugePointer Type="PointerType.Marker" Value="0"
MarkerShape="GaugeShape.Image" ImageUrl="src/circular-gauge/images/man2.png"
Radius="40%" MarkerWidth="28" MarkerHeight="28">
<CircularGaugePointerAnimation Enable="true" Duration="1500" />
</CircularGaugePointer>
</CircularGaugePointers>
<CircularGaugeRanges>
<CircularGaugeRange Start="0" End="12" Radius="105%" Color="#01aebe"
StartWidth="25" EndWidth="25">
<LinearGradient StartValue="1%" EndValue="99%">
<ColorStops>
<ColorStop Opacity="0.9" Offset="0%" Color="#fef3f9"></ColorStop>
<ColorStop Opacity="0.9" Offset="100%" Color="#f54ea2"></ColorStop>
</ColorStops>
</LinearGradient>
</CircularGaugeRange>
<CircularGaugeRange Start="0" End="11" Radius="75%" Color="#3bceac"
StartWidth="25" EndWidth="25">
<LinearGradient StartValue="1%" EndValue="99%">
<ColorStops>
<ColorStop Opacity="0.9" Offset="0%" Color="#fef3f9"></ColorStop>
<ColorStop Opacity="0.9" Offset="100%" Color="#f54ea2"></ColorStop>
</ColorStops>
</LinearGradient>
</CircularGaugeRange>
<CircularGaugeRange Start="0" End="10" Radius="45%" Color="#ee4266"
StartWidth="25" EndWidth="25">
<LinearGradient StartValue="1%" EndValue="99%">
<ColorStops>
<ColorStop Opacity="0.9" Offset="0%" Color="#fef3f9"></ColorStop>
<ColorStop Opacity="0.9" Offset="100%" Color="#f54ea2"></ColorStop>
</ColorStops>
</LinearGradient>
</CircularGaugeRange>
</CircularGaugeRanges>
<CircularGaugeAnnotations>
<CircularGaugeAnnotation Content="12 M" Radius="105%" Angle="95" ZIndex="1"
/>
<CircularGaugeAnnotation Content="11 M" Radius="77%" Angle="78" ZIndex="1"
/>
<CircularGaugeAnnotation Content="10 M" Radius="45%" Angle="65" ZIndex="1"
/>
<CircularGaugeAnnotation Radius="108%" Angle="190" ZIndex="1">
<ContentTemplate>
<div class="annotationText"><span class="templateAlign">Doe</span></div>
</ContentTemplate>
</CircularGaugeAnnotation>
<CircularGaugeAnnotation Radius="80%" Angle="185" ZIndex="1">
<ContentTemplate>
<div class="annotationText"><span class="templateAlign">Almada</span></div>
</ContentTemplate>
</CircularGaugeAnnotation>
<CircularGaugeAnnotation Radius="50%" Angle="180" ZIndex="1">
<ContentTemplate>
<div class="annotationText"><span class="templateAlign">John</span></div>
</ContentTemplate>
</CircularGaugeAnnotation>

```

```

</CircularGaugeAnnotations>
</CircularGaugeAxis>
</CircularGaugeAxes>
</SfCircularGauge>
<style>
.templateAlign {
font-size: 14px;
color: #9E9E9E;
font-family: Regular;
margin-left: -20px;
}
.annotationText {
margin-top: -30px;
}
</style>

```

Radial Gradient

Using radial gradient, colors will be applied in circular progression. The inner circle position of the radial gradient will be set using the [InnerPosition](#) property. The outer circle position of the radial gradient can be set using the [OuterPosition](#) property. The color stop values such as color, opacity and offset are set using [ColorStop](#) property.

To apply radial gradient to the range, follow the below code sample.

ASPX-CS

```

@using Syncfusion.Blazor.CircularGauge
<SfCircularGauge CenterY="57%" Title="Shot Put Distance">
<CircularGaugeAxes>
<CircularGaugeAxis StartAngle="200" EndAngle="130" Minimum="0" Maximum="14"
Radius="80%">
<CircularGaugeAxisLineStyle Width="0.001" />
<CircularGaugeAxisMajorTicks Width="0.01" />
<CircularGaugeAxisMinorTicks Width="0.01" />
<CircularGaugeAxisLabelStyle>
<CircularGaugeAxisLabelFont Size="0px" />
</CircularGaugeAxisLabelStyle>
<CircularGaugePointers>
<CircularGaugePointer Type="PointerType.Marker" Value="12"
MarkerShape="GaugeShape.Image" ImageUrl="src/circular-
gauge/images/football.png" Radius="100%" MarkerWidth="28" MarkerHeight="28">
<CircularGaugePointerAnimation Enable="true" Duration="1500" />
</CircularGaugePointer>
<CircularGaugePointer Type="PointerType.Marker" Value="11"
MarkerShape="GaugeShape.Image" ImageUrl="src/circular-
gauge/images/basketball.png" Radius="70%" MarkerWidth="28"
MarkerHeight="28">
<CircularGaugePointerAnimation Enable="true" Duration="1200" />
</CircularGaugePointer>
<CircularGaugePointer Type="PointerType.Marker" Value="10"
MarkerShape="GaugeShape.Image" ImageUrl="src/circular-
gauge/cimages/golfball.png" Radius="40%" MarkerWidth="28" MarkerHeight="28">
<CircularGaugePointerAnimation Enable="true" Duration="900" />
</CircularGaugePointer>

```

```

<CircularGaugePointer Type="PointerType.Marker" Value="12"
MarkerShape="GaugeShape.Image" ImageUrl="src/circular-
gauge/images/athletics.png" Radius="0%" MarkerWidth="90" MarkerHeight="90">
<CircularGaugePointerAnimation Enable="true" Duration="0" />
</CircularGaugePointer>
<CircularGaugePointer Type="PointerType.Marker" Value="0"
MarkerShape="GaugeShape.Image" ImageUrl="src/circular-
gauge/images/girl1.png" Radius="100%" MarkerWidth="28" MarkerHeight="28">
<CircularGaugePointerAnimation Enable="true" Duration="1500" />
</CircularGaugePointer>
<CircularGaugePointer Type="PointerType.Marker" Value="0"
MarkerShape="GaugeShape.Image" ImageUrl="src/circular-gauge/images/man1.png"
Radius="70%" MarkerWidth="28" MarkerHeight="28">
<CircularGaugePointerAnimation Enable="true" Duration="1500" />
</CircularGaugePointer>
<CircularGaugePointer Type="PointerType.Marker" Value="0"
MarkerShape="GaugeShape.Image" ImageUrl="src/circular-gauge/images/man2.png"
Radius="40%" MarkerWidth="28" MarkerHeight="28">
<CircularGaugePointerAnimation Enable="true" Duration="1500" />
</CircularGaugePointer>
</CircularGaugePointers>
<CircularGaugeRanges>
<CircularGaugeRange Start="0" End="12" Radius="105%" Color="#01aebe"
StartWidth="25" EndWidth="25">
<RadialGradient Radius="65%">
<InnerPosition X="60%" Y="60%"></InnerPosition>
<OuterPosition X="50%" Y="70%"></OuterPosition>
<ColorStops>
<ColorStop Opacity="0.9" Offset="5%" Color="#fff5f5"></ColorStop>
<ColorStop Opacity="0.9" Offset="99%" Color="#f54ea2"></ColorStop>
</ColorStops>
</RadialGradient>
</CircularGaugeRange>
<CircularGaugeRange Start="0" End="11" Radius="75%" Color="#3bceac"
StartWidth="25" EndWidth="25">
<RadialGradient Radius="65%">
<InnerPosition X="60%" Y="60%"></InnerPosition>
<OuterPosition X="50%" Y="70%"></OuterPosition>
<ColorStops>
<ColorStop Opacity="0.9" Offset="5%" Color="#fff5f5"></ColorStop>
<ColorStop Opacity="0.9" Offset="99%" Color="#f54ea2"></ColorStop>
</ColorStops>
</RadialGradient>
</CircularGaugeRange>
<CircularGaugeRange Start="0" End="10" Radius="45%" Color="#ee4266"
StartWidth="25" EndWidth="25">
<RadialGradient Radius="65%">
<InnerPosition X="60%" Y="60%"></InnerPosition>
<OuterPosition X="50%" Y="70%"></OuterPosition>
<ColorStops>
<ColorStop Opacity="0.9" Offset="5%" Color="#fff5f5"></ColorStop>
<ColorStop Opacity="0.9" Offset="99%" Color="#f54ea2"></ColorStop>
</ColorStops>
</RadialGradient>
</CircularGaugeRange>
</CircularGaugeRanges>
<CircularGaugeAnnotations>

```

```

<CircularGaugeAnnotation Content="12 M" Radius="105%" Angle="95" ZIndex="1"
/>
<CircularGaugeAnnotation Content="11 M" Radius="77%" Angle="78" ZIndex="1"
/>
<CircularGaugeAnnotation Content="10 M" Radius="45%" Angle="65" ZIndex="1"
/>
<CircularGaugeAnnotation Radius="108%" Angle="190" ZIndex="1">
<ContentTemplate>
<div class="annotationText"><span class="templateAlign">Doe</span></div>
</ContentTemplate>
</CircularGaugeAnnotation>
<CircularGaugeAnnotation Radius="80%" Angle="185" ZIndex="1">
<ContentTemplate>
<div class="annotationText"><span class="templateAlign">Almaida</span></div>
</ContentTemplate>
</CircularGaugeAnnotation>
<CircularGaugeAnnotation Radius="50%" Angle="180" ZIndex="1">
<ContentTemplate>
<div class="annotationText"><span class="templateAlign">John</span></div>
</ContentTemplate>
</CircularGaugeAnnotation>
</CircularGaugeAnnotations>
</CircularGaugeAxis>
</CircularGaugeAxes>
</SfCircularGauge>
<style>
.templateAlign {
font-size: 14px;
color: #9E9E9E;
font-family: Regular;
margin-left: -20px;
}
.annotationText {
margin-top: -30px;
}
</style>

```

See also

- [Tooltip for Ranges](#)

Pointers in Blazor Circular Gauge Component

Pointers are used to indicate values on an axis. The value of a pointer can be modified using the [Value](#) property.

ASPX-CS

```

@using Syncfusion.Blazor.CircularGauge
<SfCircularGauge>
<CircularGaugeAxes>
<CircularGaugeAxis>
<CircularGaugePointers>
<CircularGaugePointer Value="90">
</CircularGaugePointer>
</CircularGaugePointers>

```



```
</CircularGaugeAxis>  
</CircularGaugeAxes>  
</SfCircularGauge>
```



The Circular Gauge supports three types of pointers such as [Needle](#), [RangeBar](#), and [Marker](#). You can choose any pointer using the [Type](#) property.

Needle pointer

The circular gauge's default pointer type will be needle. A needle point contains three parts, a needle, a cap/knob, and a tail.

ASPX-CS

```
@using Syncfusion.Blazor.CircularGauge  
<SfCircularGauge>  
<CircularGaugeAxes>  
<CircularGaugeAxis>  
<CircularGaugePointers>  
<CircularGaugePointer Value="90">  
</CircularGaugePointer>  
</CircularGaugePointers>  
</CircularGaugeAxis>  
</CircularGaugeAxes>  
</SfCircularGauge>
```



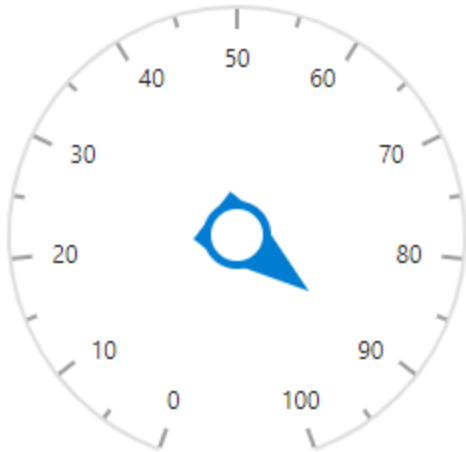
Customization

The needle, tail and cap of the pointer can be customized with the following properties.

- [CircularGaugePointer](#) - Customize the pointer's needle.
- [Radius](#) - sets the needle length.
- [PointerWidth](#) - sets the needle width.
- [Color](#) - sets the needle color.
- [CircularGaugeNeedleTail](#) - Customize the pointer's tail.
- [Length](#) - sets pointer's tail length.
- [Color](#) - sets pointer's tail color.
- [CircularGaugeNeedleTailBorder](#) - Sets pointer's tail border.
- [CircularGaugeCap](#) - Customize the pointer's cap.
- [Color](#) - sets pointer's cap color.
- [Radius](#) - sets pointer's cap radius.
- [CircularGaugeCapBorder](#) - sets pointer's cap border.
- `[Position]` - specifies the position of the Pointer. Its possible values are 'PointerRangePosition.Inside', 'PointerRangePosition.Outside' and 'PointerRangePosition.Cross'.

ASPX-CS

```
@using Syncfusion.Blazor.CircularGauge
<SfCircularGauge Height="250px" Width="250px">
  <CircularGaugeAxes>
    <CircularGaugeAxis>
      <CircularGaugePointers>
        <CircularGaugePointer Value="90"
          Radius="40%"
          PointerWidth="30"
          Color="#007DD1"
          Position="PointerRangePosition.Inside">
          <CircularGaugeCap Radius="15"
            Color="white">
            <CircularGaugeCapBorder Width="4"
              Color="#007DD1">
            </CircularGaugeCapBorder>
          </CircularGaugeCap>
          <CircularGaugeNeedleTail Length="35%"
            Color="#007DD1">
          </CircularGaugeNeedleTail>
        </CircularGaugePointer>
      </CircularGaugePointers>
    </CircularGaugeAxis>
  </CircularGaugeAxes>
</SfCircularGauge>
```



<!-- markdownlint-disable MD010 -->

The appearance of the needle pointer can be customized by using [NeedleStartWidth](#) and [NeedleEndWidth](#).

ASPX-CS

```
@using Syncfusion.Blazor.CircularGauge
<SfCircularGauge>
  <CircularGaugeAxes>
    <CircularGaugeAxis StartAngle="270" EndAngle="90" Radius="90%" Minimum="0"
Maximum="100">
      <CircularGaugeAxisLineStyle Width="3" Color="#1E7145">
      </CircularGaugeAxisLineStyle>
      <CircularGaugeAxisLabelStyle>
        <CircularGaugeAxisLabelFont Color="#1E7145" Size="0px">
        </CircularGaugeAxisLabelFont>
        </CircularGaugeAxisLabelStyle>
        <CircularGaugeAxisMajorTicks Interval="100"
Height="0"
Width="1">
        </CircularGaugeAxisMajorTicks>
        <CircularGaugeAxisMinorTicks Width="0"
Height="0"
>
        </CircularGaugeAxisMinorTicks>
      <CircularGaugePointers>
        <CircularGaugePointer Value="70"
Radius="80%" Color="green" PointerWidth="2" NeedleStartWidth="4"
NeedleEndWidth="4">
          <CircularGaugeCap Radius="8" Color="green">
          </CircularGaugeCap>
          <CircularGaugeNeedleTail Length="0%">
          </CircularGaugeNeedleTail>
        </CircularGaugePointer>
      </CircularGaugePointers>
      <CircularGaugeAnnotations>
        <CircularGaugeAnnotation Angle="180" ZIndex="1">
        <ContentTemplate>
          <div class="custom-annotation">Customized Needle</div>
```

```

</ContentTemplate>
</CircularGaugeAnnotation>
</CircularGaugeAnnotations>
</CircularGaugeAxis>
</CircularGaugeAxes>
</SfCircularGauge>
<style type="text/css">
.custom-annotation {
color: #757575;
font-family:Roboto; font-size:14px;padding-top: 26px;
}
</style>

```

Range bar pointer

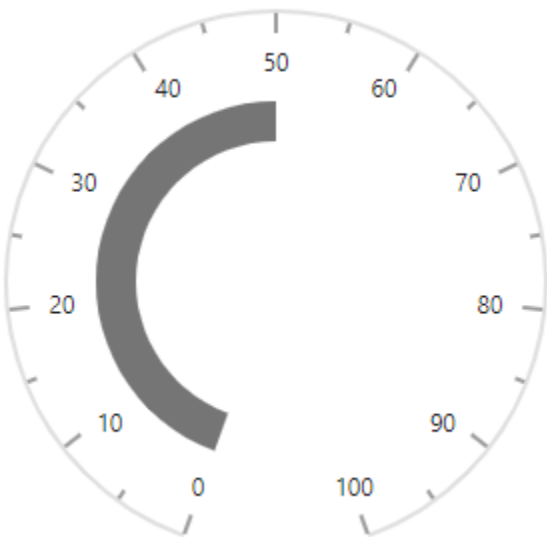
The range bar pointer is like a range in an axis that can be placed on gauge to mark the pointer value. The range bar starts from the beginning of the gauge and ends at the pointer value. You can set the pointer type using `Type` property in `CircularGaugePointer`.

ASPX-CS

```

@using Syncfusion.Blazor.CircularGauge
<SfCircularGauge>
<CircularGaugeAxes>
<CircularGaugeAxis>
<CircularGaugePointers>
<CircularGaugePointer Value="50"
Type="PointerType.RangeBar">
</CircularGaugePointer>
</CircularGaugePointers>
</CircularGaugeAxis>
</CircularGaugeAxes>
</SfCircularGauge>

```



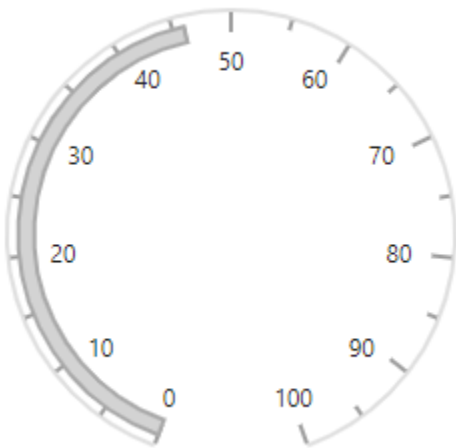
Customization

You can customize the range bar using the following properties.

- [PointerWidth](#) - Specifies the width of the range bar.
- [Color](#) - Specifies the color of the range bar.
- [Radius](#) - Specifies the range bar radius
- [RoundedCornerRadius](#) - Specifies the rounded corner of the range bar.
- [CircularGaugePointerBorder](#) - Specifies the border of the range bar.

ASPX-CS

```
@using Syncfusion.Blazor.CircularGauge
<SfCircularGauge>
  <CircularGaugeAxes>
    <CircularGaugeAxis>
      <CircularGaugePointers>
        <CircularGaugePointer Value="46"
          Type="PointerType.RangeBar"
          PointerWidth="8"
          Radius="95%"
          Color="lightgray">
          <CircularGaugePointerBorder Color="darkgray"
            Width="2">
          </CircularGaugePointerBorder>
        </CircularGaugePointer>
      </CircularGaugePointers>
    </CircularGaugeAxis>
  </CircularGaugeAxes>
</SfCircularGauge>
```



Rounded corners

The start and end pointers of a range bar in the Circular Gauge are rounded to form arc using the `RoundedCornerRadius` property.

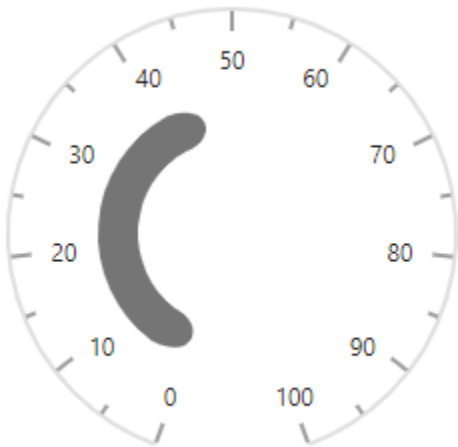
ASPX-CS

```
@using Syncfusion.Blazor.CircularGauge
<SfCircularGauge>
  <CircularGaugeAxes>
    <CircularGaugeAxis>
      <CircularGaugePointers>
```

```

<CircularGaugePointer Value="46"
RoundedCornerRadius="20"
Type="PointerType.RangeBar">
</CircularGaugePointer>
</CircularGaugePointers>
</CircularGaugeAxis>
</CircularGaugeAxes>
</SfCircularGauge>

```



Marker pointer

The different types of marker shapes can be used to mark the pointer value in an axis. You can change the marker shape using the [MarkerShape](#) property in pointer. The Circular Gauge supports the following marker shapes:

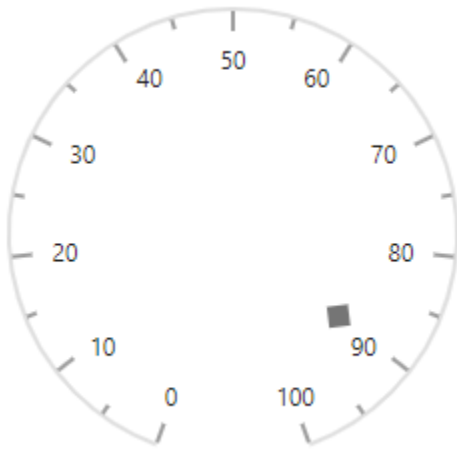
- Circle
- Rectangle
- Triangle
- InvertedTriangle
- Diamond

ASPX-CS

```

@using Syncfusion.Blazor.CircularGauge
<SfCircularGauge>
<CircularGaugeAxes>
<CircularGaugeAxis>
<CircularGaugePointers>
<CircularGaugePointer Value="90"
Type="PointerType.Marker"
MarkerShape="GaugeShape.Diamond"
MarkerHeight="15"
MarkerWidth="15">
</CircularGaugePointer>
</CircularGaugePointers>
</CircularGaugeAxis>
</CircularGaugeAxes>
</SfCircularGauge>

```



Customization

You can customize the marker pointer using the following properties.

- [Color](#) - specifies marker pointer color.
- [MarkerWidth](#) - specifies marker pointer width.
- [MarkerHeight](#) - specifies marker pointer height.
- [Radius](#) - specifies marker pointer radius.
- [CircularGaugePointerBorder](#) - specifies marker pointer's border color and width.

ASPX-CS

```
@using Syncfusion.Blazor.CircularGauge
<SfCircularGauge>
  <CircularGaugeAxes>
    <CircularGaugeAxis>
      <CircularGaugePointers>
        <CircularGaugePointer Value="90"
          Type="PointerType.Marker"
          MarkerShape="GaugeShape.InvertedTriangle"
          MarkerHeight="15"
          MarkerWidth="15"
          Color="white"
          Radius="110%">
          <CircularGaugePointerBorder Width="2"
            Color="#007DD1"></CircularGaugePointerBorder>
        </CircularGaugePointer>
      </CircularGaugePointers>
    </CircularGaugeAxis>
  </CircularGaugeAxes>
</SfCircularGauge>
```

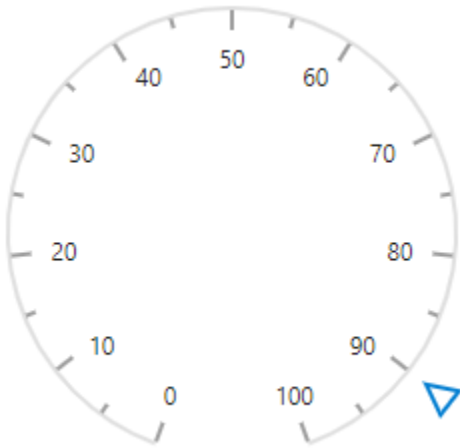
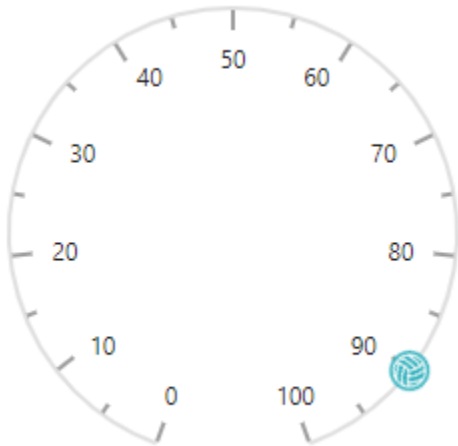


Image marker pointer

You can use image instead of rendering marker shape to denote the pointer value. It can be achieved by setting [MarkerShape](#) to 'GaugeShape.Image' and assigning image path to [ImageUrl](#) as shown in following example.

ASPX-CS

```
@using Syncfusion.Blazor.CircularGauge
<SfCircularGauge>
  <CircularGaugeAxes>
    <CircularGaugeAxis>
      <CircularGaugePointers>
        <CircularGaugePointer Value="90"
          Type="PointerType.Marker"
          MarkerShape="GaugeShape.Image"
          ImageUrl="football.png"
          MarkerHeight="20"
          MarkerWidth="20"
          Radius="100%">
        </CircularGaugePointer>
      </CircularGaugePointers>
    </CircularGaugeAxis>
  </CircularGaugeAxes>
</SfCircularGauge>
```

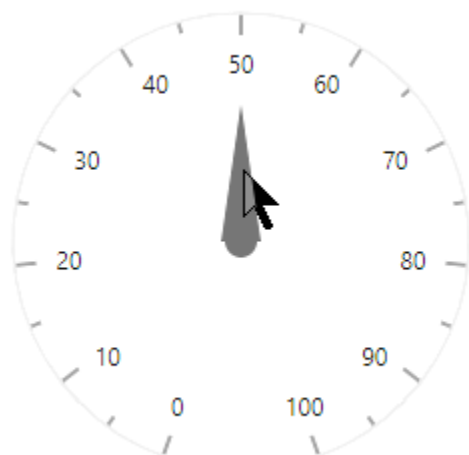
<!-- markdownlint-disable MD010 -->

Dragging pointer

The pointers can be dragged over the axis values by clicking and dragging the same. To enable or disable the pointer drag, use the [EnablePointerDrag](#) property.

ASPX-CS

```
@using Syncfusion.Blazor.CircularGauge
<SfCircularGauge EnablePointerDrag="true" Height="250px" Width="250px">
  <CircularGaugeAxes>
    <CircularGaugeAxis>
      <CircularGaugePointers>
        <CircularGaugePointer Value="50">
        </CircularGaugePointer>
      </CircularGaugePointers>
    </CircularGaugeAxis>
  </CircularGaugeAxes>
</SfCircularGauge>
```

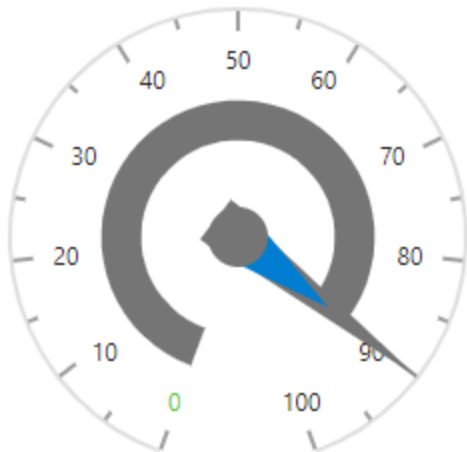


Multiple pointers

In addition to the default pointer, you can add n number of pointers to an axis using the [CircularGaugePointers](#) tag.

ASPX-CS

```
@using Syncfusion.Blazor.CircularGauge
<SfCircularGauge>
  <CircularGaugeAxes>
    <CircularGaugeAxis>
      <CircularGaugePointers>
        <CircularGaugePointer Value="90"
MarkerShape="GaugeShape.InvertedTriangle"
Radius="100%"
MarkerHeight="15"
MarkerWidth="15">
      </CircularGaugePointer>
        <CircularGaugePointer Value="90"
Type="PointerType.RangeBar"
Radius="60%"
MarkerWidth="10">
      </CircularGaugePointer>
        <CircularGaugePointer Value="90"
Radius="50%"
PointerWidth="25"
Color="#007DD1">
      <CircularGaugeCap Radius="15">
        <CircularGaugeCapBorder Width="5">
      </CircularGaugeCapBorder>
      </CircularGaugeCap>
        <CircularGaugeNeedleTail Length="25%">
      </CircularGaugeNeedleTail>
      </CircularGaugePointer>
    </CircularGaugePointers>
    <CircularGaugeAxisLabelStyle UseRangeColor="true">
    </CircularGaugeAxisLabelStyle>
    <CircularGaugeAxisMinorTicks
UseRangeColor="true"></CircularGaugeAxisMinorTicks>
    </CircularGaugeAxis>
  </CircularGaugeAxes>
</SfCircularGauge>
```



Pointer animation

The pointers are animated on loading the gauge using the [CircularGaugePointerAnimation](#) tag in pointer. The [Enable](#) property in animation allows you to enable or disable the animation. The [Duration](#) property specifies the duration of the animation in milliseconds.

ASPX-CS

```
@using Syncfusion.Blazor.CircularGauge
<SfCircularGauge Height="250" Width="250">
  <CircularGaugeAxes>
    <CircularGaugeAxis>
      <CircularGaugePointers>
        <CircularGaugePointer Value="70">
          <CircularGaugePointerAnimation Enable="true" Duration="1500">
          </CircularGaugePointerAnimation>
        </CircularGaugePointer>
      </CircularGaugePointers>
    </CircularGaugeAxis>
  </CircularGaugeAxes>
</SfCircularGauge>
```



Gradient Color

Gradient support allows to add multiple colors in the ranges and pointers of the circular gauge. The following gradient types are supported in the circular gauge.

- Linear Gradient
- Radial Gradient

Linear Gradient

Using linear gradient, colors will be applied in a linear progression. The start value of the linear gradient can be set using the [StartValue](#) property. The end value of the linear gradient will be set using the [EndValue](#) property. The color stop values such as color, opacity and offset are set using [ColorStop](#) property.

The linear gradient can be applied to all pointer types like marker, range bar, and needle. To do so, follow the below code sample.

ASPX-CS

```
@using Syncfusion.Blazor.CircularGauge
<SfCircularGauge>
<CircularGaugeAxes>
<CircularGaugeAxis StartAngle="270" EndAngle="90" Radius="90%" Minimum="0"
Maximum="100">
<CircularGaugeAxisLineStyle Width="3" Color="#E63B86" />
<CircularGaugeAxisLabelStyle>
<CircularGaugeAxisLabelFont Size="0px" />
</CircularGaugeAxisLabelStyle>
<CircularGaugeAxisMajorTicks Height="0.01" />
<CircularGaugeAxisMinorTicks Height="0.01" />
<CircularGaugePointers>
<CircularGaugePointer Value="80" Radius="80%" PointerWidth="10">
<LinearGradient StartValue="1%" EndValue="99%">
<ColorStops>
<ColorStop Opacity="0.9" Offset="0%" Color="#fef3f9"></ColorStop>
<ColorStop Opacity="0.9" Offset="100%" Color="#f54ea2"></ColorStop>
</ColorStops>
</LinearGradient>
<CircularGaugeCap Radius="8" Color="White">
<CircularGaugeCapBorder Color="#E63B86" Width="1" />
</CircularGaugeCap>
<CircularGaugeNeedleTail Length="20%">
<LinearGradient StartValue="1%" EndValue="99%">
<ColorStops>
<ColorStop Opacity="0.9" Offset="0%" Color="#fef3f9"></ColorStop>
<ColorStop Opacity="0.9" Offset="100%" Color="#f54ea2"></ColorStop>
</ColorStops>
</LinearGradient>
</CircularGaugeNeedleTail>
<CircularGaugePointerAnimation Enable="true" Duration="1000" />
</CircularGaugePointer>
<CircularGaugePointer Value="40" Radius="60%" MarkerWidth="5"
MarkerHeight="5" PointerWidth="10">
<LinearGradient StartValue="1%" EndValue="99%">
<ColorStops>
<ColorStop Opacity="0.9" Offset="0%" Color="#fef3f9"></ColorStop>
```

```

<ColorStop Opacity="0.9" Offset="100%" Color="#f54ea2"></ColorStop>
</ColorStops>
</LinearGradient>
<CircularGaugeCap Radius="8" Color="White">
<CircularGaugeCapBorder Color="#E63B86" Width="1" />
</CircularGaugeCap>
<CircularGaugeNeedleTail Length="20%">
<LinearGradient StartValue="1%" EndValue="99%">
<ColorStops>
<ColorStop Opacity="0.9" Offset="0%" Color="#fef3f9"></ColorStop>
<ColorStop Opacity="0.9" Offset="100%" Color="#f54ea2"></ColorStop>
</ColorStops>
</LinearGradient>
</CircularGaugeNeedleTail>
<CircularGaugePointerAnimation Enable="true" Duration="1000" />
</CircularGaugePointer>
</CircularGaugePointers>
</CircularGaugeAxis>
</CircularGaugeAxes>
</SfCircularGauge>

```

Radial Gradient

Using radial gradient, colors will be applied in circular progression. The inner circle position of the radial gradient will be set using the [InnerPosition](#) property. The outer circle position of the radial gradient can be set using the [OuterPosition](#) property. The color stop values such as color, opacity and offset are set using [ColorStop](#) property.

The radial gradient can be applied to all pointer types like marker, range bar, and needle. To do so, follow the below code sample.

ASPX-CS

```

@using Syncfusion.Blazor.CircularGauge
<SfCircularGauge>
<CircularGaugeAxes>
<CircularGaugeAxis StartAngle="270" EndAngle="90" Radius="90%" Minimum="0"
Maximum="100">
<CircularGaugeAxisLineStyle Width="3" Color="#E63B86" />
<CircularGaugeAxisLabelStyle>
<CircularGaugeAxisLabelFont Size="0px" />
</CircularGaugeAxisLabelStyle>
<CircularGaugeAxisMajorTicks Height="0.01" />
<CircularGaugeAxisMinorTicks Height="0.01" />
</CircularGaugeAxis>
<CircularGaugePointers>
<CircularGaugePointer Value="80" Radius="80%" PointerWidth="10">
<RadialGradient Radius="65%">
<InnerPosition X="60%" Y="60%"></InnerPosition>
<OuterPosition X="50%" Y="70%"></OuterPosition>
<ColorStops>
<ColorStop Opacity="0.9" Offset="5%" Color="#fff5f5"></ColorStop>
<ColorStop Opacity="0.9" Offset="99%" Color="#f54ea2"></ColorStop>
</ColorStops>
</RadialGradient>
<CircularGaugeCap Radius="8" Color="White">
<CircularGaugeCapBorder Color="#E63B86" Width="1" />
</CircularGaugeCap>

```

```

<CircularGaugeNeedleTail Length="20%">
  <RadialGradient Radius="65%">
    <InnerPosition X="60%" Y="60%"></InnerPosition>
    <OuterPosition X="50%" Y="70%"></OuterPosition>
    <ColorStops>
      <ColorStop Opacity="0.9" Offset="5%" Color="#fff5f5"></ColorStop>
      <ColorStop Opacity="0.9" Offset="99%" Color="#f54ea2"></ColorStop>
    </ColorStops>
  </RadialGradient>
</CircularGaugeNeedleTail>
<CircularGaugePointerAnimation Enable="true" Duration="1000" />
<CircularGaugePointer>
  <CircularGaugePointer Value="40" Radius="60%" MarkerWidth="5"
  MarkerHeight="5" PointerWidth="10">
    <RadialGradient Radius="65%">
      <InnerPosition X="60%" Y="60%"></InnerPosition>
      <OuterPosition X="50%" Y="70%"></OuterPosition>
      <ColorStops>
        <ColorStop Opacity="0.9" Offset="5%" Color="#fff5f5"></ColorStop>
        <ColorStop Opacity="0.9" Offset="99%" Color="#f54ea2"></ColorStop>
      </ColorStops>
    </RadialGradient>
  <CircularGaugeCap Radius="8" Color="White">
    <CircularGaugeCapBorder Color="#E63B86" Width="1" />
  </CircularGaugeCap>
  <CircularGaugeNeedleTail Length="20%">
    <RadialGradient Radius="65%">
      <InnerPosition X="60%" Y="60%"></InnerPosition>
      <OuterPosition X="50%" Y="70%"></OuterPosition>
      <ColorStops>
        <ColorStop Opacity="0.9" Offset="5%" Color="#fff5f5"></ColorStop>
        <ColorStop Opacity="0.9" Offset="99%" Color="#f54ea2"></ColorStop>
      </ColorStops>
    </RadialGradient>
  </CircularGaugeNeedleTail>
  <CircularGaugePointerAnimation Enable="true" Duration="1000" />
</CircularGaugePointer>
</CircularGaugePointers>
</CircularGaugeAxis>
</CircularGaugeAxes>
</SfCircularGauge>

```

<!-- markdownlint-disable MD010 -->

Annotations in Blazor Circular Gauge Component

Annotations are used to mark a specific area of interest in the Circular Gauge with texts, shapes, or images.

Customization

You can place any custom element on the axis area using [ContentTemplate](#) in the [CircularGaugeAnnotation](#).

ASPX-CS

```

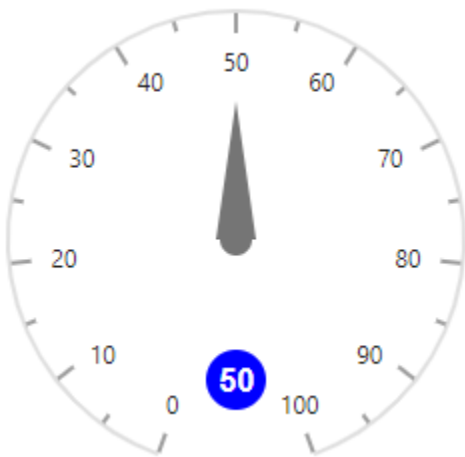
@using Syncfusion.Blazor.CircularGauge
<SfCircularGauge Height="250px" Width="250px">

```

```

<CircularGaugeAxes>
<CircularGaugeAxis>
<CircularGaugePointers>
<CircularGaugePointer Value="50"></CircularGaugePointer>
</CircularGaugePointers>
<CircularGaugeAnnotations>
<CircularGaugeAnnotation Angle="195" ZIndex="1">
<ContentTemplate>
<div class="custom-annotation">50</div>
</ContentTemplate>
</CircularGaugeAnnotation>
</CircularGaugeAnnotations>
</CircularGaugeAxis>
</CircularGaugeAxes>
</SfCircularGauge>
<style type="text/css">
.custom-annotation {
color: white;
background-color: blue;
height: 30px;
width: 30px;
border-radius: 15px;
padding: 4px 0 0 6px;
font-weight: bold;
}
</style>

```



Positioning the annotation

Annotations can be placed around an axis using the [Radius](#) and [Angle](#) properties. For example, if the angle is 90 degrees and the radius is 110%, then the annotation will be placed at the right of the axis.

The radius of an annotation takes values either in pixel or in percentage. By setting value in percentage, annotation gets its position with respect to its axis radius.

ASPX-CS

```

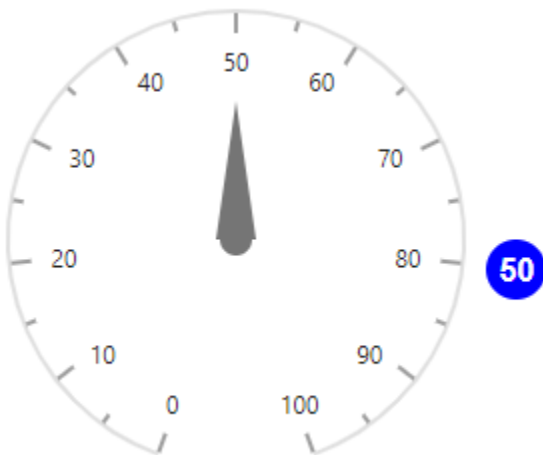
@using Syncfusion.Blazor.CircularGauge
<SfCircularGauge Height="250px" Width="250px">
<CircularGaugeAxes>
<CircularGaugeAxis>

```

```

<CircularGaugePointers>
<CircularGaugePointer Value="50"></CircularGaugePointer>
</CircularGaugePointers>
<CircularGaugeAnnotations>
<CircularGaugeAnnotation Angle="90"
Radius="110%"
ZIndex="1">
<ContentTemplate>
<div class="custom-annotation">50</div>
</ContentTemplate>
</CircularGaugeAnnotation>
</CircularGaugeAnnotations>
</CircularGaugeAxis>
</CircularGaugeAxes>
</SfCircularGauge>
<style type="text/css">
.custom-annotation {
color: white;
background-color: blue;
height: 30px;
width: 30px;
border-radius: 15px;
padding: 4px 0 0 6px;
font-weight: bold;
}
</style>

```



Multiple annotations

Using [CircularGaugeAnnotation](#), you can add multiple annotations to the circular gauge and each annotation content can be customized separately.

ASPX-CS

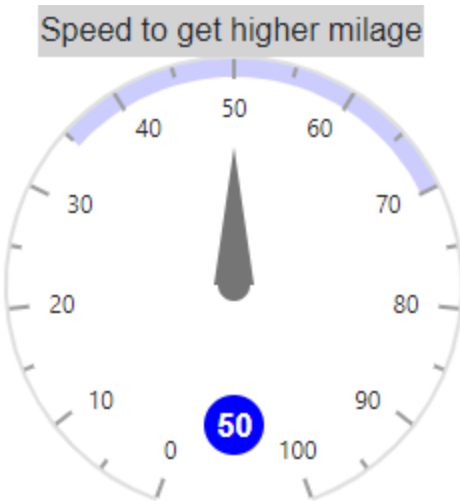
```

@using Syncfusion.Blazor.CircularGauge
<SfCircularGauge Height="250px" Width="250px">
<CircularGaugeAxes>
<CircularGaugeAxis>
<CircularGaugeRanges>
<CircularGaugeRange Start="35"

```



```
End="70"
Color="blue"
Opacity="0.2">
</CircularGaugeRange>
</CircularGaugeRanges>
<CircularGaugePointers>
<CircularGaugePointer Value="50"></CircularGaugePointer>
</CircularGaugePointers>
<CircularGaugeAnnotations>
<CircularGaugeAnnotation Angle="325" Radius="150%" ZIndex="1">
<ContentTemplate>
<div class="custom-annotation">Speed to get higher milage</div>
</ContentTemplate>
</CircularGaugeAnnotation>
<CircularGaugeAnnotation Angle="195" ZIndex="1">
<ContentTemplate>
<div class="speed">50</div>
</ContentTemplate>
</CircularGaugeAnnotation>
</CircularGaugeAnnotations>
</CircularGaugeAxis>
</CircularGaugeAxes>
</SfCircularGauge>
<style type="text/css">
.speed {
color: white;
background-color: blue;
height: 30px;
width: 30px;
border-radius: 15px;
padding: 4px 0 0 6px;
font-weight: bold;
}
.custom-annotation {
background-color: lightgray;
width: 100%;
padding: 1px;
}
</style>
```



See also

- [Tooltip for Annotation](#)

Legend in Blazor Circular Gauge Component

Legend provides valuable information for interpreting what the circular gauge axis range displays, and they can be represented in various colors, shapes, and other identifiers based on the data. It gives a breakdown of what each symbol represents in the axis range of circular gauge.

You can add the legend for circular gauge ranges by setting the visible property of `CircularGaugeLegendSettings` to true.

<!-- markdownlint-disable MD036 -->

Legend customization

Customization option is also provided for the legend shape, alignment, and position.

Position and alignment

The position of the legend is used to place legend in various positions. You can use the `Position` property in `CircularGaugeLegendSettings`. Based on the position, the legend item will be aligned. The following options are available to customize the legend position:

- Top
- Bottom
- Left
- Right
- Custom
- Auto

The legend alignment is used to align the legend items in specific location. You can use the alignment property in `CircularGaugeLegendSettings` to align the legend items. The following options are available to customize the legend alignment:

- Near

- Center
- Far

The legends can also be positioned to absolute position using the `Location` properties available in `legendSettings`.

Legend size

The legend size can be modified using the `Height` and `Width` properties in `CircularGaugeLegendSettings`.

Legend opacity

To specify the transparency for legend shape, set the `Opacity` property in `CircularGaugeLegendSettings`.

Legend shape

To change the legend item shape, specify the desired `Shape` in the shape property of the legend. By default, the shape of the legend is `Circle`.

It also supports the following shapes:

- Circle
- Rectangle
- Diamond
- Triangle
- InvertedTriangle
- Image

You can customize a shape using the `ShapeWidth` and `ShapeHeight` properties.

Legend padding

You can control the spacing between the legend items using the `Padding` option of the legend. The default value of padding is 5.

Legend border

You can customize the legend border using the `Border` option in the legend. The legend border can be customized using the border `Color` and `Width` properties.

Font of the legend text

The `Font` of the legend item text can be customized using the following properties:

- `fontFamily`
- `fontStyle`
- `fontWeight`
- `opacity`
- `color`
- `size`

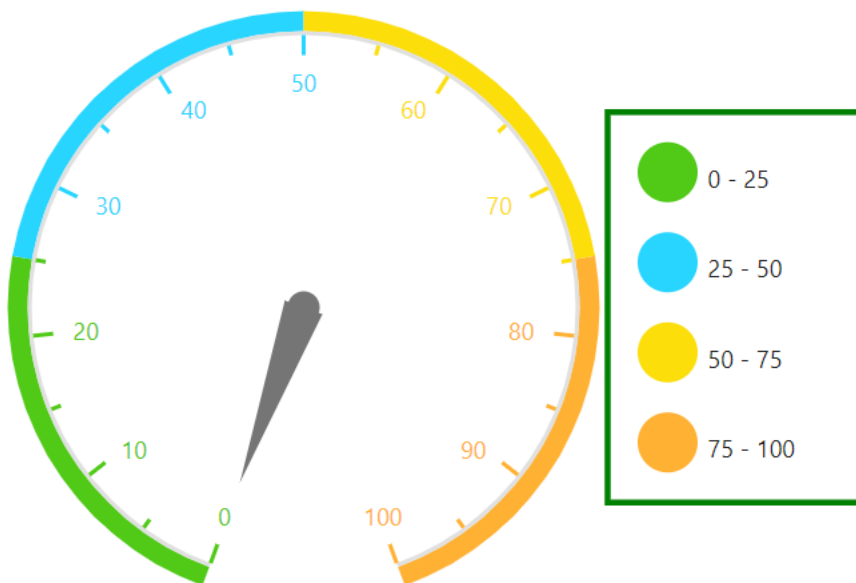
The following code example shows how to add legend in the gauge.

ASPX-CS

```

@using Syncfusion.Blazor.CircularGauge
<SfCircularGauge>
  <CircularGaugeLegendSettings Visible="true" ShapeWidth="30" ShapeHeight="30"
  Padding="15">
    <CircularGaugeLegendBorder Color="green"
    Width="3"></CircularGaugeLegendBorder>
  </CircularGaugeLegendSettings>
  <CircularGaugeAxes>
    <CircularGaugeAxis Minimum="0" Maximum="100">
      <CircularGaugeAxisMajorTicks UseRangeColor="true">
      </CircularGaugeAxisMajorTicks>
      <CircularGaugeAxisMinorTicks UseRangeColor="true">
      </CircularGaugeAxisMinorTicks>
      <CircularGaugeAxisLabelStyle UseRangeColor="true">
      </CircularGaugeAxisLabelStyle>
    </CircularGaugeAxis>
  </CircularGaugeAxes>
  <CircularGaugeRanges>
    <CircularGaugeRange Start="0" End="25" Radius="108%">
    </CircularGaugeRange>
    <CircularGaugeRange Start="25" End="50" Radius="108%">
    </CircularGaugeRange>
    <CircularGaugeRange Start="50" End="75" Radius="108%">
    </CircularGaugeRange>
    <CircularGaugeRange Start="75" End="100" Radius="108%">
    </CircularGaugeRange>
  </CircularGaugeRanges>
</SfCircularGauge>

```



<!-- markdownlint-disable MD036 -->

Toggle option in legend

The toggle option has been provided for legend. So, if you toggle the legend, the given color will be changed to the corresponding circular gauge range. You can enable the toggle option using `ToggleVisibility` in the `CircularGaugeLegendSettings` property.

ASPX-CS

```
@using Syncfusion.Blazor.CircularGauge
<SfCircularGauge>
  <CircularGaugeLegendSettings Visible="true" ToggleVisibility="true">
  <CircularGaugeLegendBorder Color="green"
Width="3"></CircularGaugeLegendBorder>
</CircularGaugeLegendSettings>
<CircularGaugeAxes>
  <CircularGaugeAxis Minimum="0" Maximum="100">
  <CircularGaugeAxisMajorTicks UseRangeColor="true">
</CircularGaugeAxisMajorTicks>
  <CircularGaugeAxisMinorTicks UseRangeColor="true">
</CircularGaugeAxisMinorTicks>
  <CircularGaugeAxisLabelStyle UseRangeColor="true">
</CircularGaugeAxisLabelStyle>
<CircularGaugeRanges>
  <CircularGaugeRange Start="0" End="25" Radius="108%">
</CircularGaugeRange>
  <CircularGaugeRange Start="25" End="50" Radius="108%">
</CircularGaugeRange>
  <CircularGaugeRange Start="50" End="75" Radius="108%">
</CircularGaugeRange>
  <CircularGaugeRange Start="75" End="100" Radius="108%">
</CircularGaugeRange>
</CircularGaugeRanges>
</CircularGaugeAxis>
</CircularGaugeAxes>
</SfCircularGauge>
```



Paging support in legend

By default, paging will be enabled if the legend items exceed the legend bounds. You can view each legend item by navigating between the pages using navigation buttons.

ASPX-CS

```
@using Syncfusion.Blazor.CircularGauge
<SfCircularGauge>
  <CircularGaugeLegendSettings Visible="true" Height="50">
    <CircularGaugeLegendBorder Color="green"
      Width="3"></CircularGaugeLegendBorder>
  </CircularGaugeLegendSettings>
  <CircularGaugeAxes>
    <CircularGaugeAxis Minimum="0" Maximum="100">
      <CircularGaugeAxisMajorTicks UseRangeColor="true">
      </CircularGaugeAxisMajorTicks>
      <CircularGaugeAxisMinorTicks UseRangeColor="true">
      </CircularGaugeAxisMinorTicks>
      <CircularGaugeAxisLabelStyle UseRangeColor="true">
      </CircularGaugeAxisLabelStyle>
    </CircularGaugeAxis>
    <CircularGaugeRanges>
      <CircularGaugeRange Start="0" End="25" Radius="108%">
      </CircularGaugeRange>
      <CircularGaugeRange Start="25" End="50" Radius="108%">
      </CircularGaugeRange>
      <CircularGaugeRange Start="50" End="75" Radius="108%">
      </CircularGaugeRange>
    </CircularGaugeRanges>
  </CircularGaugeAxes>
</SfCircularGauge>
```

```

</CircularGaugeRange>
<CircularGaugeRange Start="75" End="100" Radius="108%">
</CircularGaugeRange>
</CircularGaugeRanges>
</CircularGaugeAxis>
</CircularGaugeAxes>
</SfCircularGauge>

```



Legend text customization

You can customize the legend text using `LabelText` property in `CircularGaugeRange`.

ASPX-CS

```

@using Syncfusion.Blazor.CircularGauge
<SfCircularGauge>
<CircularGaugeLegendSettings Visible="true" Height="50">
<CircularGaugeLegendBorder Color="green"
Width="3"></CircularGaugeLegendBorder>
</CircularGaugeLegendSettings>
<CircularGaugeAxes>
<CircularGaugeAxis Minimum="0" Maximum="100">
<CircularGaugeAxisMajorTicks UseRangeColor="true">
</CircularGaugeAxisMajorTicks>
<CircularGaugeAxisMinorTicks UseRangeColor="true">
</CircularGaugeAxisMinorTicks>
<CircularGaugeAxisLabelStyle UseRangeColor="true">
</CircularGaugeAxisLabelStyle>
<CircularGaugeRanges>
<CircularGaugeRange Start="0" End="25" Radius="108%" LegendText="light air">
</CircularGaugeRange>

```

```
<CircularGaugeRange Start="25" End="50" Radius="108%" LegendText="light  
air">  
</CircularGaugeRange>  
<CircularGaugeRange Start="50" End="75" Radius="108%" LegendText="light  
breez">  
</CircularGaugeRange>  
<CircularGaugeRange Start="75" End="100" Radius="108%" LegendText="Gentle  
breez">  
</CircularGaugeRange>  
</CircularGaugeRanges>  
</CircularGaugeAxis>  
</CircularGaugeAxes>  
</SfCircularGauge>
```



User Interaction in Blazor Circular Gauge Component

Tooltip for pointers

The Circular Gauge displays the pointer details through [CircularGaugeTooltipSettings](#), when the mouse is hovered over a pointer.

Formatting the tooltip

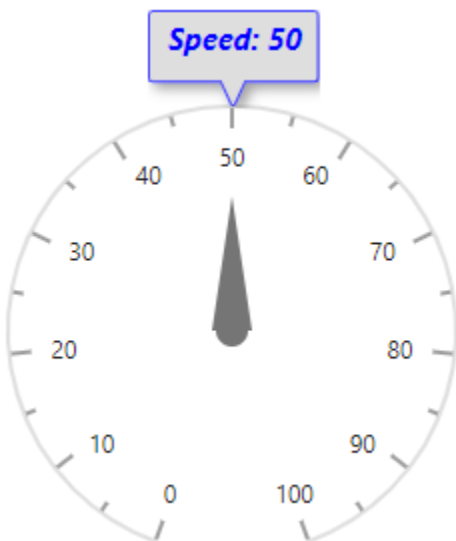
By default, the tooltip is not visible. You can enable the tooltip by setting the [Enable](#) property to true. You can use following properties to customize the tooltip.

- [CircularGaugeTooltipSettings](#)
- [Fill](#) - Specifies fill color for tooltip
- [EnableAnimation](#) - To enable or disable animation
- [Format](#) - To customize the tooltip content
- [CircularGaugeTooltipBorder](#)
- [Color](#) - Specifies tooltip border color
- [Width](#) - Specifies tooltip border width

- [CircularGaugeTooltipTextStyle](#)
- [Color](#) - Specifies tooltip text color
- [FontStyle](#) - Specifies font style for tooltip text
- [FontWeight](#) - Specifies font weight for tooltip text
- [FontFamily](#) - Specifies font family for tooltip
- [Opacity](#) - Specifies opacity for tooltip text
- [Size](#) - Specifies size for tooltip text

ASPX-CS

```
@using Syncfusion.Blazor.CircularGauge
<SfCircularGauge>
<CircularGaugeAxes>
<CircularGaugeAxis>
<CircularGaugePointers>
<CircularGaugePointer Value="50">
</CircularGaugePointer>
</CircularGaugePointers>
<CircularGaugeTooltipSettings Enable="true"
Fill="lightgray"
EnableAnimation="true"
Format="Speed: {value}">
<CircularGaugeTooltipBorder Color="blue"
Width="1">
</CircularGaugeTooltipBorder>
<CircularGaugeTooltipTextStyle Color="blue"
FontStyle="italic"
FontWeight="bold"
Size="15px">
</CircularGaugeTooltipTextStyle>
</CircularGaugeTooltipSettings>
</CircularGaugeAxis>
</CircularGaugeAxes>
</SfCircularGauge>
```

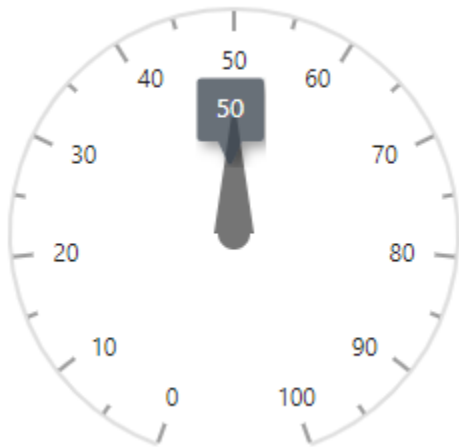


Showing tooltip at mouse position

By default tooltip will be shown on the axis, you can show the tooltip at the cursor position using [ShowAtMousePosition](#) property.

ASPX-CS

```
@using Syncfusion.Blazor.CircularGauge
<SfCircularGauge>
  <CircularGaugeAxes>
    <CircularGaugeAxis>
      <CircularGaugePointers>
        <CircularGaugePointer Value="50">
        </CircularGaugePointer>
      </CircularGaugePointers>
      <CircularGaugeTooltipSettings Enable="true" ShowAtMousePosition="true">
      </CircularGaugeTooltipSettings>
    </CircularGaugeAxis>
  </CircularGaugeAxes>
</SfCircularGauge>
```



Tooltip for ranges

Circular gauge displays the information about the ranges through tooltip when hovering the mouse over the ranges. You can enable this feature by setting the type property of tooltip to 'Range' in the array collection.

Tooltip customization for ranges

To customize the range tooltip, use the `CircularGaugeRangeTooltipSettings` property in tooltip. The following options are available to customize the range tooltip:

- `Fill` - Specifies the range tooltip fill color.
- `CircularGaugeRangeTooltipTextStyle` - Specifies the range tooltip text style.
- `Format` - Specifies the range content format.
- `Template` - Specifies the custom template for tooltip.
- `EnableAnimation` - Animates as it moves from one point to another.
- `CircularGaugeRangeTooltipBorder` - Specifies the tooltip border.
- `showMouseAtPosition` - Displays the position of the tooltip on the cursor position.

Tooltip for annotations

Circular gauge displays the information about the annotations through tooltip when hovering the mouse over the annotation. You can enable this feature by setting the `Type` property of tooltip to 'Annotation' in the array collection.

Tooltip customization for annotations

To customize the annotation tooltip, use the `CircularGaugeAnnotationTooltipSettings` property in tooltip. The following options are available to customize the annotation tooltip:

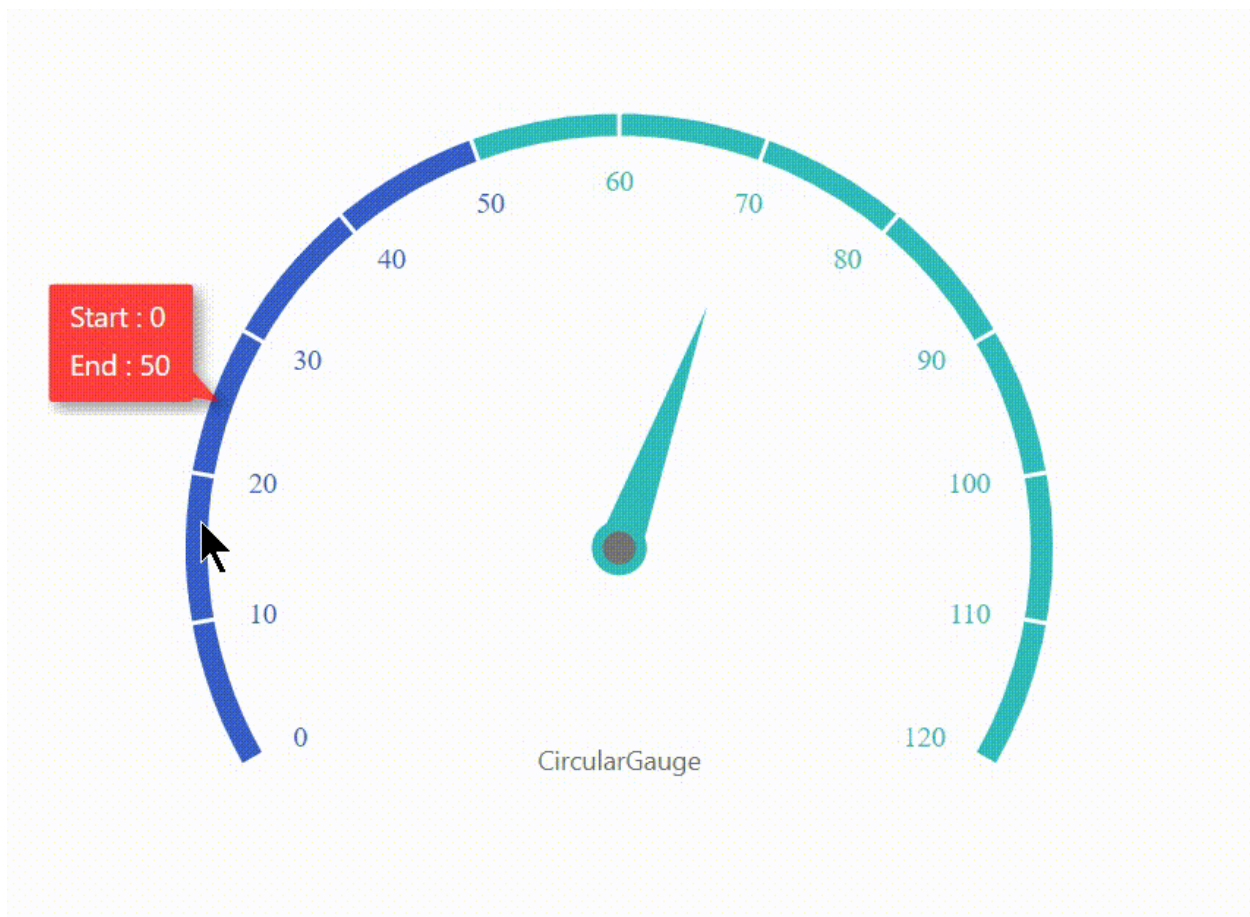
- `Fill` - Specifies the annotation tooltip fill color.
- `CircularGaugeAnnotationTooltipTextStyle` - Specifies the annotation tooltip text style.
- `Format` - Specifies the annotation content format.
- `Template` - Specifies the tooltip content with custom template.
- `EnableAnimation` - Animates as it moves from one point to another.
- `CircularGaugeAnnotationTooltipBorder` - Specifies the tooltip border.

The following code example shows the tooltip for the pointers, ranges, and annotations.

ASPX-CS

```
@using Syncfusion.Blazor.CircularGauge
<SfCircularGauge EnablePointerDrag="true">
  <CircularGaugeTooltipSettings Enable="true" Type="@TooltipType">
    <CircularGaugeAnnotationTooltipSettings Format="Circulargauge">
    </CircularGaugeAnnotationTooltipSettings>
    <CircularGaugeRangeTooltipSettings Fill="red">
    </CircularGaugeRangeTooltipSettings>
  </CircularGaugeTooltipSettings>
  <CircularGaugeAxes>
    <CircularGaugeAxis StartAngle="240" EndAngle="120" Minimum="0" Maximum="120"
    Radius="90%">
    <CircularGaugePointers>
      <CircularGaugePointer Value="70" Radius="60%" Color="#33BCBD">
      <CircularGaugeCap Radius="10" Color="white">
      <CircularGaugeCapBorder Width="5" Color="#33BCBD"></CircularGaugeCapBorder>
      </CircularGaugeCap>
      <CircularGaugePointerAnimation Enable="false">
      </CircularGaugePointerAnimation>
      </CircularGaugePointer>
      </CircularGaugePointers>
      <CircularGaugeAxisMajorTicks Color="white" Height="12" Offset="-5">
      </CircularGaugeAxisMajorTicks>
      <CircularGaugeAxisMinorTicks Color="transparent" Width="0">
      </CircularGaugeAxisMinorTicks>
      <CircularGaugeAxisLabelStyle UseRangeColor="true">
      <CircularGaugeAxisLabelFont Color="#424242" Size="13px" FontFamily="Roboto">
      </CircularGaugeAxisLabelFont>
      </CircularGaugeAxisLabelStyle>
      <CircularGaugeAxisLineStyle Width="0">
      </CircularGaugeAxisLineStyle>
      <CircularGaugeRanges>
      <CircularGaugeRange Start="0" End="50" Radius="102%">
      </CircularGaugeRange>
      <CircularGaugeRange Start="50" End="120" Radius="102%">
      </CircularGaugeRange>
    </CircularGaugeAxis>
  </CircularGaugeAxes>
</SfCircularGauge>
```

```
</CircularGaugeRange>
</CircularGaugeRanges>
<CircularGaugeAnnotations>
<CircularGaugeAnnotation Angle="180" ZIndex="1">
<ContentTemplate>
<div>Circulargauge</div>
</ContentTemplate>
</CircularGaugeAnnotation>
</CircularGaugeAnnotations>
</CircularGaugeAxis>
</CircularGaugeAxes>
</SfCircularGauge>
@code {
public string[] TooltipType = new string[] { "Range", "Annotation",
"Pointer"};
}
```



Dragging pointer

The pointers can be dragged over the axis values by clicking and dragging the pointer. To enable or disable the pointer drag, use the [EnablePointerDrag](#) property.

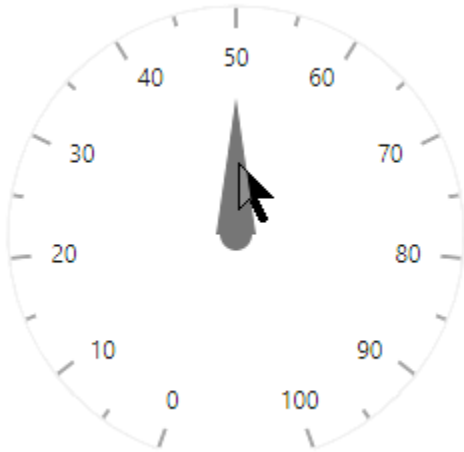
ASPX-CS

```
@using Syncfusion.Blazor.CircularGauge
<SfCircularGauge EnablePointerDrag="true">
```

```

<CircularGaugeAxes >
<CircularGaugeAxis>
<CircularGaugePointers>
<CircularGaugePointer Value="50">
</CircularGaugePointer>
</CircularGaugePointers>
</CircularGaugeAxis>
</CircularGaugeAxes>
</SfCircularGauge>

```



Dragging Range

The ranges can be dragged over the axis values by clicking and dragging the range. To enable or disable the range drag, use the `EnableRangeDrag` property.

ASPX-CS

```

@using Syncfusion.Blazor.CircularGauge
<SfCircularGauge EnableRangeDrag="true">
<CircularGaugeAxes >
<CircularGaugeAxis>
<CircularGaugePointers>
<CircularGaugePointer Value="50">
</CircularGaugePointer>
</CircularGaugePointers>
</CircularGaugeAxis>
</CircularGaugeAxes>
</SfCircularGauge>

```

Print and Export in Blazor Circular Gauge Component

Print

To use the print functionality, you should set the `AllowPrint` property to `true`. The rendered circular gauge can be printed directly from the browser by calling the method `print`. You can get the Circular Gauge component object using `@ref="Gauge"`

ASPX-CS

```

@using Syncfusion.Blazor.CircularGauge

```

```
<button @onclick="PrintGauge">Print</button>
<SfCircularGauge @ref="Gauge" AllowPrint="true">
</SfCircularGauge>
@code {
SfCircularGauge Gauge;
void PrintGauge()
{
this.Gauge.Print();
}
}
```

Print



Export

Image Export

To use the image export functionality, you should set the [AllowImageExport](#) property to **true**. The rendered circular gauge can be exported as an image using the [export](#) method. The method requires two parameters: image type and file name. The circular gauge can be exported as an image in the following formats.

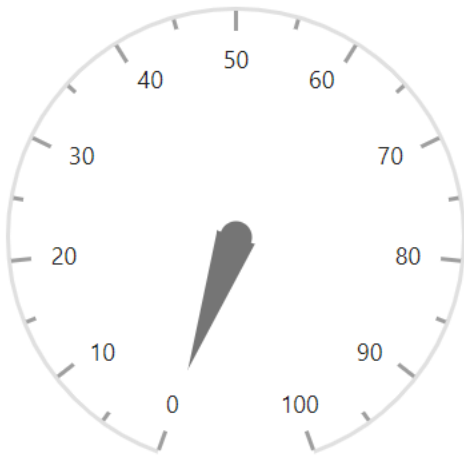
- JPEG
- PNG
- SVG

ASPX-CS

```
@using Syncfusion.Blazor.CircularGauge
<button @onclick="ExportGauge">Export</button>
<SfCircularGauge @ref="Gauge" AllowImageExport="true">
</SfCircularGauge>
@code {
SfCircularGauge Gauge;
void ExportGauge()
{
this.Gauge.Export(ExportType.PNG, "CircularGauge");
}
}
```

```
}
```

Export



PDF Export

To use the PDF export functionality, you should set the [AllowPdfExport](#) property to **true**. The rendered circular gauge can be exported as PDF using the [export](#) method. The [export](#) method requires three parameters: file type, file name and orientation of the PDF document. The orientation setting is optional and "0" indicates portrait and "1" indicates landscape.

ASPX-CS

```
@using Syncfusion.Blazor.CircularGauge
<button @onclick="ExportGauge">Export</button>
<SfCircularGauge @ref="Gauge" AllowPdfExport="true">
</SfCircularGauge>
@code {
    SfCircularGauge Gauge;
    void ExportGauge()
    {
        this.Gauge.Export(ExportType.PDF, "CircularGauge", 0);
    }
}
```



Appearance in Blazor Circular Gauge Component

Circular gauge title

You can add a title to the Circular Gauge using the [Title](#) property. The title can be customized using the [CircularGaugeTitleStyle](#) tag.

ASPX-CS

```
@using Syncfusion.Blazor.CircularGauge
<SfCircularGauge Title="Speedometer">
  <CircularGaugeTitleStyle Color="blue" FontWeight="bold"
    Size="25"></CircularGaugeTitleStyle>
</SfCircularGauge>
```

Speedometer



Circular gauge position

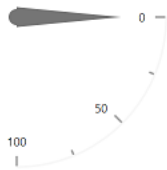
The Circular Gauge can be positioned anywhere in the container using the [CenterX](#) and [CenterY](#) properties which accept values either in percentage or in pixels. The default value of the [CenterX](#) and [CenterY](#) property are 50%, so the Circular Gauge will get rendered to the center of the container.

In pixel

You can set the mid point of the Circular Gauge in pixel as shown below.

ASPX-CS

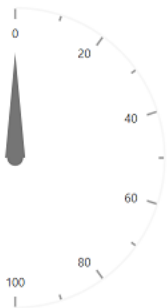
```
@using Syncfusion.Blazor.CircularGauge
<SfCircularGauge CenterX="20px" CenterY="20px">
  <CircularGaugeAxes>
    <CircularGaugeAxis StartAngle="90" EndAngle="180"></CircularGaugeAxis>
  </CircularGaugeAxes>
</SfCircularGauge>
```

*In percentage*

By setting the value in percentage, Circular Gauge gets its mid point with respect to its plot area. For example, when setting the value of [CenterX](#) to '1%' and the value of [CenterY](#) to '50%', the gauge will be positioned at the top-left corner of the plot area.

ASPX-CS

```
@using Syncfusion.Blazor.CircularGauge
<SfCircularGauge CenterX="1%" CenterY="50%">
  <CircularGaugeAxes>
    <CircularGaugeAxis StartAngle="0" EndAngle="180"></CircularGaugeAxis>
  </CircularGaugeAxes>
</SfCircularGauge>
```

**Background customization**

Using the [Background](#) and [CircularGaugeBorder](#) properties, you can change the background color and border of the Circular Gauge.

ASPX-CS

```
@using Syncfusion.Blazor.CircularGauge
<SfCircularGauge Background="skyblue">
  <CircularGaugeBorder Color="#FF0000"
    Width="2">
```

```

</CircularGaugeBorder>
<CircularGaugeAxes>
<CircularGaugeAxis>
<CircularGaugeRanges>
<CircularGaugeRange Start="1"
End="50"
Radius="110%">
</CircularGaugeRange>
<CircularGaugeRange Start="80"
End="100"
Radius="110%">
</CircularGaugeRange>
</CircularGaugeRanges>
<CircularGaugePointers>
<CircularGaugePointer Value="50"
Radius="60%">
</CircularGaugePointer>
</CircularGaugePointers>
</CircularGaugeAxis>
</CircularGaugeAxes>
</SfCircularGauge>

```



Radius calculation based on angles

You can render semi or quarter Circular Gauge by modifying the start and end angles. By enabling the radius based on angle option, the radius of circular gauge will be calculated based on the start and end angles to avoid excess white space.

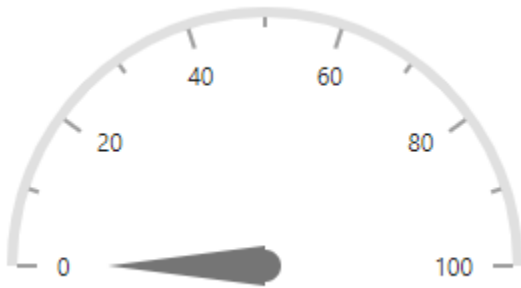
ASPX-CS

```

@using Syncfusion.Blazor.CircularGauge
<SfCircularGauge>
<CircularGaugeAxes>
<CircularGaugeAxis Radius="60%" StartAngle="270" EndAngle="90">
<CircularGaugeAxisLineStyle Width="5">
</CircularGaugeAxisLineStyle>

```

```
</CircularGaugeAxis>  
</CircularGaugeAxes>  
</SfCircularGauge>
```



Accessibility in Blazor Circular Gauge Component

The Circular Gauge provides built-in compliance with the [WAI-ARIA](#) specifications. The WAI-ARIA accessibility supports are achieved through the attributes such as `aria-label`. It helps to provide information about elements in a document for assistive technology.

Aria-label: Provides the text label with some default description for the following elements in gauge.

<!-- markdownlint-disable MD033 -->

Element	Default description
Pointer	Reads the pointer value.
Annotation	Reads the annotation description.
Gauge Title	Reads the gauge title.

You can change this default description using the description property available in [CircularGaugePointers](#), [CircularGaugeAnnotations](#), and [SfCircularGauge](#) tags. It helps the screen reader to read for assistive purpose.

Globalization in Blazor Circular Gauge Component

The internationalization library provides support for formatting and parsing the number using the official [Unicode CLDR](#) JSON data and also provides the `loadCldr` method to load the culture-specific CLDR JSON data. The Circular Gauge component comes with built-in internationalization support to adapt to different cultures.

By default, all the Blazor components are specific to the English culture ('en-US'). If you need a different culture, follow the given steps:

Install the [CLDR-Data](#) package using the following command (it installs the CLDR JSON data).

ASPX-CS

```
@using Syncfusion.Blazor  
@using Syncfusion.Blazor.CircularGauge  
@using Microsoft.JSInterop;  
<SfCircularGauge>
```

```

<CircularGaugeAxes>
<CircularGaugeAxis>
<CircularGaugeAxisLabelStyle Format='c' Position="Position.Inside">
</CircularGaugeAxisLabelStyle>
</CircularGaugeAxis>
</CircularGaugeAxes>
</SfCircularGauge>
@code {
[Inject]
protected IJSRuntime JsRuntime { get; set; }
protected override void OnAfterRender()
{
this.JsRuntime.Sf().LoadCldrData(new string[] { "wwwroot/cldr-
data/de/currencies.json", "wwwroot/cldr-data/de/numbers.json"
}).SetCulture("de").SetCurrencyCode("EUR");
}
}

```



Events in Blazor Circular Gauge Component

Using events in Circular Gauge component

In the following example, the event [OnDragMove](#) binds to the circular gauge component, so the event handler [UpdatePointerValue](#) will be called when you drag the pointer and update the pointer value in the div element.

ASPX-CS

```

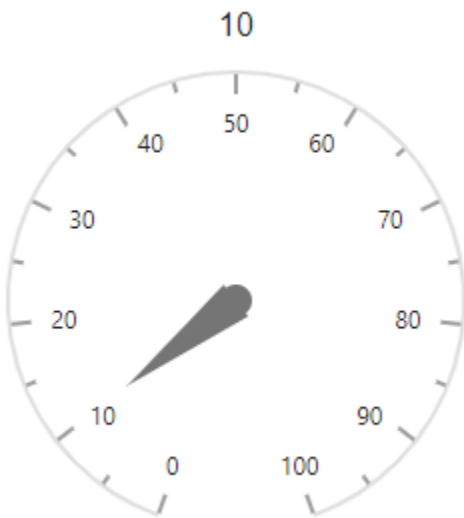
@using Syncfusion.Blazor.CircularGauge
<div style="width:250px">
<div style="text-align: center">@pointerValue</div>
<SfCircularGauge EnablePointerDrag="true" Height="250px" Width="250px">
<CircularGaugeEvents OnDragMove="@UpdatePointerValue"></CircularGaugeEvents>
<CircularGaugeAxes>
<CircularGaugeAxis>
<CircularGaugePointers>
<CircularGaugePointer Value="@pointerValue"></CircularGaugePointer>
</CircularGaugePointers>

```

```

</CircularGaugeAxis>
</CircularGaugeAxes>
</SfCircularGauge>
</div>
@code {
private double pointerValue = 10;
void UpdatePointerValue(IPointerDragEventArgs args)
{
    pointerValue = args.CurrentValue;
}
}

```



Available events

AnimationCompleted

Description: Triggers after animation is completed.

AnnotationRendering

Description: Triggers before rendering on each annotation. You can customize annotations using these arguments.

Argument name	Description
Content	Specifies the annotation content
TextStyle	To customize the text style
Name	Specifies the name of the event
Cancel	Specifies the event cancel status

Loaded

Description: Triggers after the gauge component has been loaded.

OnDragEnd

Description: Triggers when you finished dragging the pointer needle.

Argument name	Description
---------------	-------------

-----	-----
AxisIndex	Specifies the current axis index value
CurrentValue	Specifies the current value of label
PointerIndex	Index of the current pointer instance
Name	Specifies the name of the event

OnDragMove

Description: Triggers when you drag the pointer needle.

Argument name	Description
-----	-----
AxisIndex	Specifies the axis index value
CurrentValue	Specifies the current value of label
PointerIndex	Index of the current pointer instance
PreviousValue	PreviousValue instance
Name	Specifies the name of the event

OnDragStart

Description: Triggers when you start to drag the pointer needle.

Argument name	Description
-----	-----
AxisIndex	Specifies the axis index value
CurrentValue	Specifies the current value of label
PointerIndex	Index of the current pointer instance
Name	Specifies the name of the event

OnGaugeMouseDown

Description: Triggers when you click the mouse on the gauge.

Argument name	Description
-----	-----
Target	Defines current mouse event target id
X	Define the current mouse x location
Y	Define the current mouse y location
Name	Specifies the name of the event

OnGaugeMouseLeave

Description: Triggers when the mouse pointer is moved out of the gauge.

Argument name	Description
-----	-----

Target	Defines current mouse event target id	
X	Define the current mouse x location	
Y	Define the current mouse y location	
Name	Specifies the name of the event	

OnGaugeMouseMove

Description: Triggers when the cursor moves over the gauge.

Argument name	Description	
-----	-----	
Target	Defines current mouse event target id	
X	Define the current mouse x location	
Y	Define the current mouse Y location	
Name	Specifies the name of the event	
Cancel	Specifies the event cancel status	

OnGaugeMouseUp

Description: Triggers when you release a mouse on the gauge.

Argument name	Description	
-----	-----	
Target	Defines current mouse event target id	
X	Define the current mouse x location	
Y	Define the current mouse Y location	
Name	Specifies the name of the event	

OnLoad

Description: Triggers before rendering the gauge. Gauge will trigger this event first.

OnRadiusCalculate

Description: Triggers before the radius is calculated for the gauge. You can customize the gauge radius using these arguments.

Argument name	Description	
-----	-----	
CurrentRadius	Specifies the current radius value	
MidPoint	Specifies the mid point of the gauge location	
Name	Specifies the name of the event	
Cancel	Specifies the event cancel status	

Resizing

Description: Triggers when you resize the gauges.

Argument name	Description	
---------------	-------------	--

-----	-----
CurrentSize	Define the current size of the gauge
PreviousSize	Define the previous size of the gauge
Name	Specifies the name of the event

TooltipRendering

Description: Triggers before rendering the gauge tooltip.

Argument name	Description
-----	-----
Content	Specifies the tooltip text
Event	Specifies the mouse arguments
Location	Specifies the tooltip location
appendInBodyTag	Specifies the tooltip to append in body tag
Tooltip	Tooltip instance, to customize the tooltip settings
Name	Specifies the name of the event
Cancel	Specifies the event cancel status
Axis	Specifies the axis
Range	Specifies the range

Methods in Blazor Circular Gauge Component

The following methods are available in the Circular Gauge component.

SetAnnotationValueAsync

To change the annotation content dynamically, use the [SetAnnotationValueAsync](#) method in the Circular Gauge component. The following are the arguments for this method.

Argument name	Description
-----	-----
axisIndex	Specifies the index of the axis where the annotation is to be placed.
annotationIndex	Specifies the index number of the annotation to be updated.
content	Specifies the text for the annotation to be updated.

ASPX-CS

```
@using Syncfusion.Blazor.CircularGauge
<button style="margin-left:34px" @onclick="ChangeAnnotationValue">Change
annotation value</button>
<SfCircularGauge @ref="gauge" Width="250px" Height="250px">
  <CircularGaugeAxes>
    <CircularGaugeAxis>
      <CircularGaugePointers>
        <CircularGaugePointer Value="50"></CircularGaugePointer>
      </CircularGaugePointers>
    </CircularGaugeAxes>
```



```

<CircularGaugeAnnotation Angle="195" ZIndex="1" Content="Gauge">
</CircularGaugeAnnotation>
</CircularGaugeAnnotations>
</CircularGaugeAxis>
</CircularGaugeAxes>
</SfCircularGauge>
@code {
SfCircularGauge gauge;
public async Task ChangeAnnotationValue()
{
await gauge.SetAnnotationValueAsync(0, 0, "Circular Gauge");
}
}

```

SetPointerValueAsync

To change the pointer value dynamically, use the [SetPointerValueAsync](#) method in the Circular Gauge component. The following are the arguments for this method.

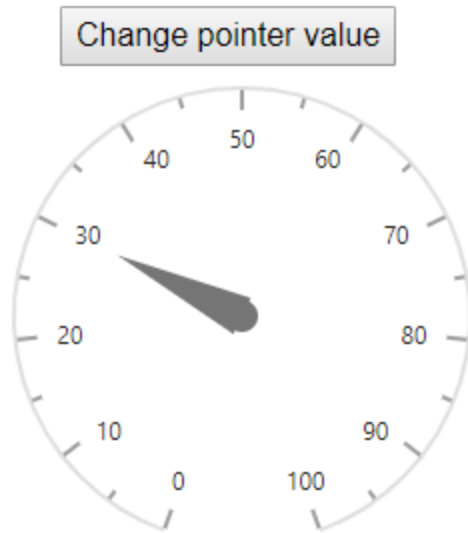
Argument name	Description
axis index	Specifies the index of the axis in which the pointer value is to be updated.
pointerIndex	Specifies the index of the pointer to be updated.
value	Specifies the value of the pointer to be updated.

ASPX-CS

```

@using Syncfusion.Blazor.CircularGauge
<button style="margin-left:34px" @onclick="ChangePoinerValue">Change pointer
value</button>
<SfCircularGauge @ref="gauge" Width="250px" Height="250px">
<CircularGaugeAxes>
<CircularGaugeAxis>
<CircularGaugePointers>
<CircularGaugePointer Value="50"></CircularGaugePointer>
</CircularGaugePointers>
</CircularGaugeAxis>
</CircularGaugeAxes>
</SfCircularGauge>
@code {
SfCircularGauge gauge;
public async Task ChangePoinerValue()
{
await gauge.SetPointerValueAsync(0, 0, 30);
}
}

```



SetRangeValue

To change the start and end of a range in axis, use the [SetRangeValue](#) method in the Circular Gauge component. The following are the arguments for this method.

Argument name	Description
axis index	Specifies the index of the axis in which the range value is to be updated.
rangeIndex	Specifies the index of the range to be updated.
start	Specifies the start value of the range.
end	Specifies the end value of the range

ASPX-CS

```
@using Syncfusion.Blazor.CircularGauge
<button style="margin-left:34px" @onclick="ChangeRangeValue">Change range
value</button>
<SfCircularGauge @ref="gauge" Width="250px" Height="250px">
  <CircularGaugeAxes>
    <CircularGaugeAxis>
      <CircularGaugeRanges>
        <CircularGaugeRange Start="40" End="80">
        </CircularGaugeRange>
      </CircularGaugeRanges>
    </CircularGaugeAxis>
  </CircularGaugeAxes>
</SfCircularGauge>
@code {
  SfCircularGauge gauge;
  public async Task ChangeRangeValue()
  {
    gauge.SetRangeValue(0, 0, 10, 50);
    await gauge.RefreshAsync();
  }
}
```

RefreshAsync

The [RefreshAsync](#) method can be used to change the state of the component and render it again.

ASPX-CS

```
@using Syncfusion.Blazor.CircularGauge
<button style="margin-left:34px" @onclick="RefreshAsync">Refresh</button>
<SfCircularGauge @ref="gauge" Width="250px" Height="250px">
  <CircularGaugeAxes>
    <CircularGaugeAxis>
      <CircularGaugeRanges>
        <CircularGaugeRange Start="40" End="80">
        </CircularGaugeRange>
      </CircularGaugeRanges>
    </CircularGaugeAxis>
  </CircularGaugeAxes>
</SfCircularGauge>
@code {
  SfCircularGauge gauge;
  public async Task RefreshAsync()
  {
    await gauge.RefreshAsync();
  }
}
```

Color Picker

Getting Started with Blazor Color Picker Component

This section briefly explains about how to include Color Picker Component in your Blazor server-side application. You can refer [Getting Started with Syncfusion Blazor for Server-side in Visual Studio](#) page for the introduction and configuring the common specifications.

To get started quickly with Color Picker Component using Blazor, you can check out this video:

{% youtube

"youtube:https://www.youtube.com/watch?v=ll_5h-ZUSHw"%}

Importing Syncfusion Blazor component in the application

1. Install the **Syncfusion.Blazor** NuGet package to the application by using the **NuGet Package Manager**.
2. You can add the client-side style resources through [CDN](#) or from [NuGet](#) package in the `element` of the `~/Pages/_Host.cshtml` page.

HTML

```
<head>
<link href="_content/Syncfusion.Blazor/styles/bootstrap4.css"
rel="stylesheet" />
@*<link href="https://cdn.syncfusion.com/blazor/{{ site.blazorversion
}}/styles/bootstrap4.css" rel="stylesheet" />*@
</head>
```

For Internet Explorer 11, kindly refer the polyfills. Refer the [documentation](#) for more information.

HTML

```
<head>
<link href="_content/Syncfusion.Blazor/styles/bootstrap4.css"
rel="stylesheet" />
<script
src="https://github.com/Daddoon/Blazor.Polyfill/releases/download/3.0.1/blaz
or.polyfill.min.js"></script>
</head>
```

Adding component package to the application

Open `/_Imports.razor` file and import the **Syncfusion.Blazor.Inputs** package.

ASPX-CS

```
@using Syncfusion.Blazor.Inputs
```

Add SyncfusionBlazor service in Startup.cs

Open the **Startup.cs** file and add services required by Syncfusion components.

Add **services.AddSyncfusionBlazor()** method in the `ConfigureServices` function as follows.

CSHARP

```
namespace BlazorApplication
{
    public class Startup
    {
        ....
        ....
        public void ConfigureServices(IServiceCollection services)
        {
            ....
            ....
            services.AddSyncfusionBlazor();
        }
    }
}
```

To enable custom client side resource loading from CRG or CDN, you need to disable resource loading by **AddSyncfusionBlazor(true)** and load the scripts in the HEAD element of the `~/Pages/_Host.cshtml` page.

ASPX-CS

```
<head>
<environment include="Development">
<script src="https://cdn.syncfusion.com/blazor/{{ site.blazorversion
}}/syncfusion-blazor.min.js">
</script>
</environment>
</head>
```

Adding Color Picker component to the application

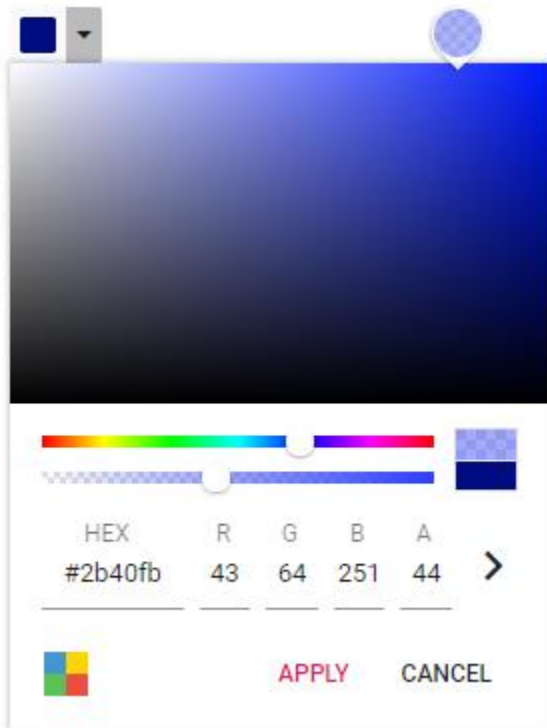
Now, add the Blazor Color Picker component in razor page in the Pages folder. For example, the Color Picker component is added in the ~/Pages/Index.razor page.

ASPX-CS

```
<SfColorPicker></SfColorPicker>
```

Run the application

The Blazor Color Picker component will render in the web browser as shown below,



See Also

- [Getting Started with Syncfusion Blazor for Client-Side in .NET Core CLI](#)
- [Getting Started with Syncfusion Blazor for Server-Side in Visual Studio](#)
- [Getting Started with Syncfusion Blazor for Server-Side in .NET Core CLI](#)

Accessibility in Blazor Color Picker Component

ARIA attributes

The web accessibility makes web content and web applications more accessible for people with disabilities. It especially helps in dynamic content change and development of advanced user interface controls with AJAX, HTML, JavaScript, and related technologies. It helps to provide information about the widget for assistive technology to the disabled person in screen reader.

Color Picker provides built-in compliance with the [WAI-ARIA](#) specifications. WAI-ARIA support is achieved through the attributes like `aria-label` and `aria-selected` applied to the color palette tiles.

| Properties | Functionality |

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

| role | Specified as `gridcell` for the tiles in the color palette. |

| aria-label | Holds the color of the tile. |

| aria-selected | Indicates the current selected state of the tile. |

Keyboard interaction

The following list of keys can be used to interact with the Color Picker after the popup has opened.

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

| --- | --- |

| **Upper Arrow** | Moves the handler/tile up from the current position. |

| **Down Arrow** | Moves the handler/tile down from the current position. |

| **Left Arrow** | Moves the handler/tile left from the current position. |

| **Right Arrow** | Moves the handler/tile right from the current position. |

| **Enter** | Apply the selected color value. |

| **Tab** | To focus the next focusable element in the Color Picker popup. |

Inline Rendering in Blazor Color Picker Component

By default, the ColorPicker will be rendered using SplitButton and open the pop-up to access the ColorPicker. To render the ColorPicker container alone and to access it directly, render it as inline. It can be achieved by setting the [Inline](#) property to `true`.

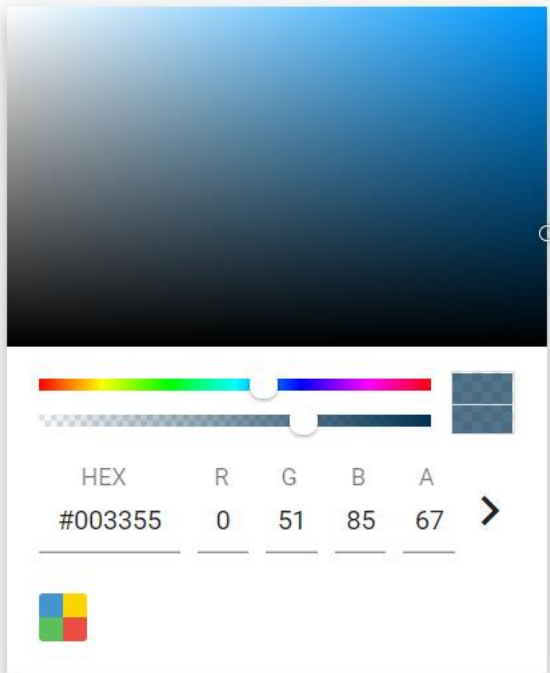
The following sample shows the inline type rendering of ColorPicker.

ASPX-CS

```
@using Syncfusion.Blazor.Inputs
<h4>Choose a color</h4>
<SfColorPicker Value="035a" Inline="true"
ShowButtons="false"></SfColorPicker>
```

Output will be like below,

Choose a color



The `ShowButtons` property is disabled in this sample because the control buttons are not needed for inline type. To know about the control buttons functionality, refer to the [ShowButtons](#).

Mode and Value in Blazor Color Picker Component

Rendering palette at initial load

By default, the `Picker` area will be rendered at initial load. To render the `Palette` area while opening the Color Picker pop-up, specify the `Mode` property as `Palette`.

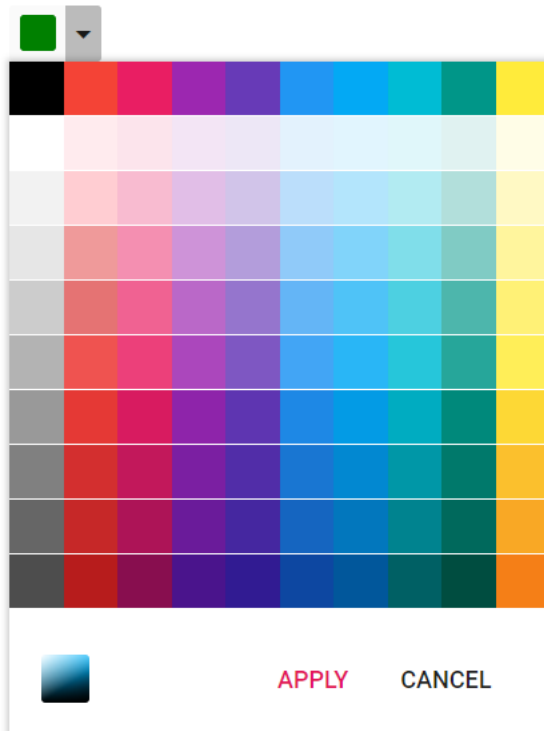
In the following sample, it will render the `Palette` at initial load.

ASPX-CS

```
@using Syncfusion.Blazor.Inputs
<h4>Choose a color</h4>
<SfColorPicker Mode="ColorPickerMode.Palette"></SfColorPicker>
```

Output will be like

Choose a color



Color value

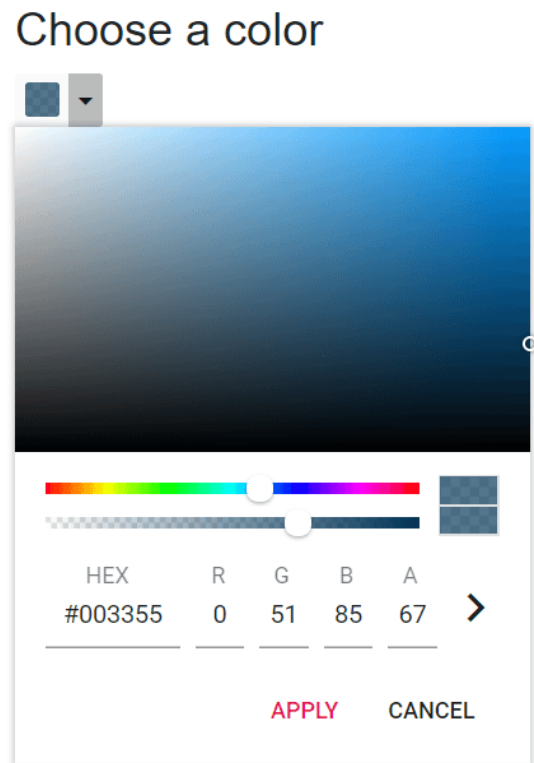
The [Value](#) property can be used to specify the color value to the Color Picker. It supports either three or six digit hex codes. To include opacity, set the color value as four or eight digit hex code.

In the following sample, the color value is set as four digit hex code, the last digit represents the opacity value.

ASPX-CS

```
@using Syncfusion.Blazor.Inputs
<h4>Choose a color</h4>
<SfColorPicker Value="035a" ModeSwitcher="false"></SfColorPicker>
```

Output will be like below,



The [Value](#) property supports hex code with or without # prefix.

Localization and RTL in Blazor Color Picker Component

Localization

The **Localization** library allows you to localize default text content of the Color Picker. The Color Picker component has static text for control buttons (apply / cancel) and mode switcher that can be changed to other cultures (Arabic, Deutsch, French, etc.) by defining the locale value and translation object. You can refer [How to enable Localization in Blazor application](#) page for the introduction and configuring the localization.

You can modify the default value in .res file added to Resource folder. Enter the key value (Locale Keywords) in the **Name** column and the translated string in the **Value** column. The following list of keys and its values are used in the Color Picker.

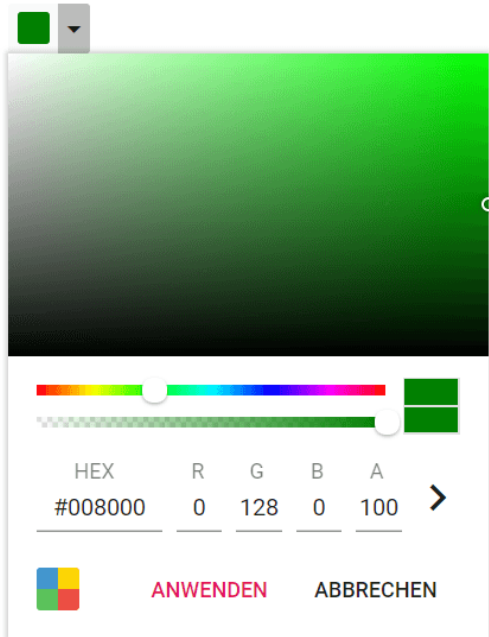
Locale key	en-US (default culture)	de (Deutsch culture)
-----	-----	-----
ColorPicker_Apply	Apply	Anwenden
ColorPicker_Cancel	Cancel	Abbrechen
ColorPicker_ModeSwitcher	Switch Mode	Modus wechseln

ASPX-CS

```
@using Syncfusion.Blazor.Inputs
```

```
<SfColorPicker></SfColorPicker>
```

Output will be like below,



RTL

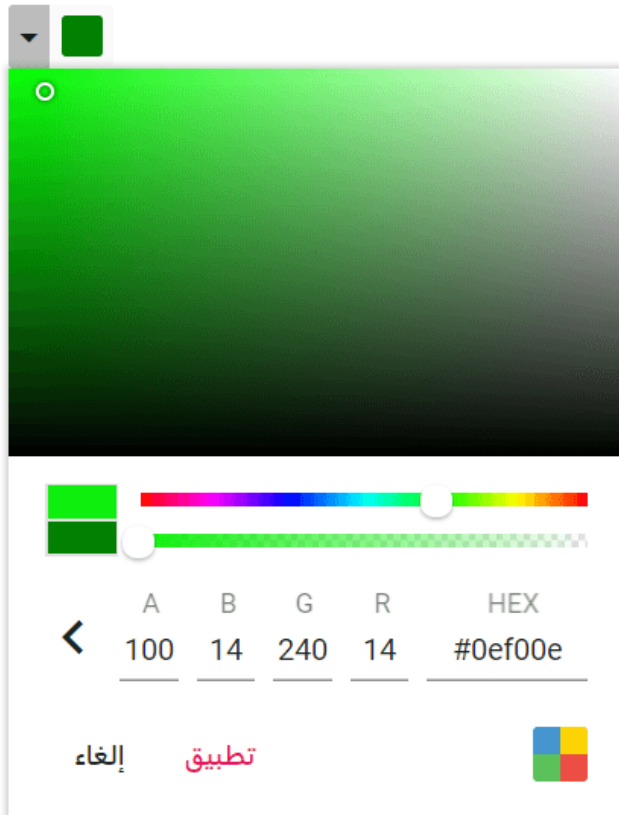
Color Picker component has **RTL** support. It helps to render the Color Picker from right-to-left direction. It improves the user experiences and accessibility for users who use right-to-left languages (Arabic, Farsi, Urdu, etc). This can be achieved by setting the [EnableRtl](#) property to true.

In the following example, Color Picker component is rendered in RTL mode with **arabic** locale.

ASPX-CS

```
@using Syncfusion.Blazor.Inputs  
<SfColorPicker EnableRtl="true"></SfColorPicker>
```

Output will be like below,



Styles and Appearances in Blazor Color Picker Component

To modify the ColorPicker appearance, you need to override the default CSS of ColorPicker component. Please find the list of CSS classes and its corresponding section in ColorPicker 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-colorpicker-popup .e-container .e-handler|To customize Color Picker selection handler. |

|.e-colorpicker-popup .e-container.e-color-picker|To customize the Color Picker container. |

|.e-colorpicker-popup .e-container .e-palette .e-tile|To customize the Color Picker palette item. |

|.e-colorpicker-popup .e-container .e-switch-ctrl-btn |To customize the Color Picker switch control. |

|.e-colorpicker-popup .e-container .e-slider-preview|To customize the Color Picker slider control. |

How To

Customize Color Picker in Blazor Color Picker Component

Custom Palette

By default, the Palette will be rendered with default colors. To load custom colors in the palette, specify the colors in the [PresetColors](#) property. To customize the color palette, add a custom class to palette tiles using [OnTileRender](#) event.

ASPX-CS

```

@using Syncfusion.Blazor.Inputs
<div id="preview" style="@styleValue"></div>
<h4>Select a color</h4>
<SfColorPicker Mode="ColorPickerMode.Palette" CssClass="circle-palette"
ModeSwitcher="false" Inline="true" ShowButtons="false" Columns="4"
PresetColors="@customColors" ValueChange="OnChange"></SfColorPicker>
@code {
private string styleValue = "background-color: #008000";
private void OnChange(ColorPickerEventArgs args)
{
styleValue = "background-color: " + args.CurrentValue.Hex;
}
private Dictionary<string, string[]> customColors = new Dictionary<string,
string[]> {
{ "Custom", new string[] { "#ef9a9a", "#e57373", "#ef5350", "#f44336",
"#f48fb1", "#f06292",
"#ec407a", "#e91e63", "#ce93d8", "#ba68c8", "#ab47bc", "#9c27b0",
"#b39ddb", "#9575cd",
"#7e57c2", "#673ab7", "#9fa8da", "#7986cb", "#5c6bc0", "#3f51b5", "#90caf9",
"#64b5f6",
"#42a5f5", "#2196f3", "#81d4fa", "#4fc3f7", "#29b6f6", "#03a9f4", "#80deea",
"#4dd0e1",
"#26c6da", "#00bcd4", "#80cbc4", "#4db6ac", "#26a69a", "#009688", "#a5d6a7",
"#81c784",
"#66bb6a", "#4caf50", "#c5e1a5", "#aed581", "#9ccc65", "#8bc34a",
"#e6ee9c", "#dce775",
"#d4e157", "#cddc39" }
}
};
}
<style>
#preview {
height: 50px;
width: 50%;
}
.circle-palette .e-container {
background-color: transparent;
border-color: transparent;
box-shadow: none;
}
.circle-palette .e-container .e-custom-palette.e-palette-group {
height: 182px;
}
.circle-palette .e-container .e-palette .e-tile {
border: 0;
color: #fff;
height: 36px;
font-size: 18px;
width: 36px;
line-height: 36px;
border-radius: 50%;
margin: 2px 5px;
font-family: "e-icons";
font-style: normal;
font-variant: normal;
font-weight: normal;
text-transform: none;

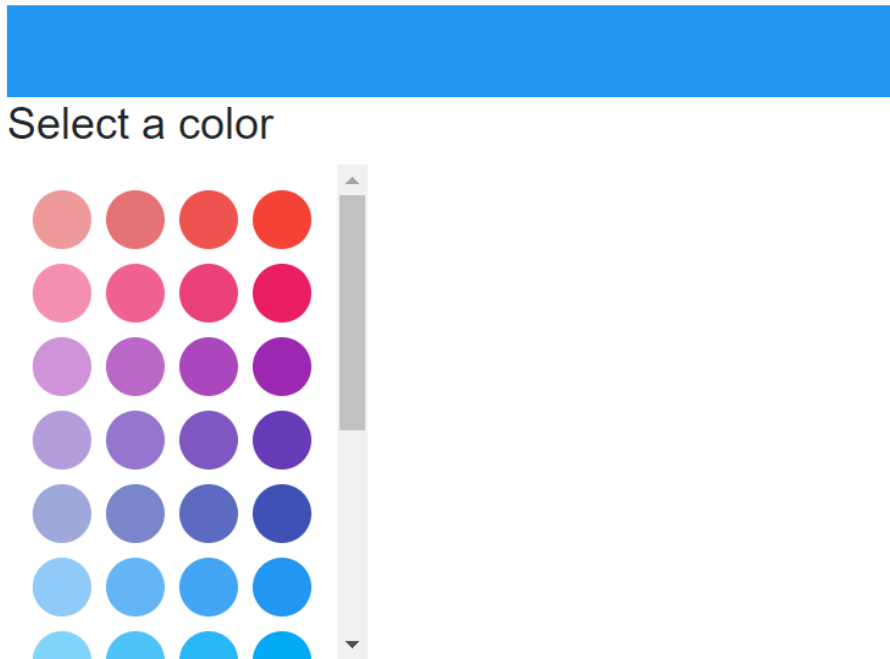
```

```

}
.circle-palette .e-container .e-palette .e-tile.e-selected::before {
content: '\e933';
}
.circle-palette .e-container .e-palette .e-tile.e-selected {
outline: none;
}
</style>

```

Output will be as follows



Hide input area from picker

By default, the input area will be rendered in Color Picker. To hide the input area from it, add `e-hide-value` class to Color Picker using the [CssClass](#) property.

In the following sample, the Color Picker is rendered without input area.

ASPX-CS

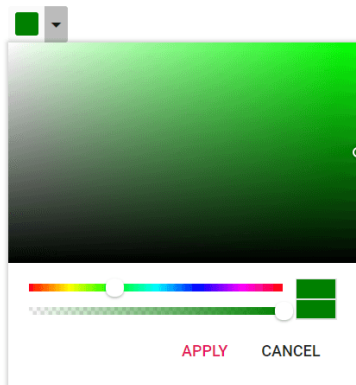
```

@using Syncfusion.Blazor.Inputs
<h4>Choose a color</h4>
<SfColorPicker ModeSwitcher="false" CssClass="e-hide-value"></SfColorPicker>

```

Output will be as follows

Choose a color



Custom Handle

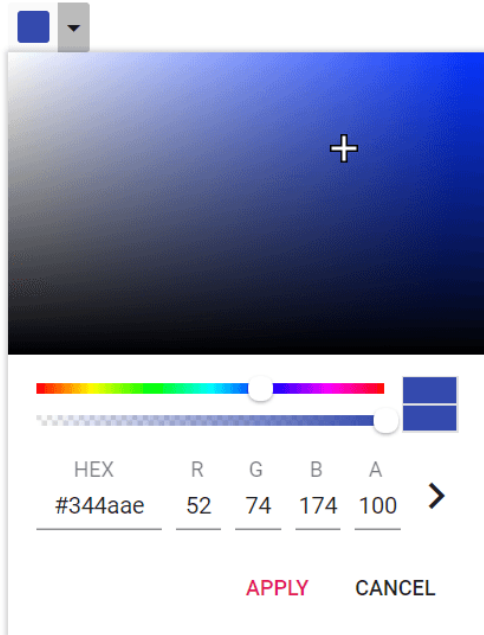
Color Picker handle shape and UI can be customized. Here, we have customized the handle as svg icon. Similarly, you can customize the handle based on your requirement.

The following sample shows the customized Color Picker handle.

ASPX-CS

```
@using Syncfusion.Blazor.Inputs
<SfColorPicker Value="#344aae" CssClass="e-custom-picker"
ModeSwitcher="false"></SfColorPicker>
<style>
.e-color-picker-tooltip.e-popup.e-popup-open {
display: none;
}
.e-custom-picker .e-container .e-handler {
background: transparent
url('data:image/svg+xml;base64,PD94bWwgdmVyc2lvcj0iMS4wIiBlbmNvZGluZz0idXRmLTgiPz4KPCVtLSBHZW51cmF0b3I6IEFkb2JlIElsbHVzdHJhdG9yIDIyLjEuMCwgU1ZHIIEV4cG9ydCBQbHVnLULuIC4gU1ZHIHFZlcnNpb246IDYuMDAgQnVpbGQgMCkgIC0tPgo8c3ZnIHZlcnNpb249IjEuMSIgaWQ9IkkxeWVyXzEiIHhtbG5zPSJodHRwOi8vd3d3LnczLm9yZy8yMDAwL3N2ZyIgeG1sbmM6eGxpbnMs9Imh0dHA6Ly93d3d3cudzMub3JnLzE5OTkveGxpbnMsIiHg9IjBweCIgeT0iMHB4IgoJIHJpZXNpdC3g9IjAgMCAxNiAxNiIgc3R5bGU9ImVuYWJsZS1iYWNRZ3JvdW5kOm5ldyAwIDAgMTYgMTY7IiB4bWw6c3BhY2U9InByZXNlcnZlIj4KPHN0eWxlIHR5cGU9InRleHQvY3NzIj4KCS5zdDB7ZmlsbDojRkZGRkZGO30KPC9zdHlsZT4KPGC+Cgk8cG9seWdvbiBjbGFzc0ic3QwIiBwb2ludHM9IjE2LDYgMTAsNiAxMCwgIDYsMCA2LDYgMCw2IDAsMTAgNiwxMCA2LDE2IDEwLDE2IDEwLDEwIDE2LDEwIAkiLz4KPC9nPgo8cGF0aCBkPSJNMTAsNlYwSDZ2NkgwdjRoNnY2aDR2LTZoNlY2SDEweiBNMTUsOUg5djZINlY5SDFWN2g2VjFoMnY2aDZWOXoiLz4KPC9zdmc+Cg==') ;
font-size: 16px;
height: 16px;
line-height: 16px;
margin-left: -8px;
margin-top: -8px;
border: none;
box-shadow: none;
width: 16px;
}
</style>
```

Output will be as follows



Disable Color Picker in Blazor Color Picker Component

To achieve disabled state in Color Picker, set the [Disabled](#) property to `true`. The Color Picker pop-up cannot be accessed in disabled state.

The following example shows the `Disabled` state of Color Picker component.

ASPX-CS

```
@using Syncfusion.Blazor.Inputs
<h4>Choose a color</h4>
<SfColorPicker Disabled="true"></SfColorPicker>
```

Output will be as follows

Choose a color



Handle No Color Support in Blazor Color Picker Component

The Color Picker component supports no color functionality. By clicking the no color tile from palette, the selected color becomes `empty` and considered as no color has been selected from Color Picker.

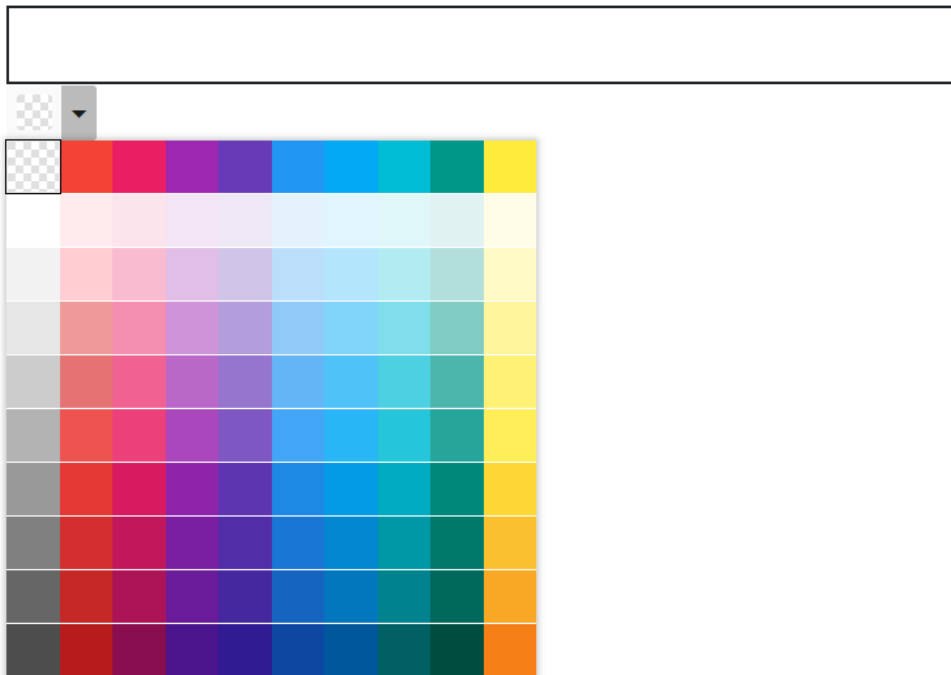
Default no color

To achieve this, set `NoColor` property as true. In the following sample, the first tile of the color palette represents the no color tile. By clicking the no color tile, you can achieve the above functionalities.

ASPX-CS

```
@using Syncfusion.Blazor.Inputs
<div id="preview" style="@colorValue"></div>
<SfColorPicker NoColor="true" Mode="ColorPickerMode.Palette"
ShowButtons="false" ModeSwitcher="false"
ValueChange="@Changed"></SfColorPicker>
@code {
private string colorValue = "background-color: #008000";
private void Changed(ColorPickerEventArgs args)
{
colorValue = "background-color:" + args.CurrentValue.Hex;
}
}
<style>
#preview {
border: 1px solid;
height: 40px;
margin-bottom: 10px;
width: 300px;
}
</style>
```

Output will be as follows



Custom No Color

The following sample shows the color palette with custom no color option.

ASPX-CS

```

@using Syncfusion.Blazor.Inputs
@using Syncfusion.Blazor.SplitButtons
<div id="preview" style="@colorValue"></div>
<SfSplitButton @ref="splitBtn" IconCss="e-icons e-picker" CssClass="color-
picker">
  <PopupContent>
    <ul class="e-dropdown-menu" tabindex="0">
      <li class="e-item e-palette-item">
        <SfColorPicker Columns="4" Inline="true" Mode="ColorPickerMode.Palette"
        PresetColors="@customColors" ShowButtons="false" ModeSwitcher="false"
        ValueChange="@Changed"></SfColorPicker>
      </li>
      <li class="e-item e-separator"></li>
      <li class="e-item" @onclick="@NoColorHandler" id="no-color" tabindex="-1">
        <span class="e-menu-icon e-nocolor"></span>
        No color
      </li>
    </ul>
  </PopupContent>
</SfSplitButton>
@code {
private SfSplitButton splitBtn;
private string colorValue = "background-color: #008000";
private Dictionary<string, string[]> customColors = new Dictionary<string,
string[]> {
  {
    "Custom", new string[] { "#f44336", "#e91e63", "#9c27b0", "#673ab7",
    "#2196f3", "#03a9f4", "#00bcd4", "#009688", "#8bc34a", "#cddc39", "#ffeb3b",
    "#ffc107" }
  }
};
private void Changed(ColorPickerEventArgs args)
{
  colorValue = "background-color:" + args.CurrentValue.Hex;
  splitBtn.Toggle();
}
private void NoColorHandler()
{
  colorValue = "background-color: transparent";
  splitBtn.Toggle();
}
}
<style>
.e-picker::before {
content: '\e35c'
}
#preview {
border: 1px solid;
height: 40px;
width: 300px;
margin-bottom: 10px;
}
.color-picker.e-dropdown-popup ul {
padding: 0;
}

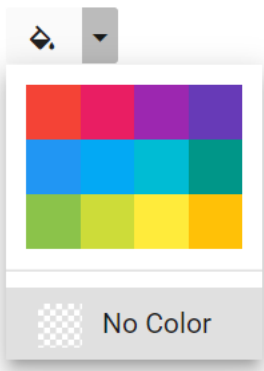
```

```

.color-picker.e-dropdown-popup ul .e-container {
box-shadow: none;
}
.color-picker.e-dropdown-popup .e-container .e-custom-palette .e-palette {
padding-bottom: 2px;
}
.color-picker.e-dropdown-popup ul .e-item.e-palette-item {
height: auto;
padding: 0;
}
.color-picker.e-dropdown-popup ul .e-item .e-menu-icon.e-nocolor {
height: 22px;
margin-top: 8px;
width: 22px;
background: transparent
url('data:image/svg+xml;base64,PD94bWwgdmVyc2lvcj0iMS4wIiBlbmNvZGluZz0iVVRGL
TgiPz4KPHN2ZyB3aWR0aD0iNnB4IiBoZWlnaHQ9IjZweCIgdmlld0JveD0iMCAwIDYgNiIgdmVyc
2lvcj0iMS4xIiB4bWxucz0iaHR0cDovL3d3dy53My5vcmcvMjAwMC9zdmciIHhtbG5zOnhsaW50PS
SjodHRwOi8vd3d3LnczLm9yZy8xOTk5L3hsaW50Ij4KICAgIDwhLS0gR2VuZXJhdG9yOiBTa2V0Y
2ggNTAgKDU0OTgzKSA0IGh0dHA6Ly93d3cuYm9oZWlpcjYw5jb2RpbmcuY29tL3NrZXRjaCA0LT4KI
CAGIDx0aXR5ZT5Hcm91cCA5PC90aXR5ZT4KICAgIDxkZXNjPkNyZWZ0ZWQgd2l0aCBTa2V0Y2guP
C9kZXNjPgogICAgPGRlZnM+PC9kZWZzPgogICAgPGcgaWQ9IiBhZ2UtMSIgc3Ryb2t1PSJub25lI
iBzdHJva2Utd2lkdGg9IjEiIGZpbGw9Im5vbmUiIGZpbGwtcnVsZT0iZXZlbn9kZCI+CiAgICAgI
CAGPGcgaWQ9Ikd2b3VwLTkiPgogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIC
WxsPSIjRTBFMEUwIiB4PSIwIiB5PSIwIiB3aWR0aD0iMyIgaGVpZ2h0PSIzIj48L3JlY3Q+CiAgI
CAGICAgICAgIDxyZWN0IGlkPSJSZWN0YW5nbGUtMTetQ29weS0yIiBmaWxsPSIjRkZGRkZGIiB4P
SIwIiB5PSIzIiB3aWR0aD0iMyIgaGVpZ2h0PSIzIj48L3JlY3Q+CiAgICAgICAgICAgICAgIDxyZWN0I
GlkPSJSZWN0YW5nbGUtMTetQ29weSIgZmlsbD0iI0ZGRkZGRiIgeD0iMyIgeT0iMCIgd2lkdGg9I
jMiIGhlaWdodD0iMyI+PC9yZWN0PgogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIC
UNvcHktMyIgzmlsbD0iI0UwRTBFMCIgeD0iMyIgeT0iMyIgd2lkdGg9IjMiIGhlaWdodD0iMyI+P
C9yZWN0PgogICAgICAgIDwvZz4KICAgIDwvZz4KPC9zdmc+');
}
</style>

```

Output will be as follows



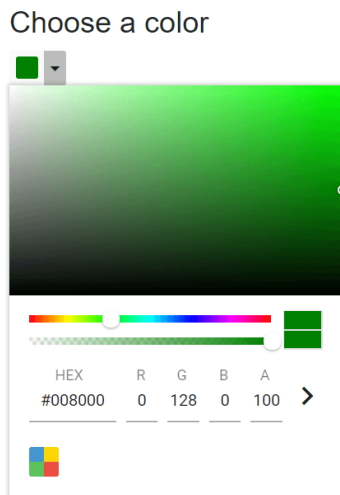
Hide control buttons in Blazor Color Picker Component

Color Picker can be rendered without control buttons (Apply/Cancel). In this case, while selecting a color, the Color Picker pop-up is closed and selected color can be applied directly. To hide control buttons, set the [ShowButtons](#) property to `false`.

ASPX-CS

```
@using Syncfusion.Blazor.Inputs
<h4>Choose a color</h4>
<SfColorPicker ShowButtons="false"></SfColorPicker>
```

Output will be as follows



Render palette alone in Blazor Color Picker Component

To render the `Palette` alone in Color Picker, specify the [Mode](#) property as `Palette`, and set the [ModeSwitcher](#) property to `false`.

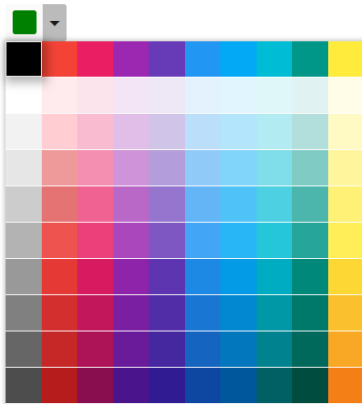
In the following sample, the [ShowButtons](#) property is set to `false` to hide the control buttons and it renders only the `Palette` area.

ASPX-CS

```
@using Syncfusion.Blazor.Inputs
<h4>Choose a color</h4>
<SfColorPicker Mode="ColorPickerMode.Palette" ModeSwitcher="false"
ShowButtons="false"></SfColorPicker>
```

Output will be as follows

Choose a color



To render Picker alone, specify the [Mode](#) property as 'Picker'.

ComboBox

Getting Started with Blazor ComboBox Component

This section briefly explains how to include a **ComboBox** Component in your Blazor client-side application. You can refer to the [Getting Started with Syncfusion Blazor for Client-side in Visual Studio 2019](#) page for introduction and configure the common specifications.

To get started quickly with Blazor ComboBox component, you can check out this video.

{% youtube

"youtube:https://www.youtube.com/watch?v=VYK2xHC_Lrg"%}

Importing Syncfusion Blazor component in the application

- Install Syncfusion.Blazor.DropDowns NuGet package to the application by using the NuGet Package Manager.
- You can add the client-side resources through [CDN](#) or from [NuGet](#) package in the **HEAD** element of the ~/Pages/_Host.cshtml page.

HTML

```
<head>
<link href="_content/Syncfusion.Blazor.Themes/bootstrap4.css"
rel="stylesheet" />
@*<link
href="https://cdn.syncfusion.com/blazor/{{version}}/styles/{{theme}}.css"
rel="stylesheet" />*@
</head>
```

For Internet Explorer 11, kindly refer the polyfills. Refer the [documentation](#) for more information.

HTML

```
<head>
<link href="_content/Syncfusion.Blazor.Themes/bootstrap4.css"
rel="stylesheet" />
<script
src="https://github.com/Daddoon/Blazor.Polyfill/releases/download/3.0.1/blaz
or.polyfill.min.js"></script>
</head>
```

Adding component package to the application

Open `~/_Imports.razor` file and import the `Syncfusion.Blazor.DropDowns` package.

ASPX-CS

```
@using Syncfusion.Blazor.DropDowns
```

Add SyncfusionBlazor service in Startup.cs

Open the **Startup.cs** file and add services required by Syncfusion components using **services.AddSyncfusionBlazor()** method. Add this method in the **ConfigureServices** function as follows.

CSHARP

```
using Syncfusion.Blazor;
namespace BlazorApplication
{
    public class Startup
    {
        ....
        ....
        public void ConfigureServices(IServiceCollection services)
        {
            ....
            ....
            services.AddSyncfusionBlazor();
        }
    }
}
```

To enable custom client side resource loading from CRG or CDN. You need to disable resource loading by `AddSyncfusionBlazor(true)` and load the scripts in the **HEAD** element of the `~/Pages/_Host.cshtml` page.

HTML

```
<head>
<script src="https://cdn.syncfusion.com/blazor/{{ site.blazorversion
}}/syncfusion-blazor.min.js"></script>
</head>
```

Adding ComboBox component to the application

To initialize the ComboBox component, add the following code to your `Index.razor` view page which is present under `~/Pages` folder.

ASPX-CS

```
<SfComboBox TValue="string" Placeholder="Select a game"></SfComboBox>
```

Run the application

After successful compilation of your application, press **F5** to run the application.

The output will be as follows.



Binding data source

After initializing, populate the ComboBox with data using the [DataSource](#) property. Here, an array of string values is passed to the ComboBox component.

The following example illustrates the output in your browser.

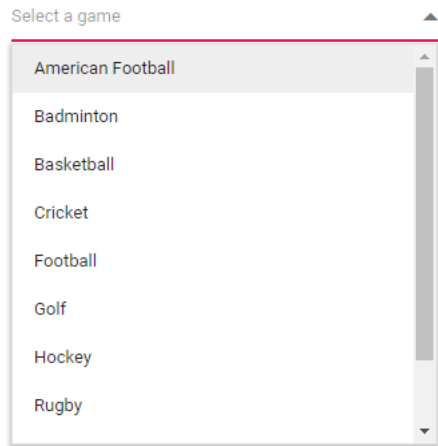
ASPX-CS

```
<SfComboBox TValue="string" TItem="Games" Placeholder="Select a game"
DataSource="@LocalData">
  <ComboBoxFieldSettings Value="ID" Text="Text"></ComboBoxFieldSettings>
</SfComboBox>

@code {
public class Games
{
public string ID { get; set; }
public string Text { get; set; }
}

List<Games> LocalData = new List<Games> {
new Games() { ID= "Game1", Text= "American Football" },
new Games() { ID= "Game2", Text= "Badminton" },
new Games() { ID= "Game3", Text= "Basketball" },
new Games() { ID= "Game4", Text= "Cricket" },
new Games() { ID= "Game5", Text= "Football" },
new Games() { ID= "Game6", Text= "Golf" },
new Games() { ID= "Game7", Text= "Hockey" },
new Games() { ID= "Game8", Text= "Rugby"},
new Games() { ID= "Game9", Text= "Snooker" },
new Games() { ID= "Game10", Text= "Tennis"},
};
}
```

The output will be as follows.



Custom values

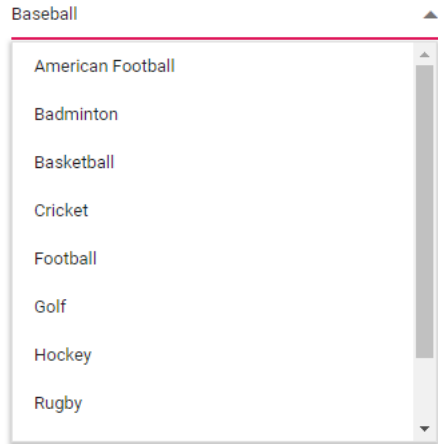
The ComboBox allows the users to give input as custom value, which is not required to present in predefined

set of values. By default, this support is enabled by [AllowCustom](#) property. In this case, both text field and value field are considered as same. The custom value will be sent to post back handler when a form is about to be submitted.

ASPX-CS

```
<SfComboBox TValue="string" TItem="Games" AllowCustom=true
Placeholder="Select a game" DataSource="@LocalData">
<ComboBoxFieldSettings Value="ID" Text="Text"></ComboBoxFieldSettings>
</SfComboBox>
@code {
public class Games
{
public string ID { get; set; }
public string Text { get; set; }
}
List<Games> LocalData = new List<Games> {
new Games() { ID= "Game1", Text= "American Football" },
new Games() { ID= "Game2", Text= "Badminton" },
new Games() { ID= "Game3", Text= "Basketball" },
new Games() { ID= "Game4", Text= "Cricket" },
new Games() { ID= "Game5", Text= "Football" },
new Games() { ID= "Game6", Text= "Golf" },
new Games() { ID= "Game7", Text= "Hockey" },
new Games() { ID= "Game8", Text= "Rugby"},
new Games() { ID= "Game9", Text= "Snooker" },
new Games() { ID= "Game10", Text= "Tennis"},
};
}
```

The output will be as follows.



Configure the popup list

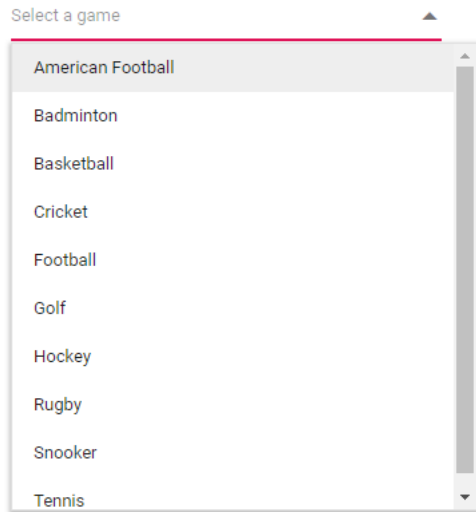
By default, the width of the popup list automatically adjusts according to the ComboBox input element's width, and the height of the popup list has 350px. The height and width of the popup list can also be customized using the [PopupHeight](#) and [PopupWidth](#) properties respectively.

In the following sample, popup list's width and height are configured.

ASPX-CS

```
<SfComboBox TValue="string" TItem="Games" PopupHeight="350px"
PopupWidth="350px" Placeholder="Select a game" DataSource="@LocalData">
<ComboBoxFieldSettings Value="ID" Text="Text"></ComboBoxFieldSettings>
</SfComboBox>
@code {
public class Games
{
public string ID { get; set; }
public string Text { get; set; }
}
List<Games> LocalData = new List<Games> {
new Games() { ID= "Game1", Text= "American Football" },
new Games() { ID= "Game2", Text= "Badminton" },
new Games() { ID= "Game3", Text= "Basketball" },
new Games() { ID= "Game4", Text= "Cricket" },
new Games() { ID= "Game5", Text= "Football" },
new Games() { ID= "Game6", Text= "Golf" },
new Games() { ID= "Game7", Text= "Hockey" },
new Games() { ID= "Game8", Text= "Rugby"},
new Games() { ID= "Game9", Text= "Snooker" },
new Games() { ID= "Game10", Text= "Tennis"},
};
}
```

The output will be as follows.



See Also

- [Getting Started with Syncfusion Blazor for Client-Side in .NET Core CLI](#)
- [Getting Started with Syncfusion Blazor for Server-side in Visual Studio](#)
- [Getting Started with Syncfusion Blazor for Server-Side in .NET Core CLI](#)

Data Binding in Blazor ComboBox Component

Data binding can be achieved by using the **bind-Value** attribute and it supports string, int, Enum, and bool types. If the component value is changed, it will affect all the places where we bind the variable for the **bind-value** attribute.

ASPX-CS

```
@using Syncfusion.Blazor.DropDowns
<p>ComboBox value is:<strong>@ComboVal</strong></p>
<SfComboBox Placeholder="e.g. Australia" @bind-Value="@ComboVal"
DataSource="@Country">
  <ComboBoxFieldSettings Value="Name"></ComboBoxFieldSettings>
</SfComboBox>
@code {
public string ComboVal = "Austarila";
public class Countries
{
public string Name { get; set; }
public string Code { get; set; }
}
List<Countries> Country = new List<Countries>
{
new Countries() { Name = "Australia", Code = "AU" },
new Countries() { Name = "Bermuda", Code = "BM" },
new Countries() { Name = "Canada", Code = "CA" },
new Countries() { Name = "Cameroon", Code = "CM" },
};
}
```

Index Value Binding

Index value binding can be achieved by using `bind-Index` attribute and it supports int and int nullable types. By using this attribute you can bind the values respective to its index.

ASPX-CS

```
@using Syncfusion.Blazor.DropDowns
<SfComboBox TValue="string" Placeholder="e.g. Australia" TItem="Countries"
@bind-Index="@ddlIndex" DataSource="@Country">
<ComboBoxFieldSettings Value="Name"></ComboBoxFieldSettings>
</SfComboBox>
@code {
private int? ddlIndex { get; set; } = 1;
public class Countries
{
public string Name { get; set; }
public string Code { get; set; }
}
List<Countries> Country = new List<Countries>
{
new Countries() { Name = "Australia", Code = "AU" },
new Countries() { Name = "Bermuda", Code = "BM" },
new Countries() { Name = "Canada", Code = "CA" },
new Countries() { Name = "Cameroon", Code = "CM" },
};
}
```

Data Source in Blazor ComboBox Component

The ComboBox 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 ComboBox also supports different kinds of data services such as OData, OData V4, and Web API, and data formats such as XML, JSON, and JSONP with the help of [DataManager](#) adaptors.

Fields	Type	Description
----- ----- -----		
Text	<code>string</code>	Specifies the display text of each list item.
Value	<code>int</code> or <code>string</code>	Specifies the hidden data value mapped to each list item that should contain a unique value.
GroupBy	<code>string</code>	Specifies the category under which the list item has to be grouped.
IconCss	<code>string</code>	Specifies the icon class of each list item.

When binding complex data to the ComboBox, fields should be mapped correctly. Otherwise, the selected item remains undefined.

Binding local data

Local data can be represented in two ways as described below.

Array of JSON data

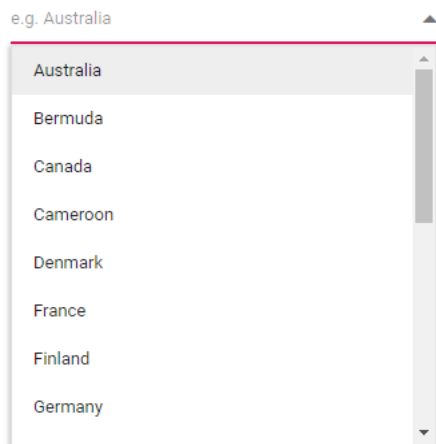
The ComboBox can generate its list items through an array of complex data. For this, the appropriate columns should be mapped to the [Fields](#) property.

In the following example, **Name** column from complex data have been mapped to the **Value** field.

ASPX-CS

```
@using Syncfusion.Blazor.DropDowns
<SfComboBox TValue="string" TItem="Countries" Placeholder="e.g. Australia"
DataSource="@Country">
<ComboBoxFieldSettings Text="Name" Value="Code"></ComboBoxFieldSettings>
</SfComboBox>
@code {
public class Countries
{
public string Name { get; set; }
public string Code { get; set; }
}
List<Countries> Country = new List<Countries>
{
new Countries() { Name = "Australia", Code = "AU" },
new Countries() { Name = "Bermuda", Code = "BM" },
new Countries() { Name = "Canada", Code = "CA" },
new Countries() { Name = "Cameroon", Code = "CM" },
new Countries() { Name = "Denmark", Code = "DK" },
new Countries() { Name = "France", Code = "FR" },
new Countries() { Name = "Finland", Code = "FI" },
new Countries() { Name = "Germany", Code = "DE" },
new Countries() { Name = "Greenland", Code = "GL" },
new Countries() { Name = "Hong Kong", Code = "HK" },
new Countries() { Name = "India", Code = "IN" },
new Countries() { Name = "Italy", Code = "IT" },
new Countries() { Name = "Japan", Code = "JP" },
new Countries() { Name = "Mexico", Code = "MX" },
new Countries() { Name = "Norway", Code = "NO" },
new Countries() { Name = "Poland", Code = "PL" },
new Countries() { Name = "Switzerland", Code = "CH" },
new Countries() { Name = "United Kingdom", Code = "GB" },
new Countries() { Name = "United States", Code = "US" },
};
}
```

The output will be as follows.



Array of Complex data

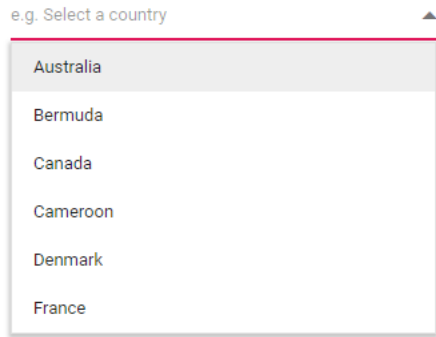
The ComboBox can generate its list items through an array of complex data. For this, the appropriate columns should be mapped to the [Fields](#) property.

In the following example, `Code.ID` column and `Country.CountryID` column from complex data have been mapped to the `Value` field and `Text` field, respectively.

ASPX-CS

```
@using Syncfusion.Blazor.DropDowns
<SfComboBox TValue="string" DataSource="@LocalData" TValue="string"
TItem="Complex" Placeholder="e.g. Select a Country">
<ComboBoxFieldSettings Text="Country.CountryID"
Value="Code.ID"></ComboBoxFieldSettings>
</SfComboBox>
@code {
public IEnumerable<Complex> LocalData { get; set; } = new
Complex().GetData();
public class Code
{
public string ID { get; set; }
}
public class Country
{
public string CountryID { get; set; }
}
public class Complex
{
public Country Country { get; set; }
public Code Code { get; set; }
public List<Complex>GetData() {
List<Complex>Data = new List<Complex>();
Data.Add(new Complex() { Country = new Country() { CountryID = "Australia"
}, Code = new Code() { ID = "AU" } });
Data.Add(new Complex() { Country = new Country() { CountryID = "Bermuda" },
Code = new Code() { ID = "BM" } });
Data.Add(new Complex() { Country = new Country() { CountryID = "Canada" },
Code = new Code() { ID = "CA" } });
Data.Add(new Complex() { Country = new Country() { CountryID = "Cameroon" },
Code = new Code() { ID = "CM" } });
Data.Add(new Complex() { Country = new Country() { CountryID = "Denmark" },
Code = new Code() { ID = "DK" } });
Data.Add(new Complex() { Country = new Country() { CountryID = "France" },
Code = new Code() { ID = "FR" } });
return Data;
}
}
}
```

The output will be as follows.



Binding remote data

The ComboBox supports retrieval of data from remote data services with the help of [DataManager](#) property is used to fetch data from the database and bind it to the ComboBox.

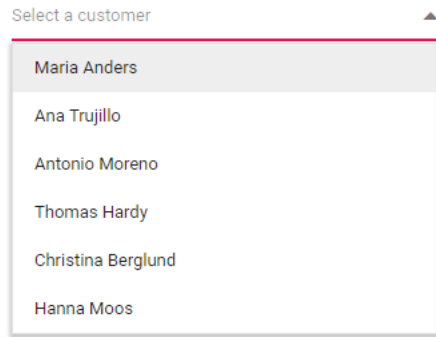
In the following sample, First 6 contacts are displayed from the **Customers** table of **Northwind** Data Service.

ASPX-CS

```
@using Syncfusion.Blazor.Data
@using Syncfusion.Blazor.DropDowns
<SfComboBox TValue="string" TItem="OrderDetails" Placeholder="Select a customer" Query="@Query">
  <SfDataManager
    Url="https://services.odata.org/V4/Northwind/Northwind.svc/Orders"
    Adaptor="Adaptors.ODataV4Adaptor" CrossDomain=true</SfDataManager>
  <ComboBoxFieldSettings Text="CustomerID"
    Value="OrderID"></ComboBoxFieldSettings>
</SfComboBox>

@code {
    public Query Query = new Query().Select(new List<string> { "CustomerID",
    "OrderID" }).Take(6).RequiresCount();
    public class OrderDetails
    {
        public int? OrderID { get; set; }
        public string CustomerID { get; set; }
        public int? EmployeeID { get; set; }
        public double? Freight { get; set; }
        public string ShipCity { get; set; }
        public bool Verified { get; set; }
        public DateTime? OrderDate { get; set; }
        public string ShipName { get; set; }
        public string ShipCountry { get; set; }
        public DateTime? ShippedDate { get; set; }
        public string ShipAddress { get; set; }
    }
}
```

The output will be as follows.



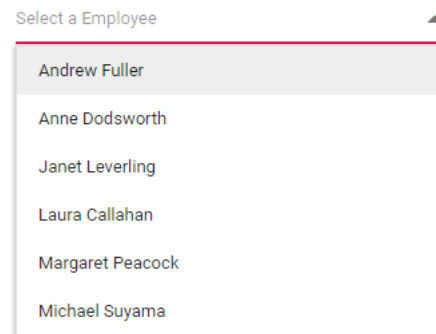
Web API Adaptor

Use the **WebApiAdaptor** to bind ComboBox with Web API created using OData.

ASPX-CS

```
@using Syncfusion.Blazor.Data
@using Syncfusion.Blazor.DropDowns
<SfComboBox TValue="string" TItem="EmployeeData" Placeholder="Select a
Employee" Query="@Query">
<SfDataManager Url="https://ej2services.syncfusion.com/production/web-
services/api/Employees" Adaptor="Adaptors.WebApiAdaptor"
CrossDomain=true></SfDataManager>
<ComboBoxFieldSettings Text="FirstName"
Value="EmployeeID"></ComboBoxFieldSettings>
</SfComboBox>
@code {
public Query Query = new Query();
public class EmployeeData
{
public int EmployeeID { get; set; }
public string FirstName { get; set; }
public string Designation { get; set; }
public string Country { get; set; }
}
}
```

Output will be like the below.



Custom Adaptor

The [SfDataManager](#) has custom adaptor support which allows you to perform manual operations on the data. This can be utilized for implementing custom data binding and editing operations in the ComboBox component.

For implementing custom data binding in ComboBox, the **DataAdaptor** class is used. This abstract class acts as a base class for the custom adaptor.

The **DataAdaptor** abstract class has both synchronous and asynchronous method signatures which can be overridden in the custom adaptor. Following are the method signatures present in this class,

CSHARP

```
public abstract class DataAdaptor
{
    /// <summary>
    /// Performs data Read operation synchronously.
    /// </summary>
    public virtual object Read(DataManagerRequest dataManagerRequest, string key
    = null)
    /// <summary>
    /// Performs data Read operation asynchronously.
    /// </summary>
    public virtual Task<object> ReadAsync(DataManagerRequest dataManagerRequest,
    string key = null)
}
```

The custom data binding can be performed in the ComboBox component by providing the custom adaptor class and overriding the Read or ReadAsync method of the DataAdaptor abstract class.

The following sample code demonstrates implementing custom data binding using custom adaptor,

ASPX-CS

```
<SfComboBox TValue="string" TItem="Orders">
<SfDataManager AdaptorInstance="@typeof(CustomAdaptor) "
Adaptor="Adaptors.CustomAdaptor"></SfDataManager>
<ComboBoxFieldSettings Value="CustomerID"></ComboBoxFieldSettings>
</SfComboBox>
@code{
public class Orders
{
    public Orders() { }
    public Orders(int OrderID, string CustomerID)
    {
        this.OrderID = OrderID;
        this.CustomerID = CustomerID;
    }
    public int OrderID { get; set; }
    public string CustomerID { get; set; }
}
public class CustomAdaptor : DataAdaptor
{
    static readonly HttpClient client = new HttpClient();
    public static List<OrdersDetails> order = OrdersDetails.GetAllRecords();
    public override object Read(DataManagerRequest dm, string key = null)
```

```
{
IEnumerable<OrdersDetails> DataSource = order;
if (dm.Search != null && dm.Search.Count > 0)
{
DataSource = DataOperations.PerformSearching(DataSource, dm.Search);
//Search
}
if (dm.Sorted != null && dm.Sorted.Count > 0) //Sorting
{
DataSource = DataOperations.PerformSorting(DataSource, dm.Sorted);
}
if (dm.Where != null && dm.Where.Count > 0) //Filtering
{
DataSource = DataOperations.PerformFiltering(DataSource, dm.Where,
dm.Where[0].Operator);
}
int count = DataSource.Cast<OrdersDetails>().Count();
if (dm.Skip != 0)
{
DataSource = DataOperations.PerformSkip(DataSource, dm.Skip);
//Paging
}
if (dm.Take != 0)
{
DataSource = DataOperations.PerformTake(DataSource, dm.Take);
}
return dm.RequiresCounts ? new DataResult() { Result = DataSource, Count =
count } : (object)DataSource;
}
}
}
```

Offline mode

To avoid post back for every action, set the ComboBox to load all data on initialization and make the actions process in client-side. To enable this behaviour, use the **Offline** property of [DataManager](#).

The following example for remote data binding and enabled offline mode,

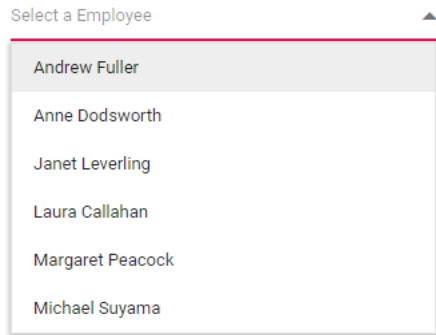
ASPX-CS

```
@using Syncfusion.Blazor.Data
@using Syncfusion.Blazor.DropDowns
<SfComboBox TValue="string" TItem="EmployeeData" Placeholder="Select a
Employee" Query="@Query">
<SfDataManager Url="https://ej2services.syncfusion.com/production/web-
services/api/Employees" Offline=true Adaptor="Adaptors.WebApiAdaptor"
CrossDomain=true></SfDataManager>
<ComboBoxFieldSettings Text="FirstName"
Value="EmployeeID"></ComboBoxFieldSettings>
</SfComboBox>
@code {
public Query Query = new Query();
public class EmployeeData
{
public int EmployeeID { get; set; }
public string FirstName { get; set; }
}
```



```
public string Designation { get; set; }
public string Country { get; set; }
}
```

The output will be as follows.



ValueTuple data binding

You can bind [ValueTuple](#) data to the ComboBox component. The following code helps you to get a string value from the enumeration data by using [ValueTuple](#).

C#

```
@using Syncfusion.Blazor.DropDowns;
<SfComboBox TItem="(DayOfWeek, string)" Width="250px" TValue="DayOfWeek"
DataSource="@ (Enum.GetValues<DayOfWeek>().Select(e => (e, e.ToString()))) ">
<ComboBoxFieldSettings Value="Item1" Text="Item2" />
</SfComboBox>
```

The output will shown as follows,



ValueTuple data binding

You can bind [ValueTuple](#) data to the ComboBox component. The following code helps you to get a string value from the enumeration data by using [ValueTuple](#).

C#

```
@using Syncfusion.Blazor.DropDowns;
<SfComboBox TItem="(DayOfWeek, string)" Width="250px" TValue="DayOfWeek"
DataSource="@ (Enum.GetValues<DayOfWeek>().Select(e => (e, e.ToString()))) ">
<ComboBoxFieldSettings Value="Item1" Text="Item2" />
</SfComboBox>
```

The output will shown as follows,



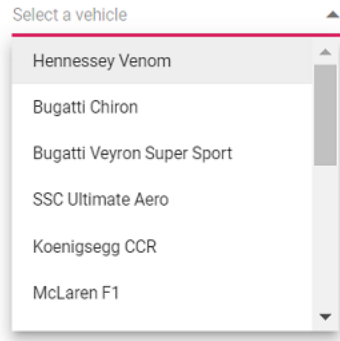
Binding ExpandoObject

You can bind [ExpandoObject](#) data to the ComboBox component. The following example `ExpandoObject` is bound to the collection of vehicles data.

CSHARP

```
@using Syncfusion.Blazor.DropDowns
@using System.Dynamic
<SfComboBox TItem="ExpandoObject" TValue="string" PopupHeight="230px"
Placeholder="Select a vehicle" DataSource="@VehicleData">
<ComboBoxFieldSettings Text="Text" Value="ID"></ComboBoxFieldSettings>
</SfComboBox>
@code{
    public List<ExpandoObject> VehicleData { get; set; } = new
    List<ExpandoObject>();
    protected override void OnInitialized()
    {
        VehicleData = Enumerable.Range(1, 15).Select((x) =>
        {
            dynamic d = new ExpandoObject();
            d.ID = (1000 + x).ToString();
            d.Text = (new string[] { "Hennessey Venom", "Bugatti Chiron", "Bugatti
            Veyron Super Sport", "SSC Ultimate Aero", "Koenigsegg CCR", "McLaren F1",
            "Aston Martin One- 77", "Jaguar XJ220", "McLaren P1", "Ferrari LaFerrari",
            "Mahindra Jaguar", "Hyundai Toyota", "Jeep Volkswagen", "Tata Maruti
            Suzuki", "Audi Mercedes Benz" }[x - 1]);
            return d;
        }).Cast<ExpandoObject>().ToList<ExpandoObject>();
    }
}
```

The output will shown as follows,



Binding DynamicObject

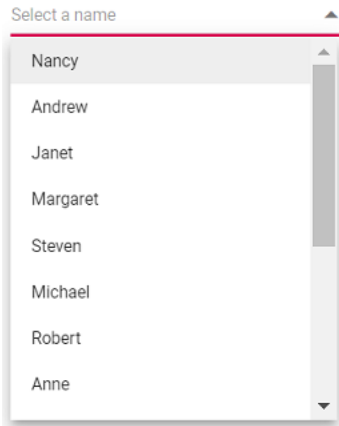
You can bind [DynamicObject](#) data to the ComboBox component. The following example `DynamicObject` is bound to the collection of customers data.

CSHARP

```
@using Syncfusion.Blazor.DropDowns
@using System.Dynamic
<SfComboBox TValue="string" TItem="DynamicDictionary" Placeholder="Select a
name" DataSource="@Orders">
<ComboBoxFieldSettings Text="CustomerName"
Value="CustomerName"></ComboBoxFieldSettings>
</SfComboBox>
@code{
public List<DynamicDictionary> Orders = new List<DynamicDictionary>() { };
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 15).Select((x) =>
{
dynamic d = new DynamicDictionary();
d.OrderID = 1000 + x;
d.CustomerName = (new string[] { "Nancy", "Andrew", "Janet", "Margaret",
"Steven", "Michael", "Robert", "Anne", "Nige", "Fuller", "Dodsworth",
"Leverling", "Callahan", "Suyama", "Davolio" }[x - 1]);
return d;
}).Cast<DynamicDictionary>().ToList<DynamicDictionary>();
}
public class DynamicDictionary : System.Dynamic.DynamicObject
{
Dictionary<string, object> dictionary = new Dictionary<string, object>();
public override bool TryGetMember(GetMemberBinder binder, out object result)
{
string name = binder.Name;
return dictionary.TryGetValue(name, out result);
}
public override bool TrySetMember(SetMemberBinder binder, object value)
{
dictionary[binder.Name] = value;
return true;
}
//The GetDynamicMemberNames method of DynamicObject class must be overridden
and return the property names to perform data operation and editing while
using DynamicObject.
public override System.Collections.Generic.IEnumerable<string>
GetDynamicMemberNames()
```

```
{
    return this.dictionary?.Keys;
}
}
```

The output will shown as follows,



Binding ObservableCollection

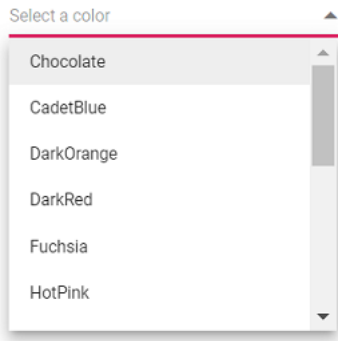
You can bind [ObservableCollection](#) data to the ComboBox component. The following example **Observable Data** is bound to a collection of colors data.

CSHARP

```
@using Syncfusion.Blazor.DropDowns
@using System.Collections.ObjectModel;
<SfComboBox TValue="string" TItem="Colors" PopupHeight="230px"
Placeholder="Select a color" DataSource="@ColorsData">
<ComboBoxFieldSettings Text="Color" Value="Code"></ComboBoxFieldSettings>
</SfComboBox>
@code{
    public class Colors
    {
        public string Code { get; set; }
        public string Color { get; set; }
    }
    private ObservableCollection<Colors> ColorsData = new
    ObservableCollection<Colors>()
    {
        new Colors() { Color = "Chocolate", Code = "#75523C" },
        new Colors() { Color = "CadetBlue", Code = "#3B8289" },
        new Colors() { Color = "DarkOrange", Code = "#FF843D" },
        new Colors() { Color = "DarkRed", Code = "#CA3832" },
        new Colors() { Color = "Fuchsia", Code = "#D44FA3" },
        new Colors() { Color = "HotPink", Code = "#F23F82" },
        new Colors() { Color = "Indigo", Code = "#2F5D81" },
        new Colors() { Color = "LimeGreen", Code = "#4CD242" },
        new Colors() { Color = "OrangeRed", Code = "#FE2A00" },
        new Colors() { Color = "Tomato", Code = "#FF745C" },
        new Colors() { Color = "Brown", Code = "#A52A2A" },
        new Colors() { Color = "Maroon", Code = "#800000" },
    }
}
```

```
new Colors() { Color = "Green", Code = "#008000" },  
new Colors() { Color = "Pink", Code = "#FFC0CB" },  
new Colors() { Color = "Purple", Code = "#800080" }  
};  
}
```

The output will shown as follows,



Entity Framework

You need to follow the below steps to consume data from the **Entity Framework** in the ComboBox component.

Create DbContext class

The first step is to create a DbContext class called **OrderContext** to connect to a Microsoft SQL Server database.

CSHARP

```
using Microsoft.EntityFrameworkCore;  
using System;  
using System.Collections.Generic;  
using System.Linq;  
using System.Threading.Tasks;  
using EFDropDown.Shared.Models;  
namespace EFDropDown.Shared.DataAccess  
{  
    public class OrderContext : DbContext  
    {  
        public virtual DbSet<Shared.Models.Order> Orders { get; set; }  
        protected override void OnConfiguring(DbContextOptionsBuilder  
optionsBuilder)  
        {  
            if (!optionsBuilder.IsConfigured)  
            {  
                optionsBuilder.UseSqlServer(@"Data  
Source=(LocalDB)\MSSQLLocalDB;AttachDbFilename=D:\Blazor\DropDownList\EFDrop  
Down\Shared\AppData\NORTHWND.MDF;Integrated Security=True;Connect  
Timeout=30");  
            }  
        }  
    }  
}
```

Create data access layer to perform data operation

Now, you need to create a class named **OrderDataAccessLayer**, which act as data access layer for retrieving the records from the database table.

CSHARP

```
using Microsoft.EntityFrameworkCore;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using EFDropDown.Shared.Models;
namespace EFDropDown.Shared.DataAccess
{
    public class OrderDataAccessLayer
    {
        OrderContext db = new OrderContext();
        //To Get all Orders details
        public DbSet<Order> GetAllOrders()
        {
            try
            {
                return db.Orders;
            }
            catch
            {
                throw;
            }
        }
    }
}
```

Creating Web API Controller

A Web API Controller has to be created which allows ComboBox directly to consume data from the Entity framework.

CSHARP

```
using EFDropDown.Shared.DataAccess;
using EFDropDown.Shared.Models;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Primitives;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using System.Web;
using Microsoft.AspNetCore.Http;
namespace EFDropDown.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    //TreeGrid
    public class DefaultController : ControllerBase
    {
        OrderDataAccessLayer db = new OrderDataAccessLayer();
    }
}
```

```
[HttpGet]
public object Get()
{
    IQueryable<Order> data = db.GetAllOrders().AsQueryable();
    var count = data.Count();
    var queryString = Request.Query;
    if (queryString.Keys.Contains("$inlinecount"))
    {
        StringValues Skip;
        StringValues Take;
        int skip = (queryString.TryGetValue("$skip", out Skip)) ?
        Convert.ToInt32(Skip[0]) : 0;
        int top = (queryString.TryGetValue("$top", out Take)) ?
        Convert.ToInt32(Take[0]) : data.Count();
        return new { Items = data.Skip(skip).Take(top), Count = count };
    }
    else
    {
        return data;
    }
}
}
```

Configure ComboBox component using Web API adaptor

Now, you can configure the ComboBox using the **'SfDataManager'** to interact with the created Web API and consume the data appropriately. To interact with web api, you need to use WebApiAdaptor.

ASPX-CS

```
@using Syncfusion.Blazor.Data
@using Syncfusion.Blazor.DropDowns
<SfComboBox TValue="string" TItem="Order" Placeholder="Select a Country">
<SfDataManager Url="api/Default" Adaptor="Adaptors.WebApiAdaptor"
CrossDomain="true"></SfDataManager>
<ComboBoxFieldSettings Text="ShipCountry"
Value="OrderID"></ComboBoxFieldSettings>
</SfComboBox>
@code{
public class Order
{
    public int? OrderID { get; set; }
    public string ShipCountry { get; set; }
}
}
```

Grouping in Blazor ComboBox Component

The ComboBox supports wrapping nested elements into a group based on different categories. The category of each list item can be mapped through the [GroupBy](#) field in the data table. The group header is displayed both as inline and fixed headers. The fixed group header content

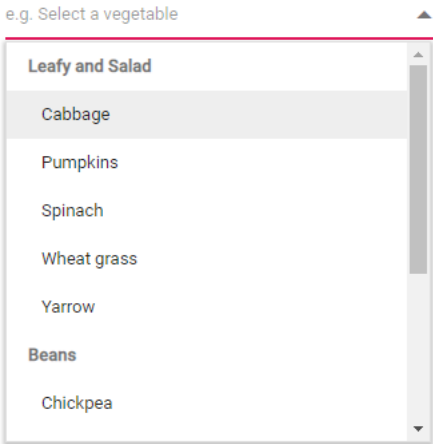
is updated dynamically on scrolling the popup list with its category value.

In the following sample, vegetables are grouped according on its category using **GroupBy** field.

ASPX-CS

```
@using Syncfusion.Blazor.Data
<SfComboBox TValue="string" TItem="Vegetables" Placeholder="e.g. Select a
vegetable" DataSource="@LocalData">
  <ComboBoxFieldSettings GroupBy="Category"
  Value="Vegetable"></ComboBoxFieldSettings>
</SfComboBox>
@code{
public IEnumerable<Vegetables> LocalData { get; set; } = new
Vegetables().VegetablesList();
public class Vegetables
{
public string Vegetable { get; set; }
public string Category { get; set; }
public string ID { get; set; }
public List<Vegetables>VegetablesList() {
List<Vegetables>Veg = new List<Vegetables>();
Veg.Add(new Vegetables { Vegetable = "Cabbage", Category= "Leafy and Salad",
ID= "item1" });
Veg.Add(new Vegetables { Vegetable = "Chickpea", Category= "Beans", ID=
"item2" });
Veg.Add(new Vegetables { Vegetable = "Garlic", Category= "Bulb and Stem",
ID= "item3" });
Veg.Add(new Vegetables { Vegetable = "Green bean", Category= "Beans", ID=
"item4" });
Veg.Add(new Vegetables { Vegetable = "Horse gram", Category= "Beans", ID=
"item5" });
Veg.Add(new Vegetables { Vegetable = "Nopal", Category= "Bulb and Stem", ID=
"item6" });
Veg.Add(new Vegetables { Vegetable = "Onion", Category= "Bulb and Stem", ID=
"item7" });
Veg.Add(new Vegetables { Vegetable = "Pumpkins", Category= "Leafy and
Salad", ID= "item8" });
Veg.Add(new Vegetables { Vegetable = "Spinach", Category= "Leafy and Salad",
ID= "item9" });
Veg.Add(new Vegetables { Vegetable = "Wheat grass", Category= "Leafy and
Salad", ID= "item10" });
Veg.Add(new Vegetables { Vegetable = "Yarrow", Category = "Leafy and Salad",
ID = "item11" });
return Veg;
}
}
```

The output will be as follows.



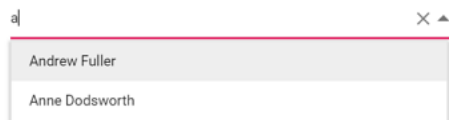
Filtering in Blazor ComboBox Component

The ComboBox has built-in support to filter data items when [AllowFiltering](#) is enabled. The filter operation starts as soon as you start typing characters in the component.

ASPX-CS

```
@using Syncfusion.Blazor.Data
@using Syncfusion.Blazor.DropDowns
<SfComboBox TValue="string" TItem="EmployeeData" Placeholder="Select a customer" Query="@Query" AllowFiltering=true>
<SfDataManager Url="https://ej2services.syncfusion.com/production/web-services/api/Employees" Adaptor="Adaptors.WebApiAdaptor"
CrossDomain=true></SfDataManager>
<ComboBoxFieldSettings Text="FirstName"
Value="EmployeeID"></ComboBoxFieldSettings>
</SfComboBox>
@code {
public Query Query = new Query();
public class EmployeeData
{
public int EmployeeID { get; set; }
public string FirstName { get; set; }
public string Designation { get; set; }
public string Country { get; set; }
}
}
```

The output will be as follows.



Custom Filtering

The ComboBox component filter queries can be customized. You can also use your own filter libraries to filter data like Fuzzy search.

ASPX-CS

```

@using Syncfusion.Blazor.Data
<SfComboBox TValue="string" @ref="comboObj" TItem="Countries"
Placeholder="e.g. Australia" DataSource="@Country" AllowFiltering="true">
<ComboBoxFieldSettings Text="Name" Value="Code"></ComboBoxFieldSettings>
<ComboBoxEvents TValue="string" TItem="Countries"
Filtering="OnFilter"></ComboBoxEvents>
</SfComboBox>
@code {
SfComboBox<string, Countries> comboObj { get; set; }
public class Countries
{
public string Name { get; set; }
public string Code { get; set; }
}
List<Countries> Country = new List<Countries>
{
new Countries() { Name = "Australia", Code = "AU" },
new Countries() { Name = "Bermuda", Code = "BM" },
new Countries() { Name = "Canada", Code = "CA" },
new Countries() { Name = "Cameroon", Code = "CM" },
new Countries() { Name = "Denmark", Code = "DK" }
};
List<Countries> Country1 = new List<Countries>
{
new Countries() { Name = "France", Code = "FR" },
new Countries() { Name = "Finland", Code = "FI" },
new Countries() { Name = "Germany", Code = "DE" },
new Countries() { Name = "Greenland", Code = "GL" }
};
private async Task OnFilter(FilteringEventArgs args)
{
args.PreventDefaultAction = true;
var query = new Query().Where(new WhereFilter() { Field = "Name", Operator =
"contains", value = args.Text, IgnoreCase = true });
query = !string.IsNullOrEmpty(args.Text) ? query : new Query();
await comboObj.FilterAsync(Country1, query);
}
}

```

Templates in Blazor ComboBox Component

The ComboBox has been provided with several options to customize each list item, group title, selected value, header, and footer elements.

Item template

The content of each list item within the ComboBox 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.

ASPX-CS

```

@using Syncfusion.Blazor.Data
@using Syncfusion.Blazor.DropDowns
<SfComboBox TValue="string" TItem="EmployeeData" Placeholder="Select a
customer" DataSource="@Data">
<ComboBoxTemplates TItem="EmployeeData">

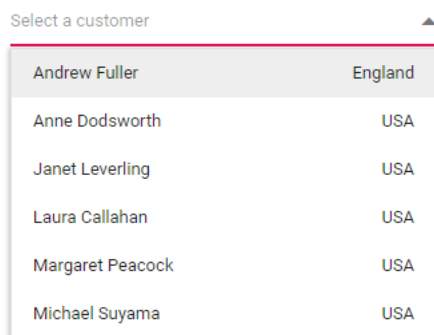
```

```

<ItemTemplate>
<span><span class='name'>@((context as EmployeeData).FirstName)</span><span
class='country'>@((context as EmployeeData).Country)</span></span>
</ItemTemplate>
</ComboBoxTemplates>
<ComboBoxFieldSettings Text="FirstName"
Value="Country"></ComboBoxFieldSettings>
</SfComboBox>
@code {
public class EmployeeData
{
public string FirstName { get; set; }
public string Country { get; set; }
}
List<EmployeeData> Data = new List<EmployeeData>
{
new EmployeeData() { FirstName = "Andrew Fuller", Country = "England" },
new EmployeeData() { FirstName = "Anne Dodsworth", Country = "USA" },
new EmployeeData() { FirstName = "Janet Leverling", Country = "USA" },
new EmployeeData() { FirstName = "Laura Callahan", Country = "USA"},
new EmployeeData() { FirstName = "Margaret Peacock", Country = "USA"},
new EmployeeData() { FirstName = "Michael Suyama", Country = "USA", },
new EmployeeData() { FirstName = "Nancy Davolio", Country = "USA"},
new EmployeeData() { FirstName = "Robert King", Country = "England"},
new EmployeeData() { FirstName = "Steven Buchanan", Country = "England"},
};
}
<style>
.country {
right: 15px;
position: absolute;
}
</style>

```

The output will be as follows.



Group template

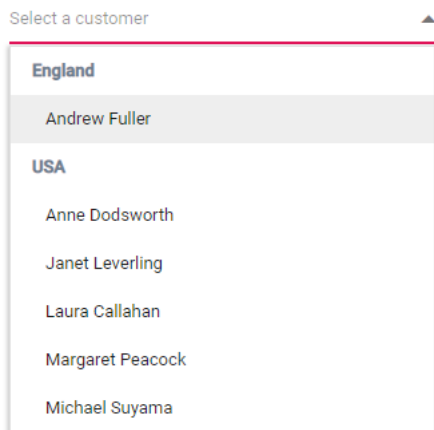
The group header title under which appropriate sub-items are categorized can also be customized with the help of [GroupTemplate](#) property. This template is common for both inline and floating group header template.

In the following sample, employees are grouped according to their country.

ASPX-CS

```
@using Syncfusion.Blazor.Data
@using Syncfusion.Blazor.DropDowns
<SfComboBox TValue="string" TItem="EmployeeData" Placeholder="Select a
customer" DataSource="@Data">
<ComboBoxTemplates TItem="EmployeeData">
<GroupTemplate>
<span class="group">@(context.Text)</span>
</GroupTemplate>
</ComboBoxTemplates>
<ComboBoxFieldSettings Value="FirstName"
GroupBy="Country"></ComboBoxFieldSettings>
</SfComboBox>
@code {
public class EmployeeData
{
public string FirstName { get; set; }
public string Country { get; set; }
}
List<EmployeeData> Data = new List<EmployeeData>
{
new EmployeeData() { FirstName = "Andrew Fuller", Country = "England" },
new EmployeeData() { FirstName = "Anne Dodsworth", Country = "USA" },
new EmployeeData() { FirstName = "Janet Leverling", Country = "USA" },
new EmployeeData() { FirstName = "Laura Callahan", Country = "USA"},
new EmployeeData() { FirstName = "Margaret Peacock", Country = "USA"},
new EmployeeData() { FirstName = "Michael Suyama", Country = "USA", },
new EmployeeData() { FirstName = "Nancy Davolio", Country = "USA"},
new EmployeeData() { FirstName = "Robert King", Country = "England"},
new EmployeeData() { FirstName = "Steven Buchanan", Country = "England"},
};
}
<style>
.group {
color: slategrey;
}
</style>
```

The output will be as follows.



Header template

The header element is shown statically at the top of the popup list items within the ComboBox, 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.

ASPX-CS

```
@using Syncfusion.Blazor.Data
@using Syncfusion.Blazor.DropDowns
<SfComboBox TValue="string" TItem="EmployeeData" Placeholder="Select a
customer" DataSource="@Data">
<ComboBoxTemplates TItem="EmployeeData">
<ItemTemplate>
<span class='item'><span class='name'>@((context as
EmployeeData).FirstName)</span><span class='city'>@((context as
EmployeeData).Country)</span></span>
</ItemTemplate>
<HeaderTemplate>
<span class='head'><span class='name'>Name</span><span
class='city'>Country</span></span>
</HeaderTemplate>
</ComboBoxTemplates>
<ComboBoxFieldSettings Value="Country"
Text="FirstName"></ComboBoxFieldSettings>
</SfComboBox>
@code {
public class EmployeeData
{
public string FirstName { get; set; }
public string Country { get; set; }
}
List<EmployeeData> Data = new List<EmployeeData>
{
new EmployeeData() { FirstName = "Andrew Fuller", Country = "England" },
new EmployeeData() { FirstName = "Anne Dodsworth", Country = "USA" },
new EmployeeData() { FirstName = "Janet Leverling", Country = "USA" },
new EmployeeData() { FirstName = "Laura Callahan", Country = "USA"},
new EmployeeData() { FirstName = "Margaret Peacock", Country = "USA"},
new EmployeeData() { FirstName = "Michael Suyama", Country = "USA", },
new EmployeeData() { FirstName = "Nancy Davolio", Country = "USA"},
new EmployeeData() { FirstName = "Robert King", Country = "England"},
new EmployeeData() { FirstName = "Steven Buchanan", Country = "England"},
};
}
<style>
.head, .item {
display: table;
width: 100%;
margin: auto;
}
.head {
height: 40px;
font-size: 15px;
font-weight: 600;
}
```

```
.name, .city {
display: table-cell;
vertical-align: middle;
width: 50%;
}
.head .name {
text-indent: 16px;
}
</style>
```

The output will be as follows.

Select a customer ▲

Name	Country
Andrew Fuller	England
Anne Dodsworth	USA
Janet Leverling	USA
Laura Callahan	USA
Margaret Peacock	USA
Michael Suyama	USA

Footer template

The ComboBox has options to show a footer element at the bottom of the list items in the popup list. Here, you can place any custom element as a footer element using the [FooterTemplate](#) property.

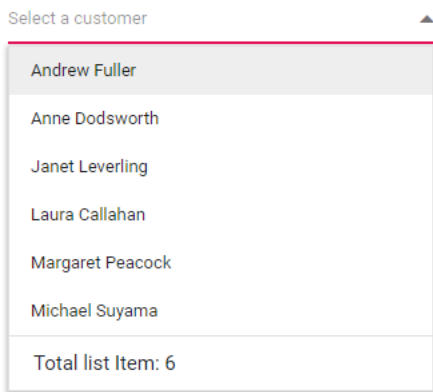
In the following sample, footer element displays the total number of list items present in the ComboBox.

ASPX-CS

```
@using Syncfusion.Blazor.Data
@using Syncfusion.Blazor.DropDowns
<SfComboBox TValue="string" TItem="EmployeeData" DataSource="@Data"
Placeholder="Select a customer">
  <ComboBoxTemplates TItem="EmployeeData">
    <FooterTemplate>
      <span class='footer'>Total list Item: 6 </span>
    </FooterTemplate>
  </ComboBoxTemplates>
  <ComboBoxFieldSettings Value="Country"
Text="FirstName"></ComboBoxFieldSettings>
</SfComboBox>
@code {
public class EmployeeData
{
public string FirstName { get; set; }
public string Country { get; set; }
}
List<EmployeeData> Data = new List<EmployeeData>
{
new EmployeeData() { FirstName = "Andrew Fuller", Country = "England" },
new EmployeeData() { FirstName = "Anne Dodsworth", Country = "USA" },
```

```
new EmployeeData() { FirstName = "Janet Leverling", Country = "USA" },
new EmployeeData() { FirstName = "Laura Callahan", Country = "USA"},
new EmployeeData() { FirstName = "Margaret Peacock", Country = "USA"},
new EmployeeData() { FirstName = "Michael Suyama", Country = "USA", },
};
}
<style>
.footer {
text-indent: 1.2em;
display: block;
font-size: 15px;
line-height: 40px;
border-top: 1px solid #e0e0e0;
}
</style>
```

The output will be as follows.



No records template

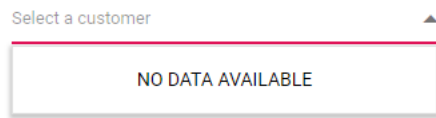
The ComboBox is provided with support to custom design the popup list content when no data is found and no matches found on search with the help of [NoRecordsTemplate](#) property.

In the following sample, popup list content displays the notification of no data available.

ASPX-CS

```
@using Syncfusion.Blazor.DropDowns
<SfComboBox TValue="string" TItem="Countries" Placeholder="Select a
customer" DataSource="@Country">
<ComboBoxTemplates TItem="Countries">
<NoRecordsTemplate>
<span class='norecord'> NO DATA AVAILABLE</span>
</NoRecordsTemplate>
</ComboBoxTemplates>
</SfComboBox>
@code {
public class EmployeeData { }
public EmployeeData Data = new EmployeeData();
public class Countries { }
List<Countries> Country = new List<Countries> { };
}
```

The output will be as follows.



Action failure template

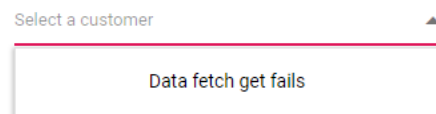
There is also an option to custom design the popup list content when the data fetch request fails at the remote server. This can be achieved using the [ActionFailureTemplate](#) property.

In the following sample, when the data fetch request fails, the ComboBox displays the notification.

ASPX-CS

```
@using Syncfusion.Blazor.Data
@using Syncfusion.Blazor.DropDowns
<SfComboBox TValue="string" TItem="EmployeeData" Placeholder="Select a customer" Query="@Query">
  <ComboBoxTemplates TItem="EmployeeData">
    <ActionFailureTemplate>
      <span class='norecord'>Data fetch get fails </span>
    </ActionFailureTemplate>
  </ComboBoxTemplates>
  <SfDataManager
    Url="https://services.odata.org/V4/Northwind/Northwind.svcs/Employees"
    Adaptor="Adaptors.ODataV4Adaptor" CrossDomain=true></SfDataManager>
  <ComboBoxFieldSettings Value="Country"
    Text="FirstName"></ComboBoxFieldSettings>
</SfComboBox>
@code {
  public class EmployeeData
  {
    public string FirstName { get; set; }
  }
  public EmployeeData Data = new EmployeeData();
  public Query Query = new Query().Select(new List<string> { "FirstName", "Country" }).Take(6).RequiresCount();
}
```

The output will be as follows.



Localization in Blazor ComboBox Component

Blazor server side

Add `UseRequestLocalization` middle-ware in Configure method in **Startup.cs** file to get browser Culture Info.

Refer the following code to add configuration in Startup.cs file

C#

```
using Microsoft.AspNetCore.Builder;
```



```
using Microsoft.AspNetCore.Localization;
namespace BlazorApplication
{
    public class Startup
    {
        ....
        ....
        public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
        {
            app.UseRequestLocalization();
            ....
            ....
        }
    }
}
```

The **Localization** library allows you to localize default text content. The ComboBox component has static text that can be changed to other cultures (Arabic, Deutsch, French, etc.).

The following example demonstrates how to enable **Localization** for ComboBox in server side Blazor samples. Here, we have used Resource file to translate the static text.

The Resource file is an XML file which contains the strings(key and value pairs) that you want to translate into different language. You can also refer [Localization link](#) to know more about how to configure and use localization in the ASP.NET Core application framework.

- Open the **Startup.cs** file and add the below configuration in the **ConfigureServices** function as follows.

CSHARP

```
using Syncfusion.Blazor;
using System.Globalization;
using Microsoft.AspNetCore.Localization;
namespace BlazorApplication
{
    public class Startup
    {
        ....
        ....
        public void ConfigureServices(IServiceCollection services)
        {
            ....
            ....
            services.AddSyncfusionBlazor();
            services.AddLocalization(options => options.ResourcesPath = "Resources");
            services.Configure<RequestLocalizationOptions>(options =>
            {
                // define the list of cultures your app will support
                var supportedCultures = new List<CultureInfo>()
                {
                    new CultureInfo("de")
                };
                // set the default culture
                options.DefaultRequestCulture = new RequestCulture("de");
            });
        }
    }
}
```

```
options.SupportedCultures = supportedCultures;
options.SupportedUICultures = supportedCultures;
options.RequestCultureProviders = new List<IRequestCultureProvider>() {
    new QueryStringRequestCultureProvider() // Here, You can also use other
    localization provider
};
});
services.AddSingleton(typeof(ISyncfusionStringLocalizer),
    typeof(SampleLocalizer));
}
}
```

- Then, write a **class** by inheriting **ISyncfusionStringLocalizer** interface and override the **Manager** property to get the resource file details from the application end.

CSHARP

```
using Syncfusion.Blazor;
namespace blazorDropdowns
{
    public class SampleLocalizer : ISyncfusionStringLocalizer
    {
        public string Get(string key)
        {
            return this.Manager.GetString(key);
        }
        public System.Resources.ResourceManager Manager
        {
            get
            {
                return blazorDropdowns.Resources.SyncfusionBlazorLocale.ResourceManager;
            }
        }
    }
}
```

- Add **.resx** file to Resource folder and enter the key value (Locale Keywords) in the **Name** column and the translated string in the Value column as follows.

Name	Value (in Deutsch culture)
---	---
ComboBox_ActionFailureTemplate	Die Anfrage ist fehlgeschlagen
ComboBox_NoRecordsTemplate	Keine Aufzeichnungen gefunden

- Finally, Specify the culture for ComboBox using **locale** property.

ASPX-CS

```
@using Syncfusion.Blazor.DropDowns
```

```

<SfComboBox TValue="string" TItem="Games" Placeholder="Select a game"
Locale="de" AllowFiltering="true" DataSource="@LocalData">
  <ComboBoxFieldSettings Value="ID" Text="Text"></ComboBoxFieldSettings>
</SfComboBox>
@code {
public class Games
{
public string ID { get; set; }
public string Text { get; set; }
}
List<Games> LocalData = new List<Games> {
new Games() { ID= "Game1", Text= "American Football" },
new Games() { ID= "Game2", Text= "Badminton" },
new Games() { ID= "Game3", Text= "Basketball" },
new Games() { ID= "Game4", Text= "Cricket" },
new Games() { ID= "Game5", Text= "Football" },
new Games() { ID= "Game6", Text= "Golf" }
};
}

```

Blazor WebAssembly

The Localization library allows you to localize the static text content of the [NoRecordsTemplate](#) and [ActionFailureTemplate](#) properties according to the culture currently assigned to the ComboBox.

| Locale key | en-US (default)

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

| NoRecordsTemplate | No records found

| ActionFailureTemplate | The request failed

The following steps explain how to render the ComboBox in French culture (fr) in Blazor Web Assembly application.

- Open the **program.cs** file and add the below configuration in the **Builder ConfigureServices** function as follows.

CSHARP

```

using Syncfusion.Blazor;
using Microsoft.AspNetCore.Builder;
namespace WebAssemblyLocale
{
public class Program
{
public static async Task Main(string[] args)
{
....
....
builder.Services.Configure<RequestLocalizationOptions>(options =>
{
// Define the list of cultures your app will support
var supportedCultures = new List<System.Globalization.CultureInfo>()
{
new System.Globalization.CultureInfo("en-US"),

```

```
new System.Globalization.CultureInfo("fr"),
};
// Set the default culture
options.DefaultRequestCulture = new
Microsoft.AspNetCore.Localization.RequestCulture("fr");
options.SupportedCultures = supportedCultures;
options.SupportedUICultures = supportedCultures;
options.RequestCultureProviders = new
List<Microsoft.AspNetCore.Localization.IRequestCultureProvider>() {
new Microsoft.AspNetCore.Localization.QueryStringRequestCultureProvider()
};
});
....
....
}
}
}
```

- To download the locale definition of Blazor components, use this [link](#).
- After downloading the blazor-locale package, copy the blazor-locale folder with required local definition file into wwwroot folder.
- By default, the blazor-locale package contains the localized text for static text present in components like button text, placeholder, tooltip, and more.
- Set the culture by using the SetCulture method.

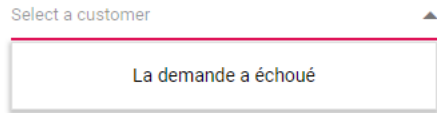
In the following sample, French culture is set to the ComboBox and no data is loaded. Hence, the [NoRecordsTemplate](#) property displays its text in French culture initially, and if the sample is run offline, the [ActionFailureTemplate](#) property displays its text appropriately.

ASPX-CS

```
@using Syncfusion.Blazor.Data
@using Syncfusion.Blazor.DropDowns
@inject HttpClient Http;
<SfComboBox TValue="string" TItem="EmployeeData" Query="@Query"
Placeholder="Select a customer" Locale="fr" AllowFiltering="true">
<SfDataManager Url="https://ej2services.syncfusion.com/production/web-
services/api/Employees" Adaptor="Adaptors.WebApiAdaptor"
CrossDomain=true></SfDataManager>
<ComboBoxFieldSettings Value="Country"
Text="FirstName"></ComboBoxFieldSettings>
</SfComboBox>
@code {
[Inject]
protected IJSRuntime JsRuntime { get; set; }
protected override async Task OnInitializedAsync()
{
this.JsRuntime.Sf().LoadLocaleData(await Http.GetJsonAsync<object>("blazor-
locale/src/fr.json")).SetCulture("fr");
}
public Query Query = new Query();
public class EmployeeData
{
public int EmployeeID { get; set; }
}
```

```
public string FirstName { get; set; }
public string Designation { get; set; }
}
```

The output will be as follows.



Style and appearance in Blazor ComboBox 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-icon.e-ddl-icon.e-icons, .e-ddl .e-input-group-icon.e-ddl-icon.e-icons: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-lib.e-input-group.e-control-wrapper.e-control-container.e-float-input.e-input-focus .e-float-text.e-label-top {
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 placeholder to add mandatory indicator(*)

Use the following CSS to add the mandatory indicator * to the float label element.

CSS

```
.e-input-group.e-control-wrapper.e-control-container.e-float-input .e-float-text::after {
```

```
content: "*";
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;
}
```

Virtualization in Blazor ComboBox Component

The ComboBox has been provided virtualization to improve the UI performance for a large amount of data when [EnableVirtualization](#) is true. This feature doesn't render out the entire data source on initial component rendering. It loads the N number of items in the popup on initial rendering and the remaining set number of items will load on each scrolling action in the popup. It can work with both local and remote data.

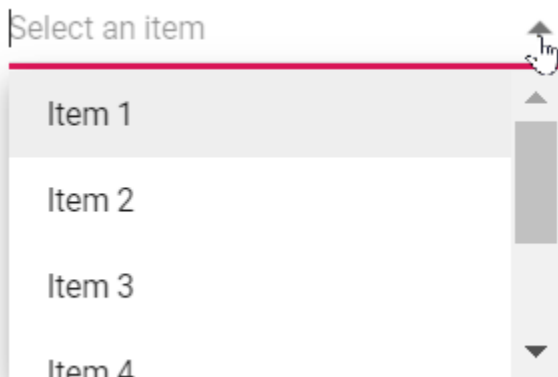
In the following code 150 items bound to the component, but only 5 items will load to the popup when you open the popup. Remaining set number of items will load on each scrolling action in the popup.

ASPX-CS

```
@using Syncfusion.Blazor.DropDowns
@using Syncfusion.Blazor.Data
```

```
<SfComboBox TValue="string" TItem="Record" Placeholder="Select an item"
DataSource="@Records" Query="@LocalDataQuery" PopupHeight="130px"
EnableVirtualization="true">
<ComboBoxFieldSettings Text="Text" Value="ID"></ComboBoxFieldSettings>
</SfComboBox>
@code{
public Query LocalDataQuery = new Query().Take(6);
public class Record
{
public string ID { get; set; }
public string Text { get; set; }
}
public List<Record> Records { get; set; }
protected override void OnInitialized()
{
this.Records = Enumerable.Range(1, 150).Select(i => new Record()
{
ID = i.ToString(),
Text = "Item " + i,
}).ToList();
}
}
```

The output will shown as follows,



Adding Custom Value to Blazor ComboBox Component

You can add custom value to the ComboBox component. When the typed character(s) is not present in the list, a button will be shown in the popup list. By clicking on this button, the custom value character(s) get added to the existing list as a new item.

ASPX-CS

```
@using Syncfusion.Blazor.DropDowns;
@using Syncfusion.Blazor.Buttons;
@using Syncfusion.Blazor.Data;
<SfComboBox @ref="@ComboObj" TValue="string" Placeholder="Select a Country"
Width="250px" TItem="TItem" DataSource="@DataSource" AllowCustom="true"
AllowFiltering="true">
<ComboBoxFieldSettings Text="Name" Value="Name"></ComboBoxFieldSettings>
```



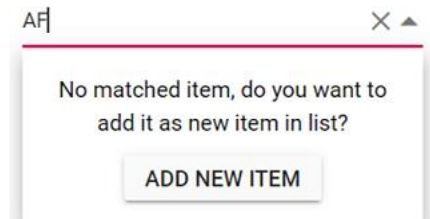
```

<ComboBoxEvents TValue="string" Filtering="@OnFiltering"
TItem="TItem"></ComboBoxEvents>
<ComboBoxTemplates TItem="TItem">
<NoRecordsTemplate>
<div>
<div id="nodata">No matched item, do you want to add it as new item in
list?</div>
<SfButton id="btn" class="e-control e-btn" style="margin-top: 10px"
@onclick="@AddItem">ADD NEW ITEM</SfButton>
</div>
</NoRecordsTemplate>
</ComboBoxTemplates>
</SfComboBox>
@code {
private SfComboBox<string, TItem> ComboObj;
private string Custom { get; set; }
private List<TItem> CustomDataItems { get; set; } = new List<TItem>();
public class TItem
{
public string Name { get; set; }
public string Code { get; set; }
}
private async Task
OnFiltering(Syncfusion.Blazor.DropDowns.FilteringEventArgs args)
{
Custom = args.Text;
args.PreventDefaultAction = true;
var query = new Query().Where(new WhereFilter() { Field = "Name", Operator =
"contains", value = args.Text, IgnoreCase = true });
query = !string.IsNullOrEmpty(args.Text) ? query : new Query();
await ComboObj.FilterAsync(CustomDataItems, query);
}
private List<TItem> DataSource = new List<TItem>
{
new TItem() { Name = "Australia", Code = "AU" },
new TItem() { Name = "Bermuda", Code = "BM" },
new TItem() { Name = "Canada", Code = "CA" },
new TItem() { Name = "Cameroon", Code = "CM" },
new TItem() { Name = "Denmark", Code = "DK" },
new TItem() { Name = "France", Code = "FR" },
new TItem() { Name = "Finland", Code = "FI" },
new TItem() { Name = "Germany", Code = "DE" },
new TItem() { Name = "Greenland", Code = "GL" },
new TItem() { Name = "Hong Kong", Code = "HK" },
new TItem() { Name = "India", Code = "IN" },
new TItem() { Name = "Italy", Code = "IT" }
};
protected override void OnInitialized()
{
CustomDataItems = DataSource;
}
private async Task AddItem()
{
var customData = new TItem() { Name = Custom, Code = Custom };
await this.ComboObj.AddItems(new List<TItem> { customData });
CustomDataItems.Add(customData);
await this.ComboObj.HidePopupAsync();
}

```

```
}
}
```

The output will be as follows.



Native Events in Blazor ComboBox Component

The following section explains the steps to include native events and pass data to event handler in ComboBox component.

Bind native events to ComboBox

You can access any native event by using on `<event>` attribute with a component. The attribute's value is treated as an event handler.

In the following example, the `keyPressed` method is called every time the key is pressed on input.

ASPX-CS

```
@using Syncfusion.Blazor.Data
<SfComboBox TValue="string" TItem="Countries" @onkeypress="@KeyPressed"
DataSource="@Country">
<ComboBoxFieldSettings Text="Name" Value="Code"></ComboBoxFieldSettings>
</SfComboBox>
@code {
public void KeyPressed()
{
Console.WriteLine("Key Pressed!");
}
public class Countries
{
public string Name { get; set; }
public string Code { get; set; }
}
List<Countries> Country = new List<Countries>
{
new Countries() { Name = "Australia", Code = "AU" },
new Countries() { Name = "Bermuda", Code = "BM" },
new Countries() { Name = "Canada", Code = "CA" },
new Countries() { Name = "Cameroon", Code = "CM" },
new Countries() { Name = "Denmark", Code = "DK" },
new Countries() { Name = "France", Code = "FR" },
};
}
```

Also, you can rewrite the previous code example as follows using Lambda expressions.

ASPX-CS

```
<SfComboBox TValue="string" @onkeypress="@(() => Console.WriteLine("Key Pressed!"))"></SfComboBox>
```

Pass event data to event handler

Blazor provides set of argument types to map to native events. The list of event types and event arguments are:

- Focus Events - FocusEventArgs
- Mouse Events - MouseEventArgs
- Keyboard Events - KeyboardEventArgs
- Input Events - ChangeEventArgs/EventArgs
- Touch Events – TouchEventArgs
- Pointer Events – PointerEventArgs

In the following example, the keyPressed method is called every time any key is pressed inside input. But the message will print when you press "c" key.

ASPX-CS

```
<SfComboBox TValue="string" Titem="Countries" @onkeypress="@ (e => KeyPressed(e))" DataSource="@Country">
<ComboBoxFieldSettings Text="Name" Value="Code"></ComboBoxFieldSettings>
</SfComboBox>
@code {
public void KeyPressed(KeyboardEventArgs args)
{
if (args.Key == "c")
{
Console.WriteLine("C was pressed");
}
}
public class Countries
{
public string Name { get; set; }
public string Code { get; set; }
}
List<Countries> Country = new List<Countries>
{
new Countries() { Name = "Australia", Code = "AU" },
new Countries() { Name = "Bermuda", Code = "BM" },
new Countries() { Name = "Canada", Code = "CA" },
new Countries() { Name = "Cameroon", Code = "CM" },
new Countries() { Name = "Denmark", Code = "DK" },
new Countries() { Name = "France", Code = "FR" },
};
}
```

Using Lambda expression also, you can pass the event data to the event handler.

List of Native events supported

| List of Native events | | |

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

onclick	onblur	onfocus	onfocusout	
onmousemove	onmouseover	onmouseout	onmousedown	onmouseup
ondblclick	onkeydown	onkeyup	onkeypress	
ontouchend	onfocusin	onmouseup	ontouchstart	

Accessibility in Blazor ComboBox Component

The ComboBox component has been designed with the WAI-ARIA specifications in mind, 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.

ARIA attributes

The ComboBox component uses the **combobox** role, and each list item has an **option** role. The following **ARIA attributes** denotes the ComboBox state:

Properties	Functionalities	
-----	-----	-----

aria-haspopup	Indicates whether the ComboBox input element has a popup list or not.	
aria-expanded	Indicates whether the popup list has expanded or not.	
aria-selected	Indicates the selected option.	
aria-readonly	Indicates the readonly state of the ComboBox element.	
aria-disabled	Indicates whether the ComboBox component is in a disabled state or not.	
aria-activedescendent	This attribute holds the ID of the active list item to focus its descendant child element.	
aria-owns	This attribute contains the ID of the popup list to indicate popup as a child element.	
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 ComboBox without interruptions:

Keyboard shortcuts	Actions
-----	-----

| Arrow Down | Selects the first item in the ComboBox when no item is selected. Otherwise, selects the item next to the currently selected item. |

| Arrow Up | Selects the item previous to the currently selected one. |

| Page Down | Scrolls down to the next page and selects the first item when popup list opens. |

| Page Up | Scrolls up to the previous page and selects the first item when popup list opens. |

| Enter | Selects the focused item and popup list closes when it is in open state. |

| Tab | Focuses on the next TabIndex element on the page when the popup is closed. Otherwise, closes the popup list and remains the focus of the component. |

| Shift + tab | Focuses on the previous TabIndex element on the page when the popup is closed. Otherwise, closes the popup list and remains the focus of the component. |

| Alt + Down | Open the popup list |

| Alt + Up | Closes the popup list |

| Esc(Escape) | Closes the popup list when it is in an open state and the currently selected item remains the same. |

| Home | Cursor moves before the first character in input |

| End | Cursor moves next to the last character in input |

In the following sample, disable the ComboBox component using t keys.

ASPX-CS

```
@using Syncfusion.Blazor.DropDowns
<SfComboBox TValue="string" TItem="Countries" @ref="ComboObj"
Enabled="@enable" Placeholder="Select a country" @onkeypress="@ (e =>
KeyPressed(e)) " DataSource="@LocalData">
<ComboBoxFieldSettings Value="Code" Text="Name"></ComboBoxFieldSettings>
</SfComboBox>
@code {
public SfComboBox<string, Countries> ComboObj;
public bool enable { get; set; } = true;
public class Countries
{
public string Name { get; set; }
public string Code { get; set; }
}
List<Countries> LocalData = new List<Countries> {
new Countries() { Name = "Australia", Code = "AU" },
```

```
new Countries() { Name = "Bermuda", Code = "BM" },
new Countries() { Name = "Canada", Code = "CA" },
new Countries() { Name = "Cameroon", Code = "CM" },
new Countries() { Name = "Denmark", Code = "DK" },
new Countries() { Name = "France", Code = "FR" },
new Countries() { Name = "Finland", Code = "FI" },
new Countries() { Name = "Germany", Code = "DE" },
new Countries() { Name = "Greenland", Code = "GL" },
new Countries() { Name = "Hong Kong", Code = "HK" },
};
public void KeyPressed(KeyboardEventArgs args)
{
    if (args.Key == "t")
    {
        enable = false;
    }
}
```

The output will be as follows.

t

Events in Blazor ComboBox Component

This section explains the list of events of the ComboBox component which will be triggered for appropriate ComboBox actions.

Blur

Blur event triggers when the input loses focus.

ASPX-CS

```
@using Syncfusion.Blazor.DropDowns
<SfDropDownList TItem="GameFields" TValue="string" DataSource="@Games">
    <DropDownListEvents TItem="GameFields" TValue="string"
    Blur="@BlurHandler"></DropDownListEvents>
    <DropDownListFieldSettings Text="Text"
    Value="ID"></DropDownListFieldSettings>
</SfDropDownList>
@code {
    public class GameFields
    {
        public string ID { get; set; }
        public string Text { get; set; }
    }
    private List<GameFields> Games = new List<GameFields>() {
        new GameFields() { ID= "Game1", Text= "American Football" },
        new GameFields() { ID= "Game2", Text= "Badminton" },
        new GameFields() { ID= "Game3", Text= "Basketball" },
        new GameFields() { ID= "Game4", Text= "Cricket" },
    };
    private void BlurHandler(Object args)
    {
        // Here you can customize your code
    }
}
```

```
}
```

ValueChanged

ValueChanged event triggers when the ComboBox value is changed.

ASPX-CS

```
@using Syncfusion.Blazor.DropDowns
<SfComboBox TItem="GameFields" TValue="string" DataSource="@Games">
  <ComboBoxEvents TItem="GameFields" TValue="string"
    ValueChange="@ValueChangeHandler"></ComboBoxEvents>
  <ComboBoxFieldSettings Text="Text" Value="ID"></ComboBoxFieldSettings>
</SfComboBox>
@code {
  public class GameFields
  {
    public string ID { get; set; }
    public string Text { get; set; }
  }
  private List<GameFields> Games = new List<GameFields>() {
    new GameFields() { ID= "Game1", Text= "American Football" },
    new GameFields() { ID= "Game2", Text= "Badminton" },
    new GameFields() { ID= "Game3", Text= "Basketball" },
    new GameFields() { ID= "Game4", Text= "Cricket" },
  };
  private void ValueChangeHandler(ChangeEventArgs<string, GameFields> args)
  {
    // Here you can customize your code
  }
}
```

Closed

Closed event triggers after the popup has been closed.

ASPX-CS

```
@using Syncfusion.Blazor.DropDowns
<SfComboBox TItem="GameFields" TValue="string" DataSource="@Games">
  <ComboBoxEvents TItem="GameFields" TValue="string"
    Closed="@CloseHandler"></ComboBoxEvents>
  <ComboBoxFieldSettings Text="Text" Value="ID"></ComboBoxFieldSettings>
</SfComboBox>
@code {
  public class GameFields
  {
    public string ID { get; set; }
    public string Text { get; set; }
  }
  private List<GameFields> Games = new List<GameFields>() {
    new GameFields() { ID= "Game1", Text= "American Football" },
    new GameFields() { ID= "Game2", Text= "Badminton" },
    new GameFields() { ID= "Game3", Text= "Basketball" },
    new GameFields() { ID= "Game4", Text= "Cricket" },
  };
  private void CloseHandler(ClosedEventArgs args)
  {
  }
}
```

```
{  
// Here you can customize your code  
}
```

Created

Created event triggers when the component is created.

ASPX-CS

```
@using Syncfusion.Blazor.DropDowns  
<SfComboBox TItem="GameFields" TValue="string" DataSource="@Games">  
  <ComboBoxEvents TItem="GameFields" TValue="string"  
    Created="@CreatedHandler"></ComboBoxEvents>  
  <ComboBoxFieldSettings Text="Text" Value="ID"></ComboBoxFieldSettings>  
</SfComboBox>  
@code {  
public class GameFields  
{  
    public string ID { get; set; }  
    public string Text { get; set; }  
}  
private List<GameFields> Games = new List<GameFields>() {  
    new GameFields(){ ID= "Game1", Text= "American Football" },  
    new GameFields(){ ID= "Game2", Text= "Badminton" },  
    new GameFields(){ ID= "Game3", Text= "Basketball" },  
    new GameFields(){ ID= "Game4", Text= "Cricket" },  
};  
private void CreatedHandler(Object args)  
{  
    // Here you can customize your code  
}
```

Destroyed

Destroyed event triggers when the component is destroyed.

ASPX-CS

```
@using Syncfusion.Blazor.DropDowns  
<SfComboBox TItem="GameFields" TValue="string" DataSource="@Games">  
  <ComboBoxEvents TItem="GameFields" TValue="string"  
    Destroyed="@DestroyedHandler"></ComboBoxEvents>  
  <ComboBoxFieldSettings Text="Text" Value="ID"></ComboBoxFieldSettings>  
</SfComboBox>  
@code {  
public class GameFields  
{  
    public string ID { get; set; }  
    public string Text { get; set; }  
}  
private List<GameFields> Games = new List<GameFields>() {  
    new GameFields(){ ID= "Game1", Text= "American Football" },  
    new GameFields(){ ID= "Game2", Text= "Badminton" },  
    new GameFields(){ ID= "Game3", Text= "Basketball" },
```



```
new GameFields(){ ID= "Game4", Text= "Cricket" },
};
private void DestroyedHandler(Object args)
{
// Here you can customize your code
}
}
```

Focus

Focus event triggers when the input gets focus.

ASPX-CS

```
@using Syncfusion.Blazor.DropDowns
<SfComboBox TItem="GameFields" TValue="string" DataSource="@Games">
  <ComboBoxEvents TItem="GameFields" TValue="string"
    Focus="@FocusHandler"></ComboBoxEvents>
  <ComboBoxFieldSettings Text="Text" Value="ID"></ComboBoxFieldSettings>
</SfComboBox>
@code {
public class GameFields
{
public string ID { get; set; }
public string Text { get; set; }
}
private List<GameFields> Games = new List<GameFields>() {
new GameFields(){ ID= "Game1", Text= "American Football" },
new GameFields(){ ID= "Game2", Text= "Badminton" },
new GameFields(){ ID= "Game3", Text= "Basketball" },
new GameFields(){ ID= "Game4", Text= "Cricket" },
};
private void FocusHandler(Object args)
{
// Here you can customize your code
}
}
```

OnOpen

OnOpen event triggers when the popup is opened. If you cancel this event, the popup remains closed.

ASPX-CS

```
@using Syncfusion.Blazor.DropDowns
<SfComboBox TItem="GameFields" TValue="string" DataSource="@Games">
  <ComboBoxEvents TItem="GameFields" TValue="string"
    OnOpen="@OnOpenHandler"></ComboBoxEvents>
  <ComboBoxFieldSettings Text="Text" Value="ID"></ComboBoxFieldSettings>
</SfComboBox>
@code {
public class GameFields
{
public string ID { get; set; }
public string Text { get; set; }
}
private List<GameFields> Games = new List<GameFields>() {
```

```
new GameFields(){ ID= "Game1", Text= "American Football" },
new GameFields(){ ID= "Game2", Text= "Badminton" },
new GameFields(){ ID= "Game3", Text= "Basketball" },
new GameFields(){ ID= "Game4", Text= "Cricket" },
};
private void OnOpenHandler(BeforeOpenEventArgs args)
{
    // Here you can customize your code
}
}
```

OnClose

OnClose event triggers before the popup is closed. If you cancel this event, the popup will remain open.

ASPX-CS

```
@using Syncfusion.Blazor.DropDowns
<SfComboBox TItem="GameFields" TValue="string" DataSource="@Games">
<ComboBoxEvents TItem="GameFields" TValue="string"
OnClose="@OnCloseHandler"></ComboBoxEvents>
<ComboBoxFieldSettings Text="Text" Value="ID"></ComboBoxFieldSettings>
</SfComboBox>
@code {
public class GameFields
{
public string ID { get; set; }
public string Text { get; set; }
}
private List<GameFields> Games = new List<GameFields>() {
new GameFields(){ ID= "Game1", Text= "American Football" },
new GameFields(){ ID= "Game2", Text= "Badminton" },
new GameFields(){ ID= "Game3", Text= "Basketball" },
new GameFields(){ ID= "Game4", Text= "Cricket" },
};
private void OnCloseHandler(PopupEventArgs args)
{
    // Here you can customize your code
}
}
```

DataBound

DataBound event triggers when the data source is populated in the popup list.

ASPX-CS

```
@using Syncfusion.Blazor.DropDowns
<SfComboBox TItem="GameFields" TValue="string" DataSource="@Games">
<ComboBoxEvents TItem="GameFields" TValue="string"
DataBound="@DataBoundHandler"></ComboBoxEvents>
<ComboBoxFieldSettings Text="Text" Value="ID"></ComboBoxFieldSettings>
</SfComboBox>
@code {
public class GameFields
{
public string ID { get; set; }
}
```

```
public string Text { get; set; }
}
private List<GameFields> Games = new List<GameFields>() {
new GameFields(){ ID= "Game1", Text= "American Football" },
new GameFields(){ ID= "Game2", Text= "Badminton" },
new GameFields(){ ID= "Game3", Text= "Basketball" },
new GameFields(){ ID= "Game4", Text= "Cricket" },
};
private void DataBoundHandler(DataBoundEventArgs args)
{
// Here you can customize your code
}
}
```

Filtering

Filtering event triggers on typing a character in the filter bar when the AllowFiltering is enabled.

ASPX-CS

```
@using Syncfusion.Blazor.DropDowns
<SfComboBox TItem="GameFields" TValue="string" AllowFiltering="true"
DataSource="@Games">
<ComboBoxEvents TItem="GameFields" TValue="string"
Filtering="@Filteringhandler"></ComboBoxEvents>
<ComboBoxFieldSettings Text="Text" Value="ID"></ComboBoxFieldSettings>
</SfComboBox>
@code {
public class GameFields
{
public string ID { get; set; }
public string Text { get; set; }
}
private List<GameFields> Games = new List<GameFields>() {
new GameFields(){ ID= "Game1", Text= "American Football" },
new GameFields(){ ID= "Game2", Text= "Badminton" },
new GameFields(){ ID= "Game3", Text= "Basketball" },
new GameFields(){ ID= "Game4", Text= "Cricket" },
};
private void Filteringhandler(FilteringEventArgs args)
{
// Here you can customize your code
}
}
```

OnActionBegin

OnActionBegin event triggers before fetching data from the remote server.

ASPX-CS

```
@using Syncfusion.Blazor.DropDowns
@using Syncfusion.Blazor.Data
<SfComboBox TValue="string" TItem="OrderDetails" Query="@RemoteDataQuery">
<SfDataManager
Url="https://js.syncfusion.com/demos/ejServices/Wcf/Northwind.svc/Orders"
```

```
CrossDomain="true"
Adaptor="Syncfusion.Blazor.Adaptors.ODataAdaptor"></SfDataManager>
<ComboBoxEvents TValue="string" TItem="OrderDetails"
OnActionBegin="@OnActionBeginhandler"></ComboBoxEvents>
<ComboBoxFieldSettings Text="CustomerID"
Value="CustomerID"></ComboBoxFieldSettings>
</SfComboBox>
@code {
public Query RemoteDataQuery = new Query().Select(new List<string> {
"CustomerID" }).Take(6).RequiresCount();
public class OrderDetails
{
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public int? EmployeeID { get; set; }
public double? Freight { get; set; }
public string ShipCity { get; set; }
public bool Verified { get; set; }
public DateTime? OrderDate { get; set; }
public string ShipName { get; set; }
public string ShipCountry { get; set; }
public DateTime? ShippedDate { get; set; }
public string ShipAddress { get; set; }
}
private void OnActionBeginhandler(ActionBeginEventArgs args)
{
// Here you can customize your code
}
}
```

OnActionComplete

OnActionComplete event triggers after data is fetched successfully from the remote server.

ASPX-CS

```
@using Syncfusion.Blazor.DropDowns
@using Syncfusion.Blazor.Data
<SfDropDownList TValue="string" TItem="OrderDetails"
Query="@RemoteDataQuery">
<SfDataManager
Url="https://js.syncfusion.com/demos/ejServices/Wcf/Northwind.svc/Orders"
CrossDomain="true"
Adaptor="Syncfusion.Blazor.Adaptors.ODataAdaptor"></SfDataManager>
<DropDownListEvents TValue="string" TItem="OrderDetails"
OnActionBegin="@OnActionBeginhandler"></DropDownListEvents>
<DropDownListFieldSettings Text="CustomerID"
Value="CustomerID"></DropDownListFieldSettings>
</SfDropDownList>
@code {
public Query RemoteDataQuery = new Query().Select(new List<string> {
"CustomerID" }).Take(6).RequiresCount();
public class OrderDetails
{
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public int? EmployeeID { get; set; }
}
```

```
public double? Freight { get; set; }
public string ShipCity { get; set; }
public bool Verified { get; set; }
public DateTime? OrderDate { get; set; }
public string ShipName { get; set; }
public string ShipCountry { get; set; }
public DateTime? ShippedDate { get; set; }
public string ShipAddress { get; set; }
}
private void OnActionBeginhandler(ActionBeginEventArgs args)
{
    // Here you can customize your code
}
}
```

OnActionFailure

OnActionFailure event triggers when the data fetch request from the remote server fails.

ASPX-CS

```
@using Syncfusion.Blazor.DropDowns
@using Syncfusion.Blazor.Data
<SfComboBox TValue="string" TItem="OrderDetails" Query="@RemoteDataQuery">
<SfDataManager
    Url="https://js.syncfusion.com/demos/ejServices/Wcf/Northwind.svc/Orders"
    CrossDomain="true"
    Adaptor="Syncfusion.Blazor.Adaptors.ODataAdaptor"></SfDataManager>
<ComboBoxEvents TValue="string" TItem="OrderDetails"
    OnActionFailure="@OnActionFailurehandler"></ComboBoxEvents>
<ComboBoxFieldSettings Text="CustomerID"
    Value="CustomerID"></ComboBoxFieldSettings>
</SfComboBox>
@code {
    public Query RemoteDataQuery = new Query().Select(new List<string> {
        "CustomerID" }).Take(6).RequiresCount();
    public class OrderDetails
    {
        public int? OrderID { get; set; }
        public string CustomerID { get; set; }
        public int? EmployeeID { get; set; }
        public double? Freight { get; set; }
        public string ShipCity { get; set; }
        public bool Verified { get; set; }
        public DateTime? OrderDate { get; set; }
        public string ShipName { get; set; }
        public string ShipCountry { get; set; }
        public DateTime? ShippedDate { get; set; }
        public string ShipAddress { get; set; }
    }
    private void OnActionFailurehandler(Exception args)
    {
        // Here you can customize your code
    }
}
```

OnValueSelect

OnValueSelect event triggers when a user selects an item in the popup using the mouse/tap or keyboard navigation.

ASPX-CS

```
@using Syncfusion.Blazor.DropDowns
<SfComboBox TItem="GameFields" TValue="string" DataSource="@Games">
<ComboBoxEvents TItem="GameFields" TValue="string"
OnValueSelect="@OnValueSelecthandler"></ComboBoxEvents>
<ComboBoxFieldSettings Text="Text" Value="ID"></ComboBoxFieldSettings>
</SfComboBox>
@code {
public class GameFields
{
public string ID { get; set; }
public string Text { get; set; }
}
private List<GameFields> Games = new List<GameFields>() {
new GameFields(){ ID= "Game1", Text= "American Football" },
new GameFields(){ ID= "Game2", Text= "Badminton" },
new GameFields(){ ID= "Game3", Text= "Basketball" },
new GameFields(){ ID= "Game4", Text= "Cricket" },
};
private void OnValueSelecthandler(SelectEventArgs<GameFields> args)
{
// Here you can customize your code
}
}
```

Opened

Opened event triggers when the popup opens.

ASPX-CS

```
@using Syncfusion.Blazor.DropDowns
<SfComboBox TItem="GameFields" TValue="string" DataSource="@Games">
<ComboBoxEvents TItem="GameFields" TValue="string"
Opened="@Openedhandler"></ComboBoxEvents>
<ComboBoxFieldSettings Text="Text" Value="ID"></ComboBoxFieldSettings>
</SfComboBox>
@code {
public class GameFields
{
public string ID { get; set; }
public string Text { get; set; }
}
private List<GameFields> Games = new List<GameFields>() {
new GameFields(){ ID= "Game1", Text= "American Football" },
new GameFields(){ ID= "Game2", Text= "Badminton" },
new GameFields(){ ID= "Game3", Text= "Basketball" },
new GameFields(){ ID= "Game4", Text= "Cricket" },
};
private void Openedhandler(PopupEventArgs args)
{
// Here you can customize your code
}
```

```
}
}
```

ComboBox is limited with these events and new events will be added in the future based on the user requests. If the event you are looking for is not on the list, then please request [here](#).

How To

DropDownList options with tooltip in Blazor ComboBox Component

You can achieve this behavior by using tooltip component. When the mouse is hovered over the DropDownList option, a tooltip appears with information about the hovered list item.

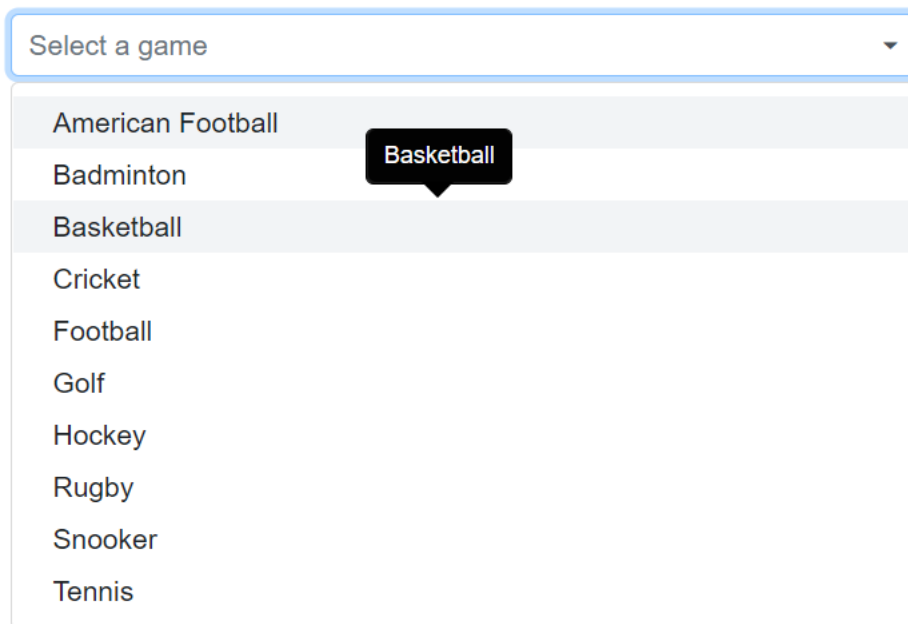
The following code demonstrates how to display a tooltip when hovering over the DropDownList option.

ASPX-CS

```
<SfTooltip @ref="TooltipObj" ID="Tooltip" Target=".e-list-item
.name[title]">
</SfTooltip>
<SfComboBox TItem="GameFields" TValue="string" Placeholder="Select a game"
DataSource="@Games">
<ComboBoxFieldSettings Text="Text" Value="ID"></ComboBoxFieldSettings>
<ComboBoxTemplates TItem="GameFields">
<ItemTemplate>
<div class="name" title="@((context as GameFields).Text)"> @((context as
GameFields).Text) </div>
</ItemTemplate>
</ComboBoxTemplates>
<ComboBoxEvents TValue="string" TItem="GameFields" Opened="OnOpen"
OnClose="OnClose"></ComboBoxEvents>
</SfComboBox>
@code {
SfTooltip TooltipObj;
public Boolean isOpen { get; set; } = false;
public class GameFields
{
public string ID { get; set; }
public string Text { get; set; }
}
private List<GameFields> Games = new List<GameFields>() {
new GameFields(){ ID= "Game1", Text= "American Football" },
new GameFields(){ ID= "Game2", Text= "Badminton" },
new GameFields(){ ID= "Game3", Text= "Basketball" },
new GameFields(){ ID= "Game4", Text= "Cricket" },
new GameFields(){ ID= "Game5", Text= "Football" },
new GameFields(){ ID= "Game6", Text= "Golf" },
new GameFields(){ ID= "Game7", Text= "Hockey" },
new GameFields(){ ID= "Game8", Text= "Rugby" },
new GameFields(){ ID= "Game9", Text= "Snooker" },
new GameFields(){ ID= "Game10", Text= "Tennis"},
};
public void OnOpen(PopupEventArgs args)
{
isOpen = true;
}
public void OnClose(PopupEventArgs args)
{
}
```

```
TooltipObj.CloseAsync();  
}  
protected override async Task OnAfterRenderAsync(bool firstRender)  
{  
    if (isOpen)  
    {  
        await TooltipObj.RefreshAsync();  
    }  
}  
}
```

The output will be as follows.



ContextMenu

Getting Started with Blazor ContextMenu Component

This section briefly explains about how to include Context Menu Component in your Blazor server-side application. You can refer [Getting Started with Syncfusion Blazor for Server-side in Visual Studio](#) page for the introduction and configuring the common specifications.

To get started quickly with Context Menu Component using Blazor, you can check out this video:

{% youtube

"youtube:https://www.youtube.com/watch?v=0-II6YezL1s"%}

Importing Syncfusion Blazor component in the application

1. Install the **Syncfusion.Blazor** NuGet package to the application by using the **NuGet Package Manager**.
2. You can add the client-side style resources through [CDN](#) or from [NuGet](#) package in the element of the `~/Pages/_Host.cshtml` page.

ASPX-CS

```
<head>
<link href="_content/Syncfusion.Blazor/styles/bootstrap4.css"
rel="stylesheet" />
@*<link href="https://cdn.syncfusion.com/blazor/{{ site.blazorversion
}}/bootstrap4.css" rel="stylesheet" />*@
</head>
```

For Internet Explorer 11, kindly refer the polyfills. Refer the [documentation](#) for more information.

ASPX-CS

```
<head>
<link href="_content/Syncfusion.Blazor/styles/bootstrap4.css"
rel="stylesheet" />
<script
src="https://github.com/Daddoon/Blazor.Polyfill/releases/download/3.0.1/blaz
or.polyfill.min.js"></script>
</head>
```

Adding component package to the application

Open `/_Imports.razor` file and import the **Syncfusion.Blazor.Navigations** package.

ASPX-CS

```
@using Syncfusion.Blazor.Navigations
```

Add SyncfusionBlazor service in Startup.cs

Open the **Startup.cs** file and add services required by Syncfusion components. Add **services.AddSyncfusionBlazor()** method in the ConfigureServices function as follows.

CSHARP

```
namespace BlazorApplication
{
    public class Startup
    {
        ....
        ....
        public void ConfigureServices(IServiceCollection services)
        {
            ....
            ....
            services.AddSyncfusionBlazor();
        }
    }
}
```

To enable custom client side resource loading from CRG or CDN. You need to disable resource loading by **AddSyncfusionBlazor(true)** and load the scripts in the HEAD element of the `~/Pages/_Host.cshtml` page.

ASPX-CS

```
<head>
```

```
<environment include="Development">
<script src="https://cdn.syncfusion.com/blazor/{ site.blazorversion
  }/syncfusion-blazor.min.js">
</script>
</environment>
</head>
```

Adding Context Menu component to the application

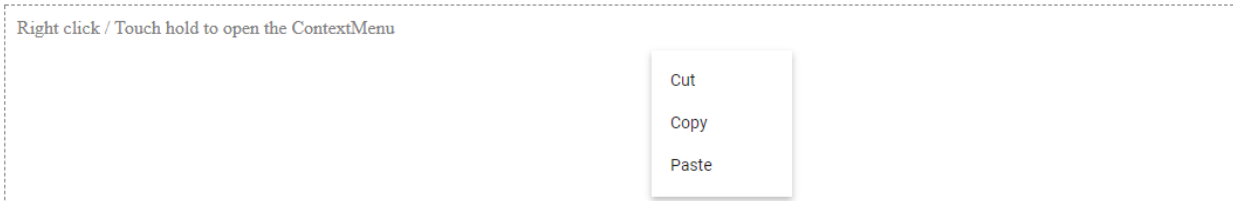
Now, add the Blazor Context Menu component in razor page in the Pages folder. For example, the Context Menu component is added in the ~/Pages/Index.razor page.

ASPX-CS

```
<div id="target">Right click/Touch hold to open the ContextMenu </div>
<SfContextMenu Target="#target" TValue="MenuItem">
  <MenuItems>
    <MenuItem Text="Cut"></MenuItem>
    <MenuItem Text="Copy"></MenuItem>
    <MenuItem Text="Paste"></MenuItem>
  </MenuItems>
</SfContextMenu>
<style>
#target {
border: 1px dashed;
height: 150px;
padding: 10px;
position: relative;
text-align: justify;
color: gray;
user-select: none;
}
</style>
```

Run the application

After successful compilation of your application, simply press F5 to run the application. The Blazor Context Menu component will render in the web browser as shown below



See Also

- [Getting Started with Syncfusion Blazor for Client-Side in .NET Core CLI](#)
- [Getting Started with Syncfusion Blazor for Server-Side in Visual Studio](#)
- [Getting Started with Syncfusion Blazor for Server-Side in .NET Core CLI](#)

Accessibility in Blazor ContextMenu Component

ARIA attributes

The web accessibility makes web content and web applications more accessible for people with disabilities. It especially helps in dynamic content change and development of advanced user interface controls with AJAX, HTML, JavaScript, and related technologies.

Context Menu provides built-in compliance with WAI-ARIA specifications. WAI-ARIA support is achieved through the attributes like `aria-expanded` and `aria-haspopup` applied for menu item in Context Menu. It helps the people with disabilities by providing information about the widget for assistive technology in the screen readers. Context Menu component contains the `menu` role and `menulitem` role.

| Properties | Functionality |

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

| menu | This role will be specified for an item which have sub menu. |

| menulitem | This role will be specified for an item that do not have sub menus. |

| aria-haspopup | Indicates the availability and type of interactive popup element. |

| aria-expanded | Indicates whether the subtree can be expanded or collapsed, as well as indicates whether its current state is expanded or collapsed. |

Keyboard interaction

<!-- markdownlint-disable MD033 -->

Keyboard shortcuts	Actions
Esc	Closes the opened ContextMenu.
Enter	Selects the focused item.
Up	Navigates up or to the previous menu item.
Down	Navigates down or to the next menu item.
Left	Closes the current sub menu and navigates to the parent menu.
Right	Navigates and opens the next sub menu.

Customizing and Multilevel nesting in Blazor ContextMenu Component

Customizing Context Menu Items

To customize Context Menu items in your application, set your customized template using [MenuTemplates](#). In the following example, the Context Menu has been rendered with customized Context Menu items.

ASPX-CS

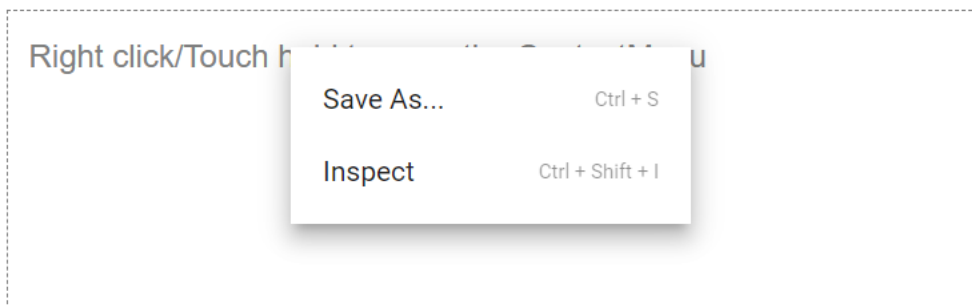
```
@using Syncfusion.Blazor.Navigations
<div id="target">Right click/Touch hold to open the ContextMenu </div>
<div class="col-lg-12 control-section">
<SfContextMenu Target="#target" TValue="MenuItem">
```

```

<MenuTemplates TValue="MenuItem">
  <Template>
    @context.Text
    <span class="shortcut">@((@context.Text == "Save As...") ? "Ctrl + S" :
    "Ctrl + Shift + I")</span>
  </Template>
</MenuTemplates>
<MenuItems>
  <MenuItem Text="Save As..."></MenuItem>
  <MenuItem Text="Inspect"></MenuItem>
</MenuItems>
</SfContextMenu>
</div>
<style>
#target {
border: 1px dashed;
height: 150px;
padding: 10px;
position: relative;
text-align: justify;
color: gray;
user-select: none;
}
.shortcut {
float: right;
font-size: 10px;
opacity: 0.5;
}
</style>
@code{
}

```

Output will be as follows



Customizing Context Menu Items using CssClass

The Context Menu items can be customized by using the `CssClass` property. In the following sample, the menu items is customized by adding new styles.

ASPX-CS

```

@using Syncfusion.Blazor.Navigations
<div id="target">Right click/Touch hold to open the ContextMenu </div>
<SfContextMenu Target="#target" TValue="MenuItem" CssClass="custom">
  <MenuItems>
    <MenuItem Text="Cut"></MenuItem>
  </MenuItems>
</SfContextMenu>

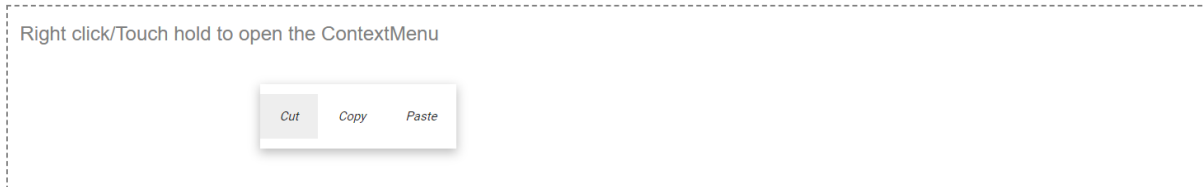
```

```

<MenuItem Text="Copy"></MenuItem>
<MenuItem Text="Paste"></MenuItem>
</MenuItems>
</SfContextMenu>
<style>
#target {
border: 1px dashed;
height: 150px;
padding: 10px;
position: relative;
text-align: justify;
color: gray;
user-select: none;
}
.custom.e-contextmenu-container .e-menu-item {
display: inline-block;
font-size: 10px;
font-style: oblique;
}
</style>

```

Output will be as follows



Multilevel nesting

The Multiple level nesting supports in Context Menu. It can be achieved by mapping the [MenuItems](#) property inside the parent [MenuItem](#). In the below sample, three level nesting of Context Menu is provided.

ASPX-CS

```

@using Syncfusion.Blazor.Navigations
<div id="target">Right click/Touch hold to open the ContextMenu </div>
<SfContextMenu Target="#target" TValue="MenuItem">
<MenuItems>
<MenuItem Text="Show All Bookmarks"></MenuItem>
<MenuItem Text="Bookmarks Toolbar">
<MenuItems>
<MenuItem Text="Most Visited">
<MenuItems>
<MenuItem Text="Google"></MenuItem>
<MenuItem Text="Gmail"></MenuItem>
</MenuItems>
</MenuItem>
</MenuItems>
</MenuItem>
<MenuItem Text="Recently Added"></MenuItem>
</MenuItems>

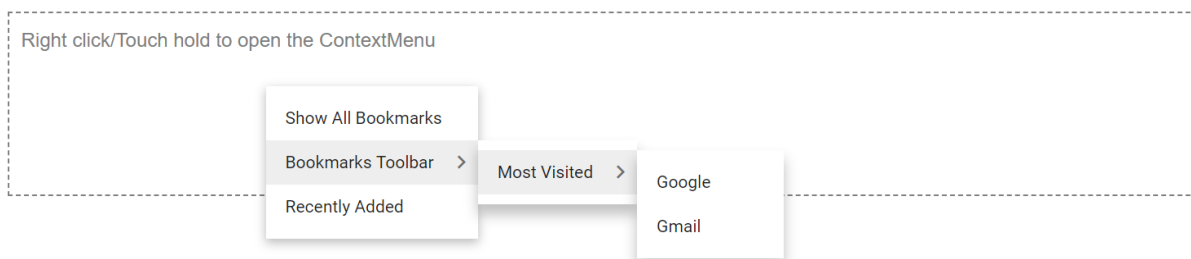
```

```

</SfContextMenu>
<style>
#target {
border: 1px dashed;
height: 150px;
padding: 10px;
position: relative;
text-align: justify;
color: gray;
user-select: none;
}
</style>

```

Output will be as follows



Icons and Navigation in Blazor ContextMenu Component

Icons

The Context Menu item has an icon/image in it to provide a visual representation of the action. To place the icon on a menu 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 menu item. In the following sample, the icons for Cut, Copy and Paste menu items are added using the [IconCss](#) property.

ASPX-CS

```

@using Syncfusion.Blazor.Navigations
<div id="target">Right click/Touch hold to open the Context Menu </div>
<SfContextMenu Target="#target" TValue="MenuItem">
<MenuItems>
<MenuItem Text="Cut" IconCss="e-icons e-cut"></MenuItem>
<MenuItem Text="Copy" IconCss="e-icons e-copy"></MenuItem>
<MenuItem Text="Paste" IconCss="e-icons e-paste"></MenuItem>
</MenuItems>
</SfContextMenu>
<style>
#target {
border: 1px dashed;
height: 150px;
padding: 10px;
position: relative;
text-align: justify;
color: gray;
user-select: none;
}
.e-cut::before {
content: '\e279';

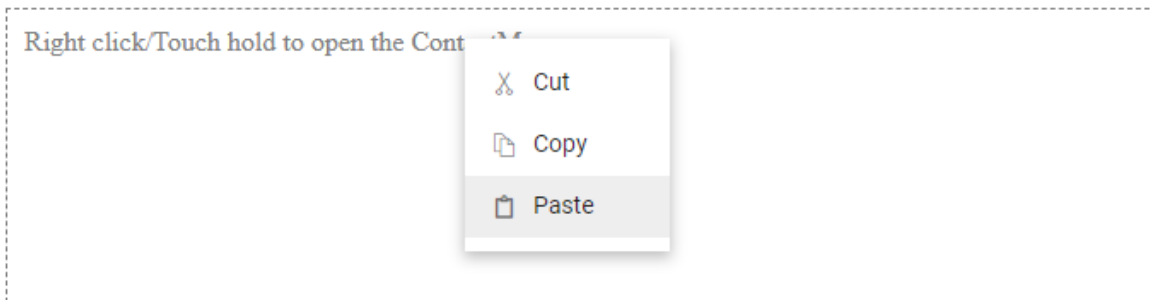
```

```

}
.e-copy::before {
content: '\e280';
}
.e-paste::before {
content: '\e601';
}
</style>

```

Output will be as follows



The Context Menu 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 Context Menu using the `IconCss` property.

Navigation

Navigation in Context Menu is used to navigate to the other web page when menu item is clicked. This can be achieved by providing link to the menu item using the `Url` property. In the following sample, Navigation URL for Flip kart, Amazon, and Snap deal menu items are added using the `Url` property.

ASPX-CS

```

@using Syncfusion.Blazor.Navigations
<div id="target">Right click/Touch hold to open the ContextMenu </div>
<SfContextMenu Target="#target" TValue="MenuItem">
  <MenuItems>
    <MenuItem Text="Flipkart"
      Url="https://www.google.co.in/search?q=flipkart"></MenuItem>
    <MenuItem Text="Amazon"
      Url="https://www.google.co.in/search?q=amazon"></MenuItem>
    <MenuItem Text="Snapdeal"
      Url="https://www.google.co.in/search?q=snapdeal"></MenuItem>
  </MenuItems>
</SfContextMenu>
<style>
#target {
border: 1px dashed;
height: 150px;
padding: 10px;
position: relative;
text-align: justify;
color: gray;

```

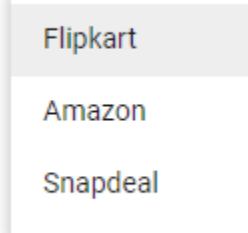
```

user-select: none;
}
</style>

```

Output will be as follows

Right click/Touch hold to open the ContextMenu



Styles and Appearances in Blazor ContextMenu Component

To modify the ContextMenu appearance, you need to override the default CSS of ContextMenu component. Please find the list of CSS classes and its corresponding section in ContextMenu 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-contextmenu-container .e-menu-parent | To customize parent context menu |

| .e-contextmenu-container ul .e-menu-item | To customize the context menu items |

| .e-contextmenu-container ul .e-menu-item.e-focused | To customize the context menu items on focus |

| .e-contextmenu-container ul .e-menu-item.e-selected .e-caret::before | To customize the selected context menu caret icon |

| .e-contextmenu-container ul .e-menu-item .e-menu-icon::before | To customize the icons of the context menu |

How To

Bind Context Menu Events in Blazor ContextMenu Component

To bind the menu event in the context menu [ItemSelected](#) is used and the event is triggered when the item in the context menu is selected.

ASPX-CS

```

@using Syncfusion.Blazor.Navigations
<div id="target">Right click/Touch hold to open the ContextMenu </div>
<SfContextMenu Target="#target" TValue="MenuItem">
  <MenuItems>
    <MenuItem Text="Cut"></MenuItem>
    <MenuItem Text="Copy"></MenuItem>
    <MenuItem Text="Paste"></MenuItem>
  </MenuItems>
  <MenuEvents TValue="MenuItem" ItemSelected="@selectedHandler"></MenuEvents>
</SfContextMenu>

```



```
@code {
public MenuItem SelectedItem;
// Triggers when the item is selected
private void selectedHandler(MenuEventArgs<MenuItem> args) {
SelectedItem = args.Item;
}
}

<style>
#target {
border: 1px dashed;
height: 150px;
padding: 10px;
position: relative;
text-align: justify;
color: gray;
user-select: none;
}
</style>
```

Output will be as follows

Right click / Touch hold to open the ContextMenu

Cut
Copy
Paste

Change animation settings in Blazor ContextMenu Component

To change the animation of the Context Menu, [MenuAnimationSettings](#) component is used to initialize the animation properties.

The supported effects for Context Menu are,

| Effect | Functionality |

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

| None | Specifies the sub menu transform with no animation effect. |

| SlideDown | Specifies the sub menu transform with slide down effect. |

| ZoomIn | Specifies the sub menu transform with zoom in effect. |

| FadeIn | Specifies the sub menu transform with fade in effect. |

The following sample illustrates how to open Context Menu with **FadeIn** effect with the **Duration** of 800ms.

ASPX-CS

```
@using Syncfusion.Blazor.Navigations
<div id="target">Right click/Touch hold to open the Context Menu</div>
<SfContextMenu Target="#target" TValue="MenuItem">
<MenuItems>
<MenuItem Text="Cut"></MenuItem>
<MenuItem Text="Copy"></MenuItem>
```

```

<MenuItem Text="Paste"></MenuItem>
</MenuItems>
<MenuAnimationSettings Effect="MenuEffect.FadeIn"
Duration="800"></MenuAnimationSettings>
</SfContextMenu>
<style>
#target {
border: 1px dashed;
height: 250px;
padding: 10px;
position: relative;
text-align: center;
color: gray;
line-height: 17;
font-size: 14px;
}
</style>

```

Output will be as follows

Right click / Touch hold to open the ContextMenu

Cut
Copy
Paste

Data Binding in Blazor ContextMenu Component

To bind local data source to the Context Menu, menu items are populated from data source and mapped to `Items` property. In the following example, custom data with different data type is mapped to `Items` property.

ASPX-CS

```

@using Syncfusion.Blazor.Navigations
<div id="target">Right click/Touch hold to open the ContextMenu </div>
<SfContextMenu Target="#target" Items="@menuItems">
<MenuFieldSettings Text="Content"></MenuFieldSettings>
</SfContextMenu>
@code {
private List<CustomItem> menuItems = new List<CustomItem>();
protected override void OnInitialized()
{
base.OnInitialized();
menuItems.Add(new CustomItem { Content = "Cut", Id = "1" });
menuItems.Add(new CustomItem { Content = "Copy", Id = "2" });
menuItems.Add(new CustomItem { Content = "Paste", Id = "3" });
menuItems.Add(new CustomItem { Content = "New", Id = "4" });
}
private class CustomItem
{
public string Content { get; set; }
public string Id { get; set; }
}
}

```

```
<style>
#target {
border: 1px dashed;
height: 150px;
padding: 10px;
position: relative;
text-align: justify;
color: gray;
user-select: none;
}
</style>
```

Output will be as follows



Enable/Disable Context Menu items in Blazor ContextMenu Component

You can enable and disable the menu items using the [Disabled](#) property in [MenuItem](#). To disable menuitems, set the [Disabled](#) property in each item to `true` and vice-versa.

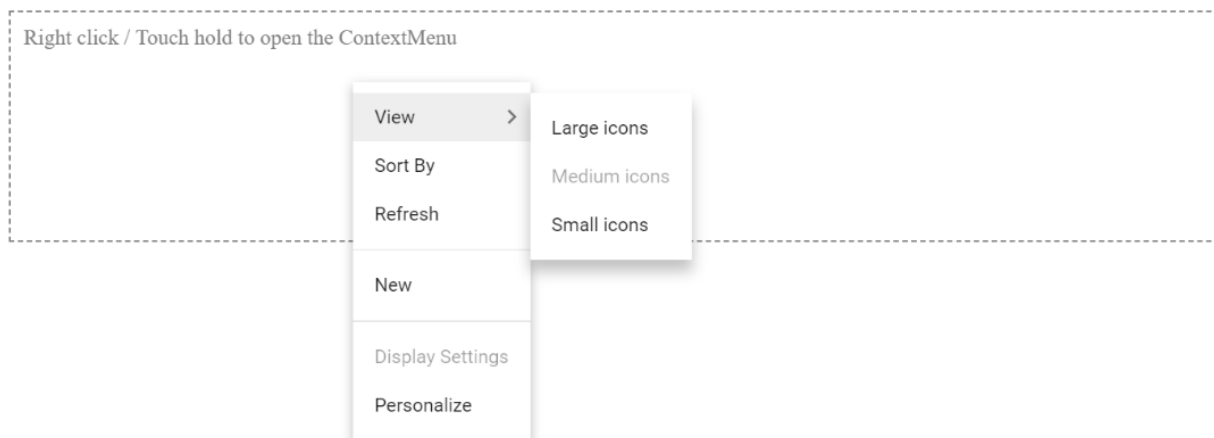
In the following example, the **Display Settings** in parent items is disabled during initial loading and **Medium icons** in sub menu items are enabled/disabled dynamically while opening the sub menu.

ASPX-CS

```
@using Syncfusion.Blazor.Navigations
<div id="target">Right click/Touch hold to open the ContextMenu </div>
<SfContextMenu Target="#target" TValue="MenuItem">
  <MenuItems>
    <MenuItem Text="View">
      <MenuItems>
        <MenuItem Text="Large Icons"></MenuItem>
        <MenuItem Text="Medium Icons" Disabled="@disableState"></MenuItem>
        <MenuItem Text="Small Icons"></MenuItem>
      </MenuItems>
    </MenuItem>
    <MenuItem Text="Sort By"></MenuItem>
    <MenuItem Text="Refresh"></MenuItem>
    <MenuItem Separator="true"></MenuItem>
    <MenuItem Text="New"></MenuItem>
    <MenuItem Separator="true"></MenuItem>
    <MenuItem Text="Display Settings" Disabled="true"></MenuItem>
    <MenuItem Text="Personalize"></MenuItem>
  </MenuItems>
  <MenuEvents TValue="MenuItem" OnOpen="@BeforeOpenHandler"></MenuEvents>
</SfContextMenu>
@code {
private bool disableState;
```

```
private void BeforeOpenHandler(BeforeOpenCloseMenuEventArgs<MenuItem> e)
{
    // While opening the first level context menu the parent item will not be
    // available, so it would be null.
    if (e.ParentItem != null && e.ParentItem.Text == "View")
        disableState = !disableState; // Execute only for the View item sub menu.
    };
<style>
#target {
border: 1px dashed;
height: 150px;
padding: 10px;
position: relative;
text-align: justify;
color: gray;
user-select: none;
}
</style>
```

Output will be as follows



To disable sub menu items, use the `OnOpen` event.

Open a dialog on Item Click in Blazor Context Menu

This section explains about how to open a dialog on Context Menu item click. This can be achieved by handling dialog open in [ItemSelected](#) event of the Context Menu.

In the following sample, Dialog will open while clicking `Save As...` item.

ASPX-CS

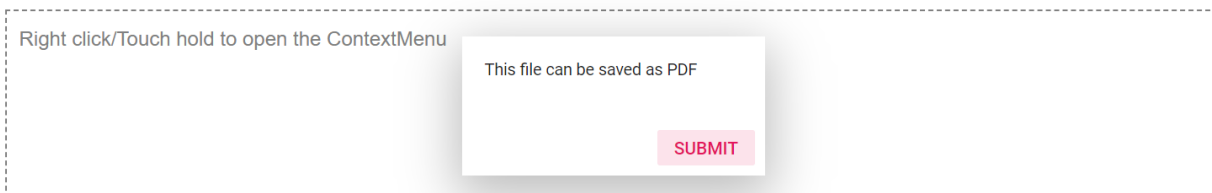
```
@using Syncfusion.Blazor.Navigations
@using Syncfusion.Blazor.Popups
<div id="target">Right click/Touch hold to open the ContextMenu </div>
<SfContextMenu Target="#target" TValue="MenuItem">
<MenuItems>
<MenuItem Text="Back"></MenuItem>
<MenuItem Text="Forward"></MenuItem>
<MenuItem Text="Reload"></MenuItem>
```

```

<MenuItem Separator="true"></MenuItem>
<MenuItem Text="Save As..."></MenuItem>
<MenuItem Text="Print"></MenuItem>
<MenuItem Text="Cast"></MenuItem>
</MenuItems>
<MenuEvents TValue="MenuItem" ItemSelected="@SelectedHandler"></MenuEvents>
</SfContextMenu>
<SfDialog @ref="dialogObj" Content="@content" Visible="false"
Target="#target" Width="200px" Height="110px">
<DialogButtons>
<DialogButton Content="Submit" IsPrimary="true"
OnClick="@Clicked"></DialogButton>
</DialogButtons>
</SfDialog>
@code {
private SfDialog dialogObj;
private string content = "This file can be saved as PDF";
private async Task SelectedHandler(MenuEventArgs<MenuItem> e)
{
if (e.Item.Text == "Save As...")
await dialogObj.Show();
}
private async Task Clicked(object args)
{
await dialogObj.Hide();
}
}
<style>
#target {
border: 1px dashed;
height: 150px;
padding: 10px;
position: relative;
text-align: justify;
color: gray;
user-select: none;
}
</style>

```

Output will be as follows



Open Sub Menu on Item Click in Blazor ContextMenu Component

This section explains about how to open a sub menu on Context Menu item click. This can be achieved by using `ShowItemOnClick` property of the Context Menu.

In the following sample, Sub Menu will open while clicking `Save` item.

ASPX-CS

```
@using Syncfusion.Blazor.Navigations
<div id="target">Right click/Touch hold to open the ContextMenu </div>
<SfContextMenu Target="#target" TValue="MenuItem" ShowItemOnClick="true">
  <MenuItems>
    <MenuItem Text="Back"></MenuItem>
    <MenuItem Text="Forward"></MenuItem>
    <MenuItem Text="Reload"></MenuItem>
    <MenuItem Separator="true"></MenuItem>
    <MenuItem Text="Save">
      <MenuItems>
        <MenuItem Text="Save"></MenuItem>
        <MenuItem Text="Save As..."></MenuItem>
      </MenuItems>
    </MenuItem>
    <MenuItem Text="Print"></MenuItem>
    <MenuItem Text="Cast"></MenuItem>
  </MenuItems>
</SfContextMenu>
<style>
#target {
border: 1px dashed;
height: 150px;
padding: 10px;
position: relative;
text-align: justify;
color: gray;
user-select: none;
}
</style>
```

Open and close Context Menu in Blazor ContextMenu Component

Open and close the Context Menu manually whenever required by using the `Open` and `Close` methods. In the following sample, the Context Menu manually opens while clicking the button using `Open` method with `ClientX` and `ClientY` coordinates.

To manually close the Context Menu, `Close` method can be used.

ASPX-CS

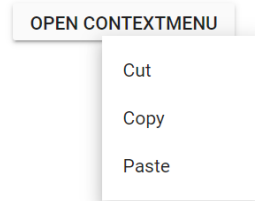
```
@using Syncfusion.Blazor.Navigations
@using Syncfusion.Blazor.Buttons
<div id="target">
  <SfContextMenu @ref="contextMenuObj" TValue="MenuItem">
    <MenuItems>
      <MenuItem Text="Cut"></MenuItem>
      <MenuItem Text="Copy"></MenuItem>
      <MenuItem Text="Paste"></MenuItem>
    </MenuItems>
  </SfContextMenu>
</div>
```

```

</SfContextMenu>
<SfButton @onclick="OpenContextMenu">Open ContextMenu</SfButton>
</div>
@code {
SfContextMenu<MenuItem> contextMenuObj;
private void OpenContextMenu(MouseEventArgs e)
{
contextMenuObj.Open(e.ClientX, e.ClientY);
}
}

```

Output will be as follows



Dashboard Layout

Getting Started with Blazor Dashboard Layout Component

This section briefly explains about how to include a **Dashboard Layout** component in your **Blazor** server-side application. You can refer [Getting Started with Syncfusion Blazor for Server-Side in Visual Studio](#) page for introduction and configuring the common specifications.

Importing Syncfusion Blazor component in the application

Using Syncfusion.Blazor NuGet Package [Old standard]

- Install **Syncfusion.Blazor** NuGet package to the application by using the **NuGet Package Manager**.
- You can add the client-side style resources using **NuGet** package to the element of the `~/wwwroot/index.html` page in Blazor WebAssembly app or `~/Pages/_Host.cshtml` page in Blazor Server app.

HTML

```

<head>
....
....
<link href="_content/Syncfusion.Blazor/styles/bootstrap4.css"
rel="stylesheet" />
</head>

```

HTML

```

<head>
<link href="https://cdn.syncfusion.com/blazor/{{ site.blazorversion
}}/styles/bootstrap4.css" rel="stylesheet" />

```

```
</head>
```

- For Internet Explorer 11, kindly refer the polyfills. Refer the [documentation](#) for more information.

HTML

```
<head>
<link href="_content/Syncfusion.Blazor/styles/bootstrap4.css"
rel="stylesheet" />
<script
src="https://github.com/Daddoon/Blazor.Polyfill/releases/download/3.0.1/blaz
or.polyfill.min.js"></script>
</head>
```

Using Syncfusion.Blazor NuGet Package [New standard]

- Install **Syncfusion.Blazor.Layouts** NuGet package to the application by using the **NuGet Package Manager**.

Warning: Syncfusion.Blazor package should not be installed along with [individual NuGet packages](#). Hence, you have to add the below **Syncfusion.Blazor.Themes** static web assets (styles) in the application.

- You can add the client-side style resources through [CDN](#) or from [NuGet](#) package to the element of the `~/wwwroot/index.html` page in Blazor WebAssembly app or `~/Pages/_Host.cshtml` page in Blazor Server app.

HTML

```
<head>
....
....
<link href="_content/Syncfusion.Blazor.Themes/bootstrap4.css"
rel="stylesheet" />
</head>
```

Warning: If you prefer the above new standard (individual NuGet packages), then skip this section. Using both old and new standards in the same application will throw ambiguous compilation errors.

Add Syncfusion Blazor service in Startup.cs (Server-side application)

Open the **Startup.cs** file and add services required by Syncfusion components using `services.AddSyncfusionBlazor()` method. Add this method in the **ConfigureServices** function as follows.

C#

```
using Syncfusion.Blazor;
namespace BlazorApplication
{
```



```
public class Startup
{
    ....
    ....
    public void ConfigureServices(IServiceCollection services)
    {
        ....
        ....
        services.AddSyncfusionBlazor();
    }
}
```

Add Syncfusion Blazor service in Program.cs (Client-side application)

Open the **Program.cs** file and add services required by Syncfusion components using `builder.services.AddSyncfusionBlazor()` method. Add this method in the **Main** function as follows.

CSHARP

```
using Syncfusion.Blazor;
namespace BlazorApplication
{
    public class Program
    {
        ....
        ....
        public static async Task Main(string[] args)
        {
            ....
            ....
            builder.Services.AddSyncfusionBlazor();
        }
    }
}
```

To enable custom client side resource loading from CRG or CDN. You need to disable resource loading by **AddSyncfusionBlazor(true)** and load the scripts to the `<head>` element of the `~/wwwroot/index.html` page in Blazor WebAssembly app or `~/Pages/_Host.cshtml` page in Blazor Server app.

HTML

```
<head>
<script src="https://cdn.syncfusion.com/blazor/{{ site.blazorversion
}}/syncfusion-blazor.min.js"></script>
</head>
```

Adding Dashboard Layout component namespace to the application

Open `~/_Imports.razor` file and import the `Syncfusion.Blazor.Layouts` package.

ASPX-CS

```
@using Syncfusion.Blazor.Layouts
```

Initialize the Dashboard Layout component

It is easy to initialize a dashboard layout component with panel. To render a dashboard layout component with default **Row** and **Column** (Row=0, Column=0) value of panels, refer to the following code section.

[\[Pages/Index.razor\]](#)

ASPX-CS

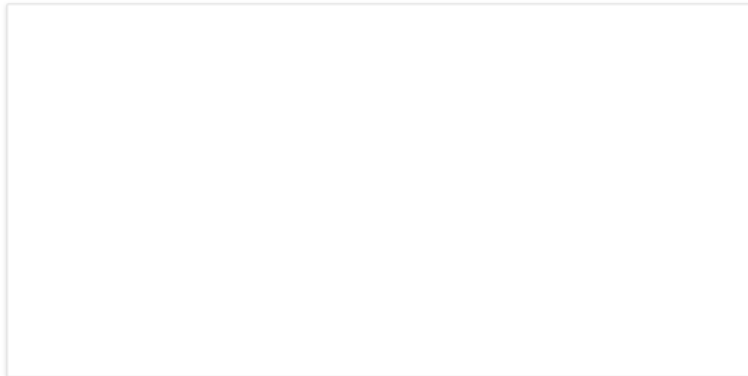
```
@using Syncfusion.Blazor.Layouts
<SfDashboardLayout>
<DashboardLayoutPanels>
<DashboardLayoutPanel>
</DashboardLayoutPanel>
</DashboardLayoutPanels>
</SfDashboardLayout>
```

There is no need to assign default value for panels. Refer to the [Panels](#) section to learn about default value.

Run the application

After successful compilation of your application, simply press **F5** to run the application.

The simple Dashboard panel is rendered with whole dimension of parent element.



Defining panels

A dashboard layout can be rendered with multiple panels to design a template with its basic properties.

A dashboard layout panel consists of two sections, which are Header and Content section.

HeaderTemplate is used to render the Header section and **ContentTemplate** is used to render the content section.

Also, it is easy to interact with the panels by dragging, floating, and resizing functionality of panels.

[Panels with simple data](#)

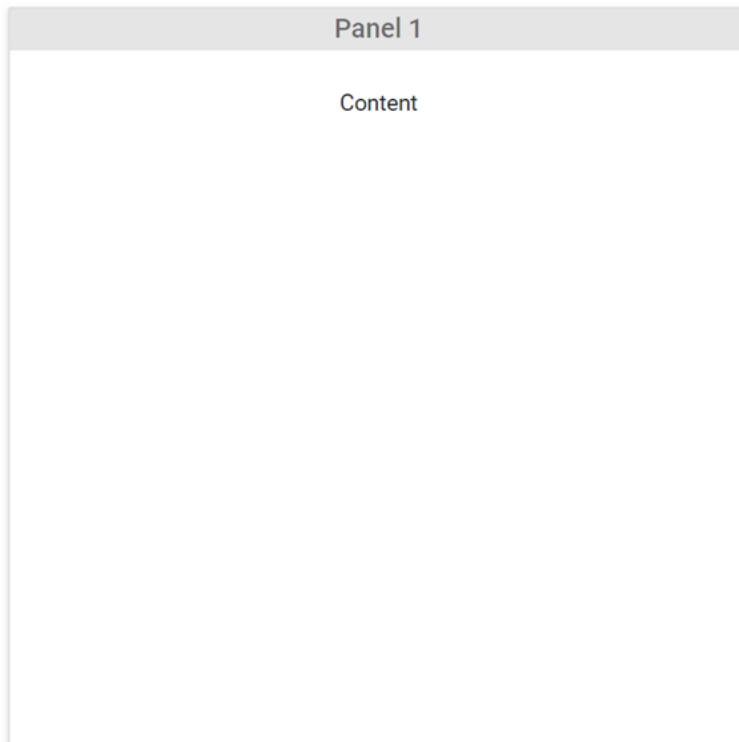
A dashboard layout panel is rendered with simple data. The header of a panel is defined by **HeaderTemplate** and content of a panel is defined by **ContentTemplate**.

ASPX-CS

```
@using Syncfusion.Blazor.Layouts
<SfDashboardLayout>
<DashboardLayoutPanels>
```

```
<DashboardLayoutPanel>
<HeaderTemplate><div>Panel 1</div></HeaderTemplate>
<ContentTemplate><div>Content</div></ContentTemplate>
</DashboardLayoutPanel>
</DashboardLayoutPanels>
</SfDashboardLayout>
<style>
.e-panel-header {
background-color: rgba(0, 0, 0, .1);
text-align: center;
}
.e-panel-content {
text-align: center;
margin-top: 10px;
}
</style>
```

The Dashboard layout with simple content will be rendered in the web browser as demonstrated in the following screenshot.



Panels with components

A dashboard layout can be rendered with the components like the chart, grids, maps, gauge, and more as a content of dashboard layout panel.

These complex data (components) are placed as the panel content by assigning the corresponding component element as the **ContentTemplate** of the panel.

ASPX-CS

```
@using Syncfusion.Blazor.Layouts
```

```

@using Syncfusion.Blazor.Charts
@using Syncfusion.Blazor.Grids
<div class="content">
<SfDashboardLayout Columns="6" CellSpacing="@ (new double[] {10 ,10 })">
<DashboardLayoutPanels>
<DashboardLayoutPanel Id="Panel1" SizeX="4" SizeY="2">
<HeaderTemplate><div class='header'> Customers Count </div></HeaderTemplate>
<ContentTemplate>
<div style="height:100%; width:100%;">
<SfChart ID="linechart" @ref="linechartObj">
<ChartPrimaryXAxis ValueType="Syncfusion.Blazor.Charts.ValueType.DateTime">
</ChartPrimaryXAxis>
<ChartSeriesCollection>
<ChartSeries DataSource="@DataSource" XName="XValue" YName="YValue"
Type="ChartSeriesType.Line">
<ChartMarker Visible="true">
<ChartDataLabel Visible="true"
Position="Syncfusion.Blazor.Charts.LabelPosition.Top">
</ChartDataLabel>
</ChartMarker>
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
</div>
</ContentTemplate>
</DashboardLayoutPanel>
<DashboardLayoutPanel Id="Panel2" SizeX="2" SizeY="2" Column="4">
<HeaderTemplate><div class='header'> Product sales in Years
</div></HeaderTemplate>
<ContentTemplate>
<div style="height:100%; width:100%;">
<SfRangeNavigator ID="range" @ref="rangeObj" Value="@Value"
ValueType="Syncfusion.Blazor.Charts.RangeValueType.DateTime"
IntervalType="RangeIntervalType.Years" LabelFormat="yyyy">
<RangeNavigatorSeriesCollection>
<RangeNavigatorSeries DataSource="@DataSource" XName="XValue"
Type="RangeNavigatorType.Area" YName="YValue"></RangeNavigatorSeries>
</RangeNavigatorSeriesCollection>
</SfRangeNavigator>
</div>
</ContentTemplate>
</DashboardLayoutPanel>
<DashboardLayoutPanel Id="Panel3" SizeX="3" SizeY="2" Row=2 Column=3>
<HeaderTemplate><div class='header'> Sales Ratio in Countries
</div></HeaderTemplate>
<ContentTemplate>
<div style="height:100%; width:100%;">
<SfChart ID="chart" @ref="barchartObj">
<ChartPrimaryXAxis Title="Country" EnableTrim="true"
ValueType="Syncfusion.Blazor.Charts.ValueType.Category">
<ChartAxisMajorGridLines Width="0"></ChartAxisMajorGridLines>
</ChartPrimaryXAxis>
<ChartSeriesCollection>
<ChartSeries DataSource="@DataSource" XName="X" YName="Y"
Type="ChartSeriesType.Bar">
</ChartSeries>
</ChartSeriesCollection>

```

```

</SfChart>
</div>
</ContentTemplate>
</DashboardLayoutPanel>
<DashboardLayoutPanel Id="Panel4" SizeX=3 SizeY=2 Row=2 Column=0>
<HeaderTemplate><div class='header'> Sales Comparison in Products
</div></HeaderTemplate>
<ContentTemplate>
<div style="height:100%; width:100%;">
<SfChart ID="chart1" @ref="chartObj" Width="100%" Height="100%">
<ChartPrimaryXAxis
ValueType="Syncfusion.Blazor.Charts.ValueType.Category"></ChartPrimaryXAxis>
<ChartSeriesCollection>
<ChartSeries DataSource="@DataSource" Name="Editors" XName="X1"
YName="YValue" Type="ChartSeriesType.StackingColumn">
</ChartSeries>
<ChartSeries DataSource="@DataSource" Name="Layouts" XName="X1"
YName="YValue" Type="ChartSeriesType.StackingColumn">
</ChartSeries>
<ChartSeries DataSource="@DataSource" Name="Grids" XName="X1" YName="YValue"
Type="ChartSeriesType.StackingColumn">
</ChartSeries>
</ChartSeriesCollection>
</SfChart>
</div>
</ContentTemplate>
</DashboardLayoutPanel>
<DashboardLayoutPanel Id="Panel5" SizeX=6 SizeY=2 Column=6 Row=4>
<HeaderTemplate><div class='header'> Top Customers
Details</div></HeaderTemplate>
<ContentTemplate>
<div style="height:100%; width:100%;">
<SfGrid ID="grid" DataSource="@Orders">
<GridPageSettings></GridPageSettings>
<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
TextAlign="Syncfusion.Blazor.Grids.TextAlign.Right"
Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="yMd" Type="ColumnType.Date"
TextAlign="Syncfusion.Blazor.Grids.TextAlign.Right"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="Syncfusion.Blazor.Grids.TextAlign.Right"
Width="120"></GridColumn>
</GridColumns>
</SfGrid>
</div>
</ContentTemplate>
</DashboardLayoutPanel>
</DashboardLayoutPanels>
</SfDashboardLayout>
</div>
@code
{ SfChart chartObj;

```

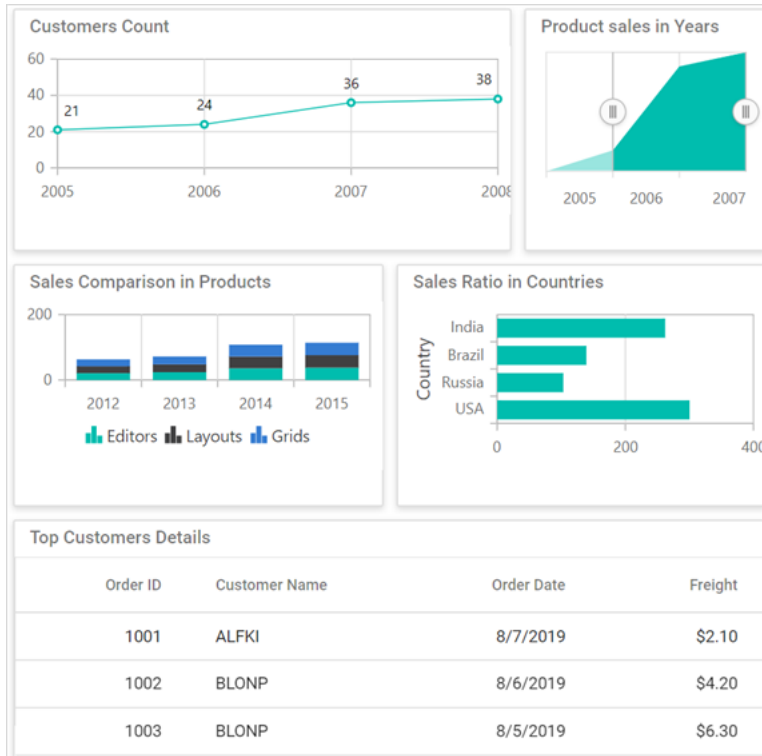
```

SfChart barchartObj;
SfRangeNavigator rangeObj;
SfChart linechartObj;
private object[] Value = new object[] { new DateTime(2006, 01, 01), new
DateTime(2008, 01, 01) };
public class ChartData
{
    public DateTime XValue;
    public double YValue;
    public string X;
    public double Y;
    public string Country;
    public string X1;
    public double Y1;
    public double Y2;
    public double Y3;
    public double Y4;
}
public List<ChartData> DataSource = new List<ChartData>
{
    new ChartData { XValue = new DateTime(2005, 01, 01), YValue = 21, X = "USA",
Y =300.2, Country = "USA: 72", X1= "2012"},
    new ChartData { XValue = new DateTime(2006, 01, 01), YValue = 24, X =
"Russia", Y = 103.1, Country = "RUS: 103.1", X1= "2013"},
    new ChartData { XValue = new DateTime(2007, 01, 01), YValue = 36, X =
"Brazil", Y = 139.1, Country = "BRZ: 139.1", X1= "2014"},
    new ChartData { XValue = new DateTime(2008, 01, 01), YValue = 38, X =
"India", Y = 262.1, Country = "IND: 262.1", X1= "2015"},
};
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
    Orders = Enumerable.Range(1, 6).Select(x => new Order()
    {
        OrderID = 1000 + x,
        CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
    })[new Random().Next(5)],
        Freight = 2.1 * x,
        OrderDate = DateTime.Now.AddDays(-x),
    }).ToList();
}
public class Order
{
    public int? OrderID { get; set; }
    public string CustomerID { get; set; }
    public DateTime? OrderDate { get; set; }
    public double? Freight { get; set; }
}
protected override async Task OnAfterRenderAsync(bool firstRender)
{
    await Task.Delay(3000); // simulate the async operations
    this.chartObj.Refresh();
    this.linechartObj.Refresh();
    this.barchartObj.Refresh();
}
}
<style>

```

```
#linechart, #grid, #chart1, #chart, #range {
height: 100% !important;
width: 100% !important;
}
</style>
```

Output for the Dashboard Layout Complex data (Blazor Components) will be as follows.



By default, the dashboard layout control is rendered with auto adjustable and [responsive](#) according to the parent dimensions.

See Also

- [Getting Started with Syncfusion Blazor for Client-Side in .NET Core CLI](#)
- [Getting Started with Syncfusion Blazor for Client-Side in Visual Studio 2019](#)
- [Getting Started with Syncfusion Blazor for Server-Side in .NET Core CLI](#)

Configuring the Grid Layout in Blazor Dashboard Layout Component

The **Dashboard Layout** is a grid structured component, which can be split into subsections of equal size known as cells.

| **Properties** | **Description** |

| --- | --- |

| **Columns** | Specifies the total number of cells in each row. |

| **CellAspectRatio** | Specifies the height of cells to any desired size. |

Modifying cell size

The size of grid cells can be modified to the required size using the `Columns` and `CellAspectRatio` properties.

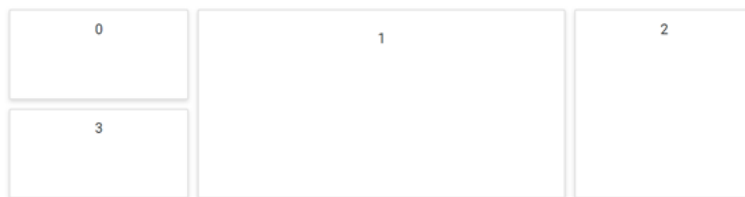
ASPX-CS

```
@using Syncfusion.Blazor.Layouts
<SfDashboardLayout CellSpacing="@ (new double[] {10 ,10 }) "
CellAspectRatio="2" Columns="5">
<DashboardLayoutPanels>
<DashboardLayoutPanel>
<ContentTemplate><div>0</div></ContentTemplate>
</DashboardLayoutPanel>
<DashboardLayoutPanel SizeX=2 SizeY=2 Column=1>
<ContentTemplate><div>1</div></ContentTemplate>
</DashboardLayoutPanel>
<DashboardLayoutPanel SizeY=2 Column=3>
<ContentTemplate><div>2</div></ContentTemplate>
</DashboardLayoutPanel>
<DashboardLayoutPanel Row=1>
<ContentTemplate><div>3</div></ContentTemplate>
</DashboardLayoutPanel>
</DashboardLayoutPanels>
</SfDashboardLayout>
<style>
.e-panel-content {
text-align: center;
margin-top: 10px;
}
</style>
```

In the above sample, width of the parent element is divided into five equal cells based on the `Columns` property value resulting the width of each cell as 100px.

The height of these cells will be 50px based on the `CellAspectRatio` value 100/50 (that is for every 100px of width, 50px will be the height of the cell).

The following output demonstrates the setting of `cellAspectRatio` and `Columns` properties in the dashboard component.



Setting cell spacing

The spacing between each panel in a row and column can be defined using the `CellSpacing` property. Adding space between the panels will make the layout effective and provides a clear data representation.

ASPX-CS

```
@using Syncfusion.Blazor.Layouts
```

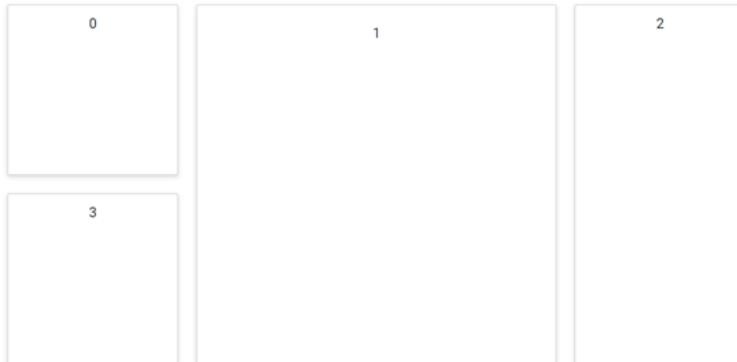


```

<SfDashboardLayout CellSpacing="@ (new double[] {20 ,20 })" Columns="5">
  <DashboardLayoutPanels>
    <DashboardLayoutPanel>
      <ContentTemplate><div>0</div></ContentTemplate>
    </DashboardLayoutPanel>
    <DashboardLayoutPanel SizeX=2 SizeY=2 Column=1>
      <ContentTemplate><div>1</div></ContentTemplate>
    </DashboardLayoutPanel>
    <DashboardLayoutPanel SizeY=2 Column=3>
      <ContentTemplate><div>2</div></ContentTemplate>
    </DashboardLayoutPanel>
    <DashboardLayoutPanel Row=1>
      <ContentTemplate><div>3</div></ContentTemplate>
    </DashboardLayoutPanel>
  </DashboardLayoutPanels>
</SfDashboardLayout>
<style>
.e-panel-content {
text-align: center;
margin-top: 10px;
}
</style>

```

The following output demonstrates the neat and clear representation of data by setting the `cellSpacing` property in dashboard component.



Graphical representation of grid layout

These cells combinedly forms a grid-structured layout, which will be hidden initially. This grid-structured layout can be made visible by enabling the `ShowGridLines` property, which clearly shows the cells split-up within the layout. These gridlines are helpful in panels sizing and placement within the layout during initial designing of a dashboard.

ASPX-CS

```

@using Syncfusion.Blazor.Layouts
<SfDashboardLayout CellSpacing="@ (new double[] {10 ,10 })" Columns="5"
ShowGridLines="true">
  <DashboardLayoutPanels>
    <DashboardLayoutPanel Col=1>
      <ContentTemplate><div>0</div></ContentTemplate>
    </DashboardLayoutPanel>
    <DashboardLayoutPanel SizeX=1 SizeY=2 Column=2>

```

```
<ContentTemplate><div>1</div></ContentTemplate>
</DashboardLayoutPanel>
<DashboardLayoutPanel SizeY=1 Column=3>
<ContentTemplate><div>2</div></ContentTemplate>
</DashboardLayoutPanel>
<DashboardLayoutPanel Row=1 Column=3>
<ContentTemplate><div>3</div></ContentTemplate>
</DashboardLayoutPanel>
</DashboardLayoutPanels>
</SfDashboardLayout>
<style>
.e-panel-content {
text-align: center;
margin-top: 10px;
}
</style>
```

The following output demonstrates the gridlines indicating the cells split-up of the layout and the data containers placed over these cells are known as panels.



Panels

Size and Position in Blazor Dashboard Layout Component

Panels are the basic building blocks of the dashboard layout component. They act as a container for the data to be visualized or presented.

The following table represents all the available panel properties and the corresponding functionalities:

PanelObject	Default Value	Description
---	---	---
Id	null	Specifies the ID value of the panel.
Row	0	Specifies the row value in which the panel to be placed.
Col	0	Specifies the column value in which the panel to be placed.
SizeX	1	Specifies the width of the panel in cells count.
SizeY	1	Specifies the height of the panel in cells count.
MinSizeX	1	Specifies the minimum width of the panel in cells count.
MinSizeY	1	Specifies the minimum height of the panel in cells count.
MaxSizeX	null	Specifies the maximum width of the panel in cells count.

| MaxSizeY | null | Specifies the maximum height of the panel in cells count. |

| HeaderTemplate | null | Specifies the header template of the panel. |

| ContentTemplate | null | Specifies the content template of the panel. |

| CssClass | null | Specifies the CSS class name that can be appended with each panel element. |

Positioning of panels

The panels within the layout can be easily positioned or ordered using the **Row** and **Col** properties of the panels. Positioning of panels will be beneficial to represent the data in any desired order.

ASPX-CS

```
@using Syncfusion.Blazor.Layouts
<SfDashboardLayout CellSpacing=@"(new double[] {20 ,20 })" Columns="4">
<DashboardLayoutPanels>
<DashboardLayoutPanel>
<ContentTemplate><div>1</div></ContentTemplate>
</DashboardLayoutPanel>
<DashboardLayoutPanel Column=1>
<ContentTemplate><div>2</div></ContentTemplate>
</DashboardLayoutPanel>
<DashboardLayoutPanel Column=2>
<ContentTemplate><div>3</div></ContentTemplate>
</DashboardLayoutPanel>
<DashboardLayoutPanel Row=1>
<ContentTemplate><div>4</div></ContentTemplate>
</DashboardLayoutPanel>
<DashboardLayoutPanel Row=1 Column=1>
<ContentTemplate><div>5</div></ContentTemplate>
</DashboardLayoutPanel>
<DashboardLayoutPanel Row=1 Column=2>
<ContentTemplate><div>6</div></ContentTemplate>
</DashboardLayoutPanel>
</DashboardLayoutPanels>
</SfDashboardLayout>
<style>
.e-panel-content {
text-align: center;
margin-top: 10px;
}
</style>
```

The following screenshot shows the positioning of panels within the dashboard layout using the row and column properties of the panels.



Sizing of panels

A panel's size can be varied easily by defining the `SizeX` and `SizeY` properties.

- `SizeX` property defines the width of a panel in cells count.
- `SizeY` property defines the height of a panel in cells count.

These properties are helpful in designing a dashboard, where the content of each panel may vary in size.

ASPX-CS

```
@using Syncfusion.Blazor.Layouts
<SfDashboardLayout CellSpacing="@ (new double[] {10 ,10 })" Columns="5">
  <DashboardLayoutPanels>
    <DashboardLayoutPanel>
      <ContentTemplate><div>0</div></ContentTemplate>
    </DashboardLayoutPanel>
    <DashboardLayoutPanel SizeX=2 Column=1>
      <ContentTemplate><div>1</div></ContentTemplate>
    </DashboardLayoutPanel>
    <DashboardLayoutPanel SizeY=2 Column=3>
      <ContentTemplate><div>2</div></ContentTemplate>
    </DashboardLayoutPanel>
    <DashboardLayoutPanel Row=1>
      <ContentTemplate><div>3</div></ContentTemplate>
    </DashboardLayoutPanel>
    <DashboardLayoutPanel Row=1 Column=1>
      <ContentTemplate><div>4</div></ContentTemplate>
    </DashboardLayoutPanel>
    <DashboardLayoutPanel Row=1 Column=2>
      <ContentTemplate><div>5</div></ContentTemplate>
    </DashboardLayoutPanel>
  </DashboardLayoutPanels>
</SfDashboardLayout>
<style>
.e-panel-content {
text-align: center;
margin-top: 10px;
}
```

```
</style>
```

The following screenshot shows the sizing of panels within the dashboard layout using the SizeX and SizeY properties of the panels.



Header and Content in Blazor Dashboard Layout Component

Basically, Dashboard layout Component have two templates to render the data in panels.

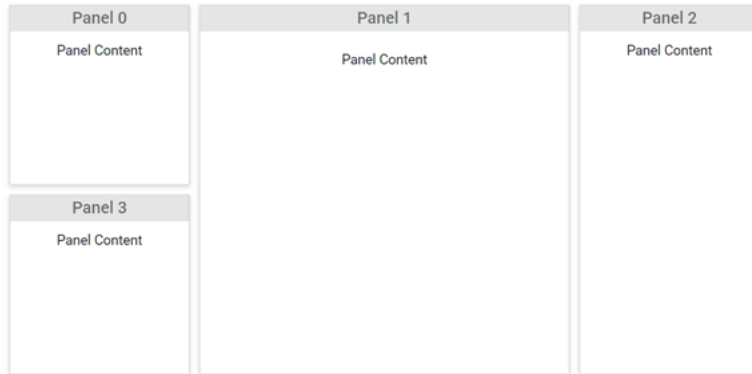
- ContentTemplate: To render data or any HTML template as the content.
- HeaderTemplate: A word or phrase that summarize the panel's content can be added as the header on the top of each panel.

ASPX-CS

```
@using Syncfusion.Blazor.Layouts
<SfDashboardLayout CellSpacing="@ (new double[] {10 ,10 })" Columns="5">
<DashboardLayoutPanels>
<DashboardLayoutPanel>
<HeaderTemplate><div>Panel 0</div></HeaderTemplate>
<ContentTemplate><div>Panel Content</div></ContentTemplate>
</DashboardLayoutPanel>
<DashboardLayoutPanel SizeX=2 SizeY=2 Column=1>
<HeaderTemplate><div>Panel 1</div></HeaderTemplate>
<ContentTemplate><div>Panel Content</div></ContentTemplate>
</DashboardLayoutPanel>
<DashboardLayoutPanel SizeY=2 Column=3>
<HeaderTemplate><div>Panel 2</div></HeaderTemplate>
<ContentTemplate><div>Panel Content</div></ContentTemplate>
</DashboardLayoutPanel>
<DashboardLayoutPanel Row=1>
<HeaderTemplate><div>Panel 3</div></HeaderTemplate>
<ContentTemplate><div>Panel Content</div></ContentTemplate>
</DashboardLayoutPanel>
</DashboardLayoutPanels>
</SfDashboardLayout>
<style>
.e-panel-header {
background-color: rgba(0, 0, 0, .1);
text-align: center;
}
.e-panel-content {
```

```
text-align: center;
margin-top: 10px;
}
</style>
```

The following output demonstrates the Header and content of Panels using templates.



Interaction With Panels

Drag and Drop in Blazor Dashboard Layout Component

The Dashboard Layout component is provided with dragging functionality to drag and reorder the panels within the layout. While dragging a panel, a holder will be highlighted below the panel indicating the panel placement on panel drop. This helps the users to decide whether to place the panel in the current position or revert to previous position without disturbing the layout.

If one or more panels collide while dragging, then the colliding panels will be pushed towards left, right, top, or bottom direction where an adaptive space for the collided panel is available. The position changes of these collided panels will be updated dynamically during dragging of a panel, so the users can conclude whether to place the panel in the current position or not.

The complete panel will act as the handler for dragging the panel such that the dragging action occurs on clicking anywhere over a panel.

ASPX-CS

```
@using Syncfusion.Blazor.Layouts
<SfDashboardLayout CellSpacing="@ (new double[] {10 ,10 }) "
CellAspectRatio="2" Columns="5">
<DashboardLayoutPanels>
<DashboardLayoutPanel>
<ContentTemplate><div>0</div></ContentTemplate>
</DashboardLayoutPanel>
<DashboardLayoutPanel SizeX=2 SizeY=2 Column=1>
<ContentTemplate><div>1</div></ContentTemplate>
</DashboardLayoutPanel>
<DashboardLayoutPanel SizeY=2 Column=3>
<ContentTemplate><div>2</div></ContentTemplate>
</DashboardLayoutPanel>
<DashboardLayoutPanel Row=1>
<ContentTemplate><div>3</div></ContentTemplate>
</DashboardLayoutPanel>
</DashboardLayoutPanels>
```

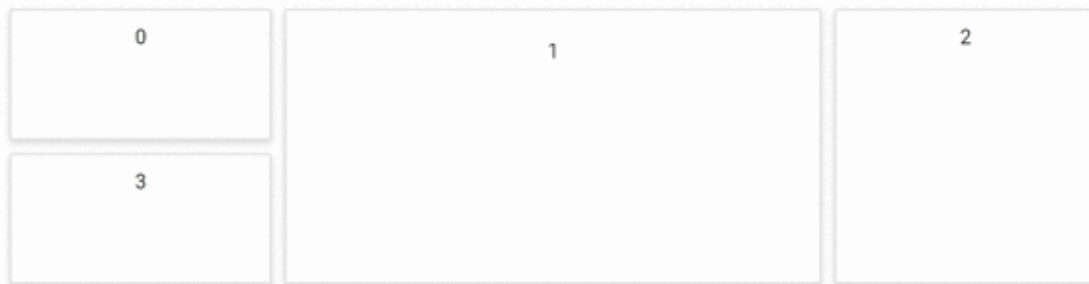
```

</SfDashboardLayout>
<style>
.e-panel-content {
text-align: center;
margin-top: 10px;
}
</style>

```

The above sample demonstrates dragging and pushing of panels. For example, while dragging the panel 0 over panel 1, these panels get collided and push the panel 1 towards the feasible direction, so that, the panel 0 gets placed in the panel 1 position.

The following output demonstrates the dragging functionality of dashboard component.



Customizing the dragging handler

The dragging handler for the panels can be customized using the `DraggableHandle` property to restrict the dragging action within a particular element in the panel.

ASPX-CS

```

@using Syncfusion.Blazor.Layouts
<SfDashboardLayout CellSpacing="@ (new double[] {10 ,10 })"
CellAspectRatio="2" Columns="5" DraggableHandle=".e-panel-header">
<DashboardLayoutPanels>
<DashboardLayoutPanel>
<HeaderTemplate><div>Panel 1</div></HeaderTemplate>
</DashboardLayoutPanel>
<DashboardLayoutPanel SizeX=2 SizeY=2 Column=1>
<HeaderTemplate><div>Panel 2</div></HeaderTemplate>
</DashboardLayoutPanel>
<DashboardLayoutPanel SizeY=2 Column=3>
<HeaderTemplate><div>Panel 3</div></HeaderTemplate>
</DashboardLayoutPanel>
<DashboardLayoutPanel Row=1>
<HeaderTemplate><div>Panel 4</div></HeaderTemplate>
</DashboardLayoutPanel>
</DashboardLayoutPanels>

```

```

</SfDashboardLayout>
<style>
.e-panel-header {
background-color: rgba(0, 0, 0, .1);
text-align: center;
}
.e-panel-content {
text-align: center;
margin-top: 10px;
}
</style>

```

The following output demonstrates customizing the dragging handler of the panels, where the dragging action of panel occurs only with the header of the panel.



Resizing Panels in Blazor Dashboard Layout Component

The DashboardLayout component is also provided with the panel resizing functionality, which can be enabled or disabled using the `AllowResizing` property. This functionality allows you to resize the panels dynamically through UI interactions using the resizing handlers, which controls the panel resizing in various directions.

ASPX-CS

```

@using Syncfusion.Blazor.Layouts
<SfDashboardLayout CellSpacing="@ (new double[] {10 ,10 })" Columns="5"
CellAspectRatio="2" AllowResizing="true">
<DashboardLayoutPanels>
<DashboardLayoutPanel>
<ContentTemplate><div>0</div></ContentTemplate>
</DashboardLayoutPanel>
<DashboardLayoutPanel SizeX=2 SizeY=2 Column=1>
<ContentTemplate><div>1</div></ContentTemplate>
</DashboardLayoutPanel>
<DashboardLayoutPanel SizeY=2 Column=3>
<ContentTemplate><div>2</div></ContentTemplate>
</DashboardLayoutPanel>

```



```

<DashboardLayoutPanel Row=1>
<ContentTemplate><div>3</div></ContentTemplate>
</DashboardLayoutPanel>
</DashboardLayoutPanels>
</SfDashboardLayout>
<style>
.e-panel-content {
text-align: center;
margin-top: 10px;
}
</style>

```

The following output demonstrates the Resizing behavior of Dashboard layout component.



Initially, the panels can be resized only in south-east direction. However, panels can also be resized in east, west, north, south, and south-west directions by defining the required directions with the `ResizableHandles` property.

Floating Panels in Blazor Dashboard Layout Component

The `DashboardLayout` component is also provided with the panel floating functionality that can be enabled or disabled using the `AllowFloating` property. The floating functionality of the component allows to effectively use the entire layout for the panel's placement. If the floating functionality is enabled, the panels within the layout will float upwards automatically to occupy the empty cells available in previous rows.

ASPX-CS

```

@using Syncfusion.Blazor.Layouts
<SfDashboardLayout CellSpacing="@ (new double[] {10 ,10 })" Columns="5"
CellAspectRatio="2" AllowFloating="true">
<DashboardLayoutPanels>
<DashboardLayoutPanel>
<ContentTemplate><div>0</div></ContentTemplate>
</DashboardLayoutPanel>
<DashboardLayoutPanel SizeX=2 SizeY=2 Column=1>
<ContentTemplate><div>1</div></ContentTemplate>

```

```

</DashboardLayoutPanel>
<DashboardLayoutPanel SizeY=2 Column=3>
<ContentTemplate><div>2</div></ContentTemplate>
</DashboardLayoutPanel>
<DashboardLayoutPanel Row=1>
<ContentTemplate><div>3</div></ContentTemplate>
</DashboardLayoutPanel>
</DashboardLayoutPanels>
</SfDashboardLayout>
<style>
.e-panel-content {
text-align: center;
margin-top: 10px;
}
</style>

```

The following output demonstrates the floating behavior of Dashboard layout component.



Responsive and Adaptive Layout in Blazor Dashboard Layout Component

The control is provided with built-in responsive support, where panels within the layout get adjusted based on their parent element's dimensions to accommodate any resolution, which relieves the burden of building responsive dashboards.

The dashboard layout is designed to automatically adapt with lower resolutions by transforming the entire layout into a stacked one. So that, the panels will be displayed in a vertical column. By default, whenever the screen resolution meets 600px or lower resolutions this layout transformation occurs. This transformation can be modified for any user defined resolution by defining the **MediaQuery** property of the component.

ASPX-CS

```

@using Syncfusion.Blazor.Layouts
<SfDashboardLayout CellSpacing="@ (new double[] {20 ,20 })" Columns="5"
MediaQuery="max-width:700px">
<DashboardLayoutPanels>
<DashboardLayoutPanel>

```

```

<ContentTemplate><div>0</div></ContentTemplate>
</DashboardLayoutPanel>
<DashboardLayoutPanel SizeX=2 SizeY=2 Column=1>
<ContentTemplate><div>1</div></ContentTemplate>
</DashboardLayoutPanel>
<DashboardLayoutPanel SizeY=2 Column=3>
<ContentTemplate><div>2</div></ContentTemplate>
</DashboardLayoutPanel>
<DashboardLayoutPanel Row=1>
<ContentTemplate><div>3</div></ContentTemplate>
</DashboardLayoutPanel>
</DashboardLayoutPanels>
</SfDashboardLayout>
<style>
.e-panel-content {
text-align: center;
margin-top: 10px;
}
</style>

```

The following output will demonstrate the Responsive dashboard layout component.



The above sample demonstrates usage of the **MediaQuery** property to turn out the layout into a stacked one in user defined resolution. Here, whenever the window size reaches 700px or lesser, the layout becomes a stacked layout.

CSS Structure in Blazor Dashboard Layout 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 dashboard layout panel header

Use the following CSS to customize the dashboard layout panel header.

CSS

```

.e-dashboardlayout.e-control .e-panel .e-panel-container .e-panel-header {
color: #754131;
background-color: #c9e2f7;
text-align: center;
}

```

Customizing the dashboard layout panel content

Use the following CSS to customize the dashboard layout panel content.

CSS

```
.e-dashboardlayout.e-control .e-panel .e-panel-container .e-panel-content {  
  background-color: #c9e2f7;  
  padding: 50px;  
}
```

Customizing the dashboard layout panel resize icon

Use the following CSS to customize the dashboard layout resize icon.

CSS

```
.e-dashboardlayout.e-control .e-panel .e-panel-container .e-resize.e-double{  
  color: #0378d5;  
  font-size: 30px;  
  height: 20px;  
  width: 20px;  
}
```

Customizing the dashboard layout panel background

Use the following CSS to customize the dashboard layout panel background.

CSS

```
.e-dashboardlayout.e-control.e-responsive {  
  background: #b3d3ed;  
}
```

Accessibility in Blazor Dashboard Layout Component

The Dashboard Layout component has been designed with the **WAI-ARIA** specifications in mind and applying the **WAI-ARIA** roles, states, and properties. This component is characterized by ARIA accessibility support, which makes navigation easy for people who use assistive technologies (AT).

ARIA attributes

The following **ARIA** Attribute denote the Dashboard Layout state.

| **Property** | **Functionalities** |

| --- | --- |

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

DataManager

<!-- markdownlint-disable MD024 -->

Getting Started with Blazor DataManager Component

This section explains about how to connect the [SfDataManager](#) to a data source and perform queries on it in your Blazor Server-Side application. You can refer [Getting Started with Syncfusion Blazor for Server-Side in Visual Studio page](#) for the introduction and configuring the common specifications.

Importing Syncfusion Blazor component in the application

1. Install the **Syncfusion.Blazor** NuGet package to the application by using the **NuGet Package Manager**.
2. You can add the client-side resources through [CDN](#) or from [NuGet](#) package in the `element` of the `~/Pages/_Host.cshtml` page.

ASPX-CS

```
<head>
<environment include="Development">
    ....
    ....
<link href="_content/Syncfusion.Blazor/styles/fabric.css" rel="stylesheet"
/>
<!--CDN-->
@*<link href="https://cdn.syncfusion.com/blazor/{{ site.blazorversion
}}/styles/fabric.css" rel="stylesheet" />*@
</environment>
</head>
```

Adding component package to the application

Open `~/_Imports.razor` file and import the **Syncfusion.Blazor.Data** package.

ASPX-CS

```
@using Syncfusion.Blazor.Data
```

Add Data Manager Component

To initialize the [SfDataManager](#) component, add the below code to your **Index.razor** view page which is present under `~/Pages` folder.

ASPX-CS

```
<SfDataManager>
</SfDataManager>
```

Since the [SfDataManager](#) component is mainly used in conjunction with Syncfusion Blazor components that supports data binding, we are going to use Blazor DataGrid component to depict the usage of [SfDataManager](#) throughout this documentation.

Connection to a data source

The DataManager acts as a gateway for both local and remote data to interact with the data source based on the provided query.

Binding to JSON data

Local JSON data can be bound to the DataGrid component by assigning the array of objects to the [Json](#) property of the [SfDataManager](#) component.

The following sample code demonstrates binding local data through the [SfDataManager](#) to the DataGrid component,

ASPX-CS

```
@using Syncfusion.Blazor.Data
@using Syncfusion.Blazor.Grids
<SfGrid TValue="EmployeeData" ID="Grid">
<SfDataManager Json=@Employees></SfDataManager>
<GridColumns>
<GridColumn Field=@nameof(EmployeeData.EmployeeID)
TextAlign="TextAlign.Center" HeaderText="Employee ID"
Width="120"></GridColumn>
<GridColumn Field=@nameof(EmployeeData.Name) HeaderText="First Name"
Width="130"></GridColumn>
<GridColumn Field=@nameof(EmployeeData.Title) HeaderText="Title"
Width="120"></GridColumn>
</GridColumns>
</SfGrid>
@code{
public class EmployeeData
{
public int EmployeeID { get; set; }
public string Name { get; set; }
public string Title { get; set; }
}
public EmployeeData[] Employees = new EmployeeData[]
{
new EmployeeData { EmployeeID = 1, Name = "Nancy Fuller", Title = "Vice
President" },
new EmployeeData { EmployeeID = 2, Name = "Steven Buchanan", Title = "Sales
Manager" },
new EmployeeData { EmployeeID = 3, Name = "Janet Leverling", Title = "Sales
Representative" },
new EmployeeData { EmployeeID = 4, Name = "Andrew Davolio", Title = "Inside
Sales Coordinator" },
new EmployeeData { EmployeeID = 5, Name = "Steven Peacock", Title = "Inside
Sales Coordinator" },
new EmployeeData { EmployeeID = 6, Name = "Janet Buchanan", Title = "Sales
Representative" },
new EmployeeData { EmployeeID = 7, Name = "Andrew Fuller", Title = "Inside
Sales Coordinator" },
new EmployeeData { EmployeeID = 8, Name = "Steven Davolio", Title = "Inside
Sales Coordinato" },
new EmployeeData { EmployeeID = 9, Name = "Janet Davolio", Title = "Sales
Representative" },
new EmployeeData { EmployeeID = 10, Name = "Andrew Buchanan", Title = "Sales
Representative" }
};
}
```

Binding to OData

Remote data can be bound to the DataGrid component by binding the [SfDataManager](#) component to it and then assigning the service end point URL to the [Url](#) property of the [SfDataManager](#).

The following sample code demonstrates binding OData through the [SfDataManager](#) to the DataGrid component,

ASPX-CS

```
@using Syncfusion.Blazor
@using Syncfusion.Blazor.Data
@using Syncfusion.Blazor.Grids
<SfGrid TValue="Order" ID="Grid" AllowPaging="true">
  <SfDataManager
    Url="https://js.syncfusion.com/demos/ejServices/Wcf/Northwind.svc/Orders"
    Adaptor="Adaptors.ODataAdaptor"></SfDataManager>
  <GridColumn>
    <GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
      IsPrimaryKey="true" TextAlign="TextAlign.Right" Width="120"></GridColumn>
    <GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
      Width="150"></GridColumn>
    <GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
      Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
      Width="130"></GridColumn>
    <GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
      TextAlign="TextAlign.Right" Width="120"></GridColumn>
  </GridColumn>
</SfGrid>
@code{
  public class Order {
    public int? OrderID { get; set; }
    public string CustomerID { get; set; }
    public DateTime? OrderDate { get; set; }
    public double? Freight { get; set; }
  }
}
```

Component binding

As mentioned, the [SfDataManager](#) can be used with Syncfusion components which supports data binding.

Here, the [SfDataManager](#) is bound with DropDownList component to demonstrate data binding for the components.

Local data binding

Local data can be bound to the DropDownList component by assigning the array of objects to the [Json](#) property of the [SfDataManager](#) component.

The following sample code demonstrates binding local data through the [SfDataManager](#) to the DropDownList component,

ASPX-CS

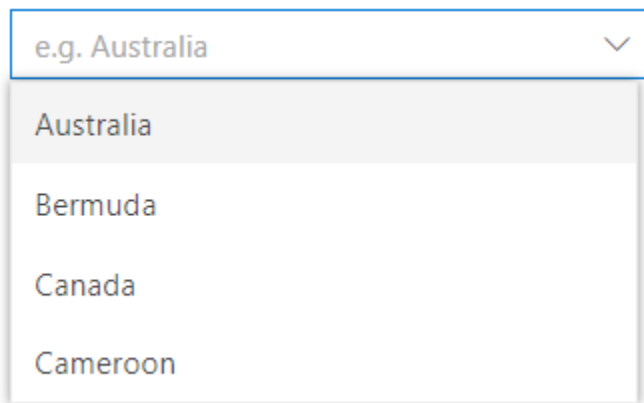
```
@using Syncfusion.Blazor.Data
@using Syncfusion.Blazor.DropDowns
<SfDropDownList Placeholder="e.g. Australia" TValue="Countries">
```

```

<SfDataManager Json=@Country></SfDataManager>
<DropDownListFieldSettings Value="Name"></DropDownListFieldSettings>
</SfDropDownList>
@code{
public class Countries
{
public string Name { get; set; }
public string Code { get; set; }
}
public Countries[] Country = new Countries[]
{
new Countries { Name = "Australia", Code = "AU" },
new Countries { Name = "Bermuda", Code = "BM" },
new Countries { Name = "Canada", Code = "CA" },
new Countries { Name = "Cameroon", Code = "CM" }
};
}

```

The following image represents DropDownList bound with local data through the **SfDataManager** component,



Remote data binding

Remote data can be bound to the DropDownList component by binding the [SfDataManager](#) component to it and then assigning the service end point URL to the [Url](#) property of the [SfDataManager](#).

The following sample code demonstrates binding remote data through the [SfDataManager](#) to the DropDownList component,

ASPX-CS

```

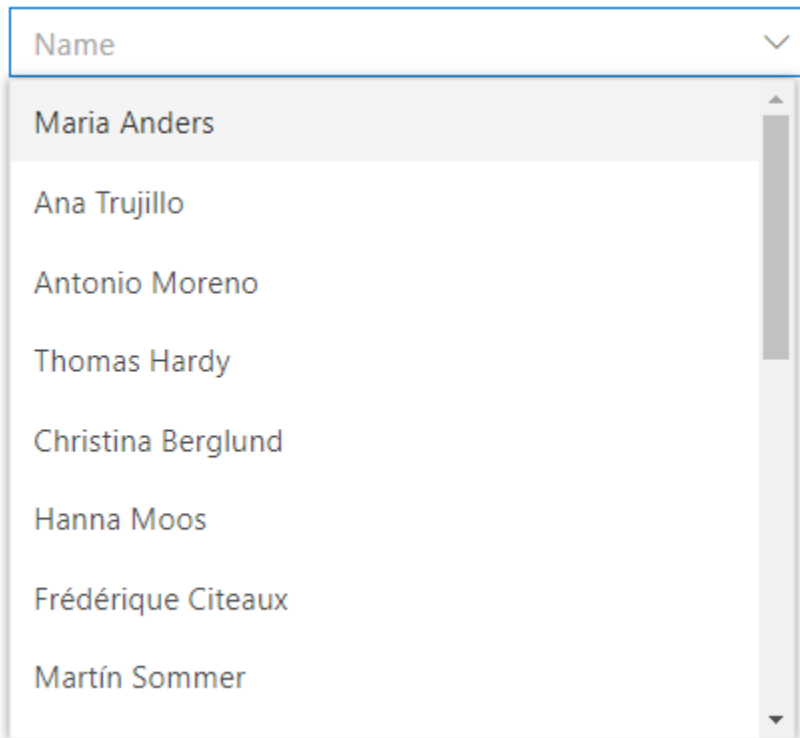
@using Syncfusion.Blazor
@using Syncfusion.Blazor.Data
@using Syncfusion.Blazor.DropDowns
<SfDropDownList Placeholder="Name" TValue="Contact">
<SfDataManager
Url="https://services.odata.org/V4/Northwind/Northwind.svc/Customers"
Adaptor="Adaptors.ODataV4Adaptor"></SfDataManager>
<DropDownListFieldSettings Value="CustomerID"
Text="ContactName"></DropDownListFieldSettings>
</SfDropDownList>

```



```
@code{
public class Contact
{
public string ContactName { get; set; }
public string CustomerID { get; set; }
}
}
```

The following image represents DropDownList bound with remote data through the **SfDataManager** component,



<!-- markdownlint-disable MD024 -->

Data Binding in Blazor DataManager Component

The [SfDataManager](#) acts as a gateway for both local and remote data source which uses the query to interact with the data source. It supports both local object binding and RESTful JSON data services binding.

Local data binding

Local data can be bound to the DataGrid component by assigning the array of objects to the [Json](#) property of [SfDataManager](#).

The following sample code demonstrates binding local data through the [SfDataManager](#) to the DataGrid component,

ASPX-CS

```
@using Syncfusion.Blazor.Data
@using Syncfusion.Blazor.Grids
```

```
<SfGrid TValue="EmployeeData" ID="Grid">
<SfDataManager Json=@Employees></SfDataManager>
<GridColumn>
<GridColumn Field=@nameof(EmployeeData.EmployeeID)
TextAlign="TextAlign.Center" HeaderText="Employee ID"
Width="120"></GridColumn>
<GridColumn Field=@nameof(EmployeeData.Name) HeaderText="First Name"
Width="130"></GridColumn>
<GridColumn Field=@nameof(EmployeeData.Title) HeaderText="Title"
Width="120"></GridColumn>
</GridColumn>
</SfGrid>
@code{
public class EmployeeData
{
public int EmployeeID { get; set; }
public string Name { get; set; }
public string Title { get; set; }
}
public EmployeeData[] Employees = new EmployeeData[]
{
new EmployeeData { EmployeeID = 1, Name = "Nancy Fuller", Title = "Vice
President" },
new EmployeeData { EmployeeID = 2, Name = "Steven Buchanan", Title = "Sales
Manager" },
new EmployeeData { EmployeeID = 3, Name = "Janet Leverling", Title = "Sales
Representative" },
new EmployeeData { EmployeeID = 4, Name = "Andrew Davolio", Title = "Inside
Sales Coordinator" },
new EmployeeData { EmployeeID = 5, Name = "Steven Peacock", Title = "Inside
Sales Coordinator" },
new EmployeeData { EmployeeID = 6, Name = "Janet Buchanan", Title = "Sales
Representative" },
new EmployeeData { EmployeeID = 7, Name = "Andrew Fuller", Title = "Inside
Sales Coordinator" },
new EmployeeData { EmployeeID = 8, Name = "Steven Davolio", Title = "Inside
Sales Coordinato" },
new EmployeeData { EmployeeID = 9, Name = "Janet Davolio", Title = "Sales
Representative" },
new EmployeeData { EmployeeID = 10, Name = "Andrew Buchanan", Title = "Sales
Representative" }
};
}
```

The following image represents DataGrid bound with local data through the **SfDataManager** component,

Employee ID	First Name	Title
1	Nancy Fuller	Vice President
2	Steven Buchanan	Sales Manager
3	Janet Leverling	Sales Representative
4	Andrew Davolio	Inside Sales Coordinator
5	Steven Peacock	Inside Sales Coordinator
6	Janet Buchanan	Sales Representative
7	Andrew Fuller	Inside Sales Coordinator
8	Steven Davolio	Inside Sales Coordinato
9	Janet Davolio	Sales Representative
10	Andrew Buchanan	Sales Representative

Remote data binding

Remote data can be bound to the Grid component by binding the [SfDataManager](#) component to it and then assigning the service end point URL to the [Url](#) property of the [SfDataManager](#).

The following sample code demonstrates binding remote data through the [SfDataManager](#) to the [DataGrid](#) component,

ASPX-CS

```
@using Syncfusion.Blazor
@using Syncfusion.Blazor.Data
@using Syncfusion.Blazor.Grids
<SfGrid TValue="Order" ID="Grid" AllowPaging="true">
  <SfDataManager
    Url="https://js.syncfusion.com/demos/ejServices/Wcf/Northwind.svc/Orders"
    Adaptor="Adaptors.ODataAdaptor"></SfDataManager>
  <GridColumn>
    <GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
      IsPrimaryKey="true" TextAlign="TextAlign.Right" Width="120"></GridColumn>
    <GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
      Width="150"></GridColumn>
    <GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
      Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
      Width="130"></GridColumn>
    <GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
      TextAlign="TextAlign.Right" Width="120"></GridColumn>
  </GridColumn>
</SfGrid>
@code{
  public class Order {
    public int? OrderID { get; set; }
    public string CustomerID { get; set; }
    public DateTime? OrderDate { get; set; }
  }
```

```
public double? Freight { get; set; }
}
```

The following image represents DataGrid bound with remote data through the **SfDataManager** component,

Order ID	Customer Name	Order Date	Freight
10248	VINET	7/4/1996	32.3800
10249	TOMSP	7/5/1996	11.6100
10250	HANAR	7/8/1996	65.8300
10251	VICTE	7/8/1996	41.3400
10252	SUPRD	7/9/1996	51.3000
10253	HANAR	7/10/1996	58.1700
10254	CHOPS	7/11/1996	22.9800
10255	RICSU	7/12/1996	148.3300

K
<
1
2
3
4
5
6
7
8
9
10
...
>
X
1 of 104 pages (830 items)

<!-- markdownlint-disable MD024 -->

Adaptors in Blazor DataManager Component

Each data source or remote service uses different way for accepting request and sending back the response. The **SfDataManager** cannot anticipate every way a data source works. To tackle this problem the **SfDataManager** uses adaptor concept to communicate with the particular data source.

- For local data sources, the role of the data adaptor is to query the object array based on the Query object and manipulate them.
- For remote data source, the data adaptor is used to send the request that the server can understand which then processes the server response.

The adaptor can be assigned using the **Adaptor** property of the **SfDataManager**.

Json adaptor

The **JsonAdaptor** is used to query and manipulate object array.

The following sample code demonstrates binding data to the DataGrid component through the **SfDataManager** using **JsonAdaptor**,

ASPX-CS

```
@using Syncfusion.Blazor
@using Syncfusion.Blazor.Data
@using Syncfusion.Blazor.Grids
```

```
<SfGrid TValue="EmployeeData" ID="Grid">
  <SfDataManager Json=@Employees
  Adaptor="Adaptors.JsonAdaptor"></SfDataManager>
  <GridColumn>
  <GridColumn Field=@nameof(EmployeeData.EmployeeID)
  TextAlign="TextAlign.Center" HeaderText="Employee ID"
  Width="120"></GridColumn>
  <GridColumn Field=@nameof(EmployeeData.Name) HeaderText="First Name"
  Width="130"></GridColumn>
  <GridColumn Field=@nameof(EmployeeData.Title) HeaderText="Title"
  Width="120"></GridColumn>
  </GridColumn>
</SfGrid>
@code{
public class EmployeeData
{
public int EmployeeID { get; set; }
public string Name { get; set; }
public string Title { get; set; }
}
public EmployeeData[] Employees = new EmployeeData[]
{
new EmployeeData { EmployeeID = 1, Name = "Nancy Fuller", Title = "Vice
President" },
new EmployeeData { EmployeeID = 2, Name = "Steven Buchanan", Title = "Sales
Manager" },
new EmployeeData { EmployeeID = 3, Name = "Janet Leverling", Title = "Sales
Representative" },
new EmployeeData { EmployeeID = 4, Name = "Andrew Davolio", Title = "Inside
Sales Coordinator" },
new EmployeeData { EmployeeID = 5, Name = "Steven Peacock", Title = "Inside
Sales Coordinator" },
new EmployeeData { EmployeeID = 6, Name = "Janet Buchanan", Title = "Sales
Representative" },
new EmployeeData { EmployeeID = 7, Name = "Andrew Fuller", Title = "Inside
Sales Coordinator" },
new EmployeeData { EmployeeID = 8, Name = "Steven Davolio", Title = "Inside
Sales Coordinato" },
new EmployeeData { EmployeeID = 9, Name = "Janet Davolio", Title = "Sales
Representative" },
new EmployeeData { EmployeeID = 10, Name = "Andrew Buchanan", Title = "Sales
Representative" }
};
}
```

Url adaptor

The `UrlAdaptor` acts as the base adaptor for interacting with remote data services. Most of the built-in adaptors are derived from the `UrlAdaptor`.

The following sample code demonstrates binding data to the DataGrid component through the [SfDataManager](#) using `UrlAdaptor`,

ASPX-CS

```
@using Syncfusion.Blazor
@using Syncfusion.Blazor.Data
```

```

@using Syncfusion.Blazor.Grids
<SfGrid TValue="EmployeeData" ID="Grid" AllowPaging="true">
<SfDataManager Url="http://controller.com/actions"
Adaptor="Adaptors.UrlAdaptor"></SfDataManager>
<GridColumn>
<GridColumn Field=@nameof(EmployeeData.EmployeeID)
TextAlign="TextAlign.Center" HeaderText="Employee ID"
Width="120"></GridColumn>
<GridColumn Field=@nameof(EmployeeData.Name) HeaderText="First Name"
Width="130"></GridColumn>
<GridColumn Field=@nameof(EmployeeData.Title) HeaderText="Title"
Width="120"></GridColumn>
</GridColumn>
</SfGrid>
@code{
public class EmployeeData
{
public int EmployeeID { get; set; }
public string Name { get; set; }
public string Title { get; set; }
}
}

```

The above mentioned URL is given for reference purposes. In that place, you can provide your service URL.

UrlAdaptor expects response as a JSON object with properties `result` and `count` which contains the collection of entities and the total number of records respectively.

The sample response object should be as follows,

ASPX-CS

```

@using Syncfusion.Blazor
@using Syncfusion.Blazor.Data
@using Syncfusion.Blazor.Grids
<SfGrid TValue="EmployeeData" ID="Grid" AllowPaging="true">
<SfDataManager
Url="https://js.syncfusion.com/demos/ejServices/Wcf/Northwind.svc/Orders"
Adaptor="Adaptors.ODataAdaptor"></SfDataManager>
<GridColumn>
<GridColumn Field=@nameof(EmployeeData.OrderID) TextAlign="TextAlign.Center"
HeaderText="Order ID" Width="120"></GridColumn>
<GridColumn Field=@nameof(EmployeeData.CustomerID)
TextAlign="TextAlign.Center" HeaderText="Customer Name"
Width="130"></GridColumn>
<GridColumn Field=@nameof(EmployeeData.EmployeeID)
TextAlign="TextAlign.Center" HeaderText="Employee ID"
Width="120"></GridColumn>
</GridColumn>
</SfGrid>
@code{
public class EmployeeData
{
public int OrderID { get; set; }
public string CustomerID { get; set; }
}
}

```

```
public int EmployeeID { get; set; }  
}  
}
```

By default, **ODataAdaptor** is used by DataManager.

ODataV4 adaptor

The ODataV4 is an improved version of OData protocols and the [SfDataManager](#) can also retrieve and consume OData v4 services. For more details on OData v4 Services, refer the [odata documentation](#). You can use the **ODataV4Adaptor** to interact with ODataV4 service.

The following sample code demonstrates binding remote data to the DataGrid component through the [SfDataManager](#) using ODataV4 service,

ASPX-CS

```
@using Syncfusion.Blazor  
@using Syncfusion.Blazor.Data  
@using Syncfusion.Blazor.Grids  
<SfGrid TValue="EmployeeData" ID="Grid" AllowPaging="true">  
  <SfDataManager  
    Url="https://services.odata.org/V4/Northwind/Northwind.svc/Orders/"  
    Adaptor="Adaptors.ODataV4Adaptor"></SfDataManager>  
  <GridColumns>  
    <GridColumn Field=@nameof(EmployeeData.OrderID) TextAlign="TextAlign.Center"  
      HeaderText="Order ID" Width="120"></GridColumn>  
    <GridColumn Field=@nameof(EmployeeData.CustomerID)  
      TextAlign="TextAlign.Center" HeaderText="Customer Name"  
      Width="130"></GridColumn>  
    <GridColumn Field=@nameof(EmployeeData.EmployeeID)  
      TextAlign="TextAlign.Center" HeaderText="Employee ID"  
      Width="120"></GridColumn>  
  </GridColumns>  
</SfGrid>  
@code{  
  public class EmployeeData  
  {  
    public int OrderID { get; set; }  
    public string CustomerID { get; set; }  
    public int EmployeeID { get; set; }  
  }  
}
```

Web API adaptor

You can use the **WebApiAdaptor** to interact with Web APIs created with OData endpoint. The **WebApiAdaptor** is extended from the **ODataAdaptor**. Hence to use **WebApiAdaptor**, the endpoint should understand the OData formatted queries sent along with request.

To enable OData query option for Web API, please refer to this [documentation](#).

The following sample code demonstrates binding remote data to the DataGrid component through the [SfDataManager](#) using Web API service,

ASPX-CS

```

@using Syncfusion.Blazor
@using Syncfusion.Blazor.Data
@using Syncfusion.Blazor.Grids
<SfGrid TValue="Order" AllowPaging="true">
<SfDataManager Url="https://ej2services.syncfusion.com/production/web-
services/api/Orders" Adaptor="Adaptors.WebApiAdaptor"></SfDataManager>
<GridColumn>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
IsPrimaryKey="true" TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText="Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumn>
</SfGrid>
@code{
public class Order
{
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
}

```

WebApiAdaptor expects JSON response from the server and the response object should contain properties **Items** and **Count** whose values are collection of entities and total count of the entities respectively.

The sample response object should look like below.

CSHARP

```

{
  "Items": [{..}, {..}, {..}, ...],
  "Count": 830
}

```

WebMethod adaptor

You can use the **WebApiAdaptor** to interact with Web APIs created with OData endpoint. The **WebApiAdaptor** is extended from the **ODataAdaptor**. Hence to use **WebApiAdaptor**, the endpoint should understand the OData formatted queries sent along with request.

To enable OData query option for Web API, please refer to this [documentation](#).

The following sample code demonstrates binding remote data to the DataGrid component through the [SfDataManager](#) using Web method service,

ASPX-CS

```

@using Syncfusion.Blazor
@using Syncfusion.Blazor.Data

```



```
@using Syncfusion.Blazor.Grids
<SfGrid TValue="EmployeeData" ID="Grid">
<SfDataManager Url="https://demoUrl/action"
Adaptor="Adaptors.WebMethodAdaptor"></SfDataManager>
<GridColumn>
<GridColumn Field=@nameof(EmployeeData.EmployeeID) HeaderText="Employee ID"
Width="120"></GridColumn>
<GridColumn Field=@nameof(EmployeeData.FirstName) HeaderText="First Name"
Width="130"></GridColumn>
<GridColumn Field=@nameof(EmployeeData.Designation) HeaderText="Designation"
Width="120"></GridColumn>
<GridColumn Field=@nameof(EmployeeData.Country) HeaderText="Country"
Width="120"></GridColumn>
</GridColumn>
</SfGrid>
@code{
public class EmployeeData
{
public int EmployeeID { get; set; }
public string FirstName { get; set; }
public string Designation { get; set; }
public string Country { get; set; }
}
}
```

The above mentioned URL is given for reference purposes. In that place, you can provide your service URL.

WebMethodAdaptor expects JSON response from the server and the response object should contain properties **result** and **count** whose values are collection of entities and total count of the entities respectively.

The sample response object should look like below.

CSHARP

```
{
  "result": [{..}, {..}, {..}, ...],
  "count": 830
}
```

The controller method's data parameter name must be **value**.

Writing custom adaptor

Sometimes the built-in adaptors do not meet your requirements and in such cases you can create your own adaptor.

To create and use custom adaptor, please follow the following steps,

- Create a class which extended from **DataAdaptor** class. **DataAdaptor** class will act as base class for your custom adaptor.
- Override the desired method to achieve your requirement.
- Assign the custom adaptor class to the **AdaptorInstance** property of **SfDataManager** component.

You can refer to this [link](#) for more details on the working of custom adaptor.

<!-- markdownlint-disable MD024 -->

Custom Binding in Blazor DataManager Component

The [SfDataManager](#) has custom adaptor support which allows you to perform manual operations on the data. This is demonstrated below by implementing custom data binding and editing operations in the DataGrid component.

For implementing custom data binding in DataGrid, the **DataAdaptor** class is used. This abstract class acts as a base class for the custom adaptor.

The **DataAdaptor** abstract class has both synchronous and asynchronous method signatures which can be overridden in the custom adaptor. Following are the method signatures present in this class,

CSHARP

```
public abstract class DataAdaptor
{
    /// <summary>
    /// Performs data Read operation synchronously.
    /// </summary>
    public virtual object Read(DataManagerRequest dataManagerRequest, string key
    = null)
    /// <summary>
    /// Performs data Read operation asynchronously.
    /// </summary>
    public virtual Task<object> ReadAsync(DataManagerRequest dataManagerRequest,
    string key = null)
    /// <summary>
    /// Performs Insert operation synchronously.
    /// </summary>
    public virtual object Insert(DataManager dataManager, object data, string
    key)
    /// <summary>
    /// Performs Insert operation asynchronously.
    /// </summary>
    public virtual Task<object> InsertAsync(DataManager dataManager, object
    data, string key)
    /// <summary>
    /// Performs Remove operation synchronously.
    /// </summary>
    public virtual object Remove(DataManager dataManager, object data, string
    keyField, string key)
    /// <summary>
    /// Performs Remove operation asynchronously.
    /// </summary>
    public virtual Task<object> RemoveAsync(DataManager dataManager, object
    data, string keyField, string key)
    /// <summary>
    /// Performs Update operation synchronously.
    /// </summary>
    public virtual object Update (DataManager dataManager, object data, string
    keyField, string key)
    /// <summary>
    /// Performs Update operation asynchronously.
    /// </summary>
}
```

```

public virtual Task<object> UpdateAsync(DataManager dataManager, object
data, string keyField, string key)
/// <summary>
/// Performs Batch CRUD operations synchronously.
/// </summary>
public virtual object BatchUpdate(DataManager dataManager, object
changedRecords, object addedRecords, object deletedRecords, string keyField,
string key)
/// <summary>
/// Performs Batch CRUD operations asynchronously.
/// </summary>
public virtual Task<object> BatchUpdateAsync(DataManager dataManager, object
changedRecords, object addedRecords, object deletedRecords, string keyField,
string key)
}

```

For implementing the custom data binding alone in the DataGrid component, provide the custom adaptor class and override the **Read** or **ReadAsync** method of the **DataAdaptor** abstract class.

If the Read/ReadAsync method is not overridden in the custom adaptor then it will be handled by the default read handler.

For implementing the CRUD operations for the custom bounded data, override the following CRUD methods of the **DataAdaptor** abstract class,

- Insert/InsertAsync
- Remove/RemoveAsync
- Update/UpdateAsync
- BatchUpdate/BatchUpdateAsync

While using batch editing in datagrid, use BatchUpdate/BatchUpdateAsync method to handle the corresponding CRUD operation

The following sample code demonstrates implementing custom adaptor of the [SfDataManager](#) for data binding and editing operations in the DataGrid component,

ASPX-CS

```

@using Syncfusion.Blazor
@using Syncfusion.Blazor.Data
@using Syncfusion.Blazor.Grids
<SfGrid TValue="Order" ID="Grid" AllowSorting="true" AllowFiltering="true"
AllowPaging="true" Toolbar="@ (new List<string>() { "Add", "Delete",
"Update", "Cancel" })">
<SfDataManager AdaptorInstance="@typeof(CustomAdaptor) "
Adaptor="Adaptors.CustomAdaptor"></SfDataManager>
<GridPageSettings PageSize="8"></GridPageSettings>
<GridEditSettings AllowEditing="true" AllowDeleting="true"
AllowAdding="true" Mode="@EditMode.Normal"></GridEditSettings>
<GridColumn>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
IsPrimaryKey="true" TextAlign="@TextAlign.Center" Width="140"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>

```

```

<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight"
Width="150"></GridColumn>
</GridColumn>
</SfGrid>
@code{
public static List<Order> Orders { get; set; }
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
}).ToList();
}
public class Order
{
public int OrderID { get; set; }
public string CustomerID { get; set; }
public double Freight { get; set; }
}
// Implementing custom adaptor by extending the DataAdaptor class
public class CustomAdaptor : DataAdaptor
{
// Performs data Read operation
public override object Read(DataManagerRequest dm, string key = null)
{
IEnumerable<Order> DataSource = Orders;
if (dm.Search != null && dm.Search.Count > 0)
{
// Searching
DataSource = DataOperations.PerformSearching(DataSource, dm.Search);
}
if (dm.Sorted != null && dm.Sorted.Count > 0)
{
// Sorting
DataSource = DataOperations.PerformSorting(DataSource, dm.Sorted);
}
if (dm.Where != null && dm.Where.Count > 0)
{
// Filtering
DataSource = DataOperations.PerformFiltering(DataSource, dm.Where,
dm.Where[0].Operator);
}
int count = DataSource.Cast<Order>().Count();
if (dm.Skip != 0)
{
//Paging
DataSource = DataOperations.PerformSkip(DataSource, dm.Skip);
}
if (dm.Take != 0)
{
DataSource = DataOperations.PerformTake(DataSource, dm.Take);
}
return dm.RequiresCounts ? new DataResult() { Result = DataSource, Count =
count } : (object)DataSource;
}
}

```

```
}
// Performs Insert operation
public override object Insert(DataManager dm, object value, string key)
{
    Orders.Insert(0, value as Order);
    return value;
}
// Performs Remove operation
public override object Remove(DataManager dm, object value, string keyField,
string key)
{
    Orders.Remove(Orders.Where(or => or.OrderID ==
int.Parse(value.ToString())).FirstOrDefault());
    return value;
}
// Performs Update operation
public override object Update(DataManager dm, object value, string keyField,
string key)
{
    var data = Orders.Where(or => or.OrderID == (value as
Order).OrderID).FirstOrDefault();
    if (data != null)
    {
        data.OrderID = (value as Order).OrderID;
        data.CustomerID = (value as Order).CustomerID;
        data.Freight = (value as Order).Freight;
    }
    return value;
}
// Performs BatchUpdate operation
public override object BatchUpdate(DataManager dm, object Changed, object
Added, object Deleted, string KeyField, string Key)
{
    if (Changed != null)
    {
        foreach (var rec in (IEnumerable<Order>)Changed)
        {
            Orders[0].CustomerID = rec.CustomerID;
        }
    }
    if (Added != null)
    {
        foreach (var rec in (IEnumerable<Order>)Added)
        {
            Orders.Add(rec);
        }
    }
    if (Deleted != null)
    {
        foreach (var rec in (IEnumerable<Order>)Deleted)
        {
            Orders.RemoveAt(0);
        }
    }
    return Orders;
}
}
```

```
}

```

If the **DataManagerRequest.RequiresCounts** value is **true**, then the Read/ReadAsync return value must be of **DataResult** with properties **Result** whose value is a collection of records and **Count** whose value is the total number of records. If the **DataManagerRequest.RequiresCounts** is **false**, then simply send the collection of records.

The following GIF demonstrates the Grid component with data bound using custom adaptor and the CRUD operations being performed on it,

+ Add 🗑 Delete 🔄 Update ✕ Cancel		
Order ID	Customer Name	Freight
<input type="text"/>	<input type="text"/>	<input type="text"/>
1001	ANTON	2.1
1002	BLONP	4.2
1003	ANTON	6.3000000000000001
1004	BLONP	8.4
1005	ANTON	10.5
1006	BOLID	12.6000000000000001
1007	ALFKI	14.7000000000000001
1008	ALFKI	16.8
K < 1 2 3 4 5 6 7 8 9 10 > ✕		1 of 10 pages (75 items)

How To

<!-- markdownlint-disable MD024 -->

Adding custom headers in Blazor DataManager Component

Custom headers can be added to the [SfDataManager](#) request using its [Headers](#) property.

The following sample code demonstrates adding custom headers to the [SfDataManager](#) request which is bound with the DataGrid component,

ASPX-CS

```
@using Syncfusion.Blazor
@using Syncfusion.Blazor.Data
@using Syncfusion.Blazor.Grids
<SfGrid TValue="Order" AllowPaging="true">
  <GridPageSettings PageSize="10"></GridPageSettings>

```

```

<SfDataManager Headers=@HeaderData
Url="https://ej2services.syncfusion.com/production/web-services/api/Orders"
Adaptor="Adaptors.WebApiAdaptor"></SfDataManager>
<GridColumn>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
IsPrimaryKey="true" TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumn>
</SfGrid>
@code{
public class Data
{
public string Grid;
};
private IDictionary<string, string> HeaderData = new Dictionary<string,
string>();
public class Order
{
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
}

```

<!-- markdownlint-disable MD024 -->

Working in offline mode in Blazor DataManager Component

On binding data through remote services, request will be sent to the server-side for every query. To avoid post back to server, you can set the [SfDataManager](#) to load all the data on initialization itself and make the query processing in client-side. This behavior can be enabled by using [Offline](#) property of the [SfDataManager](#).

The following sample code demonstrates enabling offline mode for the [SfDataManager](#) which is bound with the DataGrid component,

ASPX-CS

```

@using Syncfusion.Blazor
@using Syncfusion.Blazor.Data
@using Syncfusion.Blazor.Grids
<SfGrid TValue="EmployeeData" ID="Grid" AllowPaging="true">
<SfDataManager
Url="https://js.syncfusion.com/demos/ejServices/Wcf/Northwind.svc/Orders"
Adaptor="Adaptors.ODataAdaptor" Offline="true"></SfDataManager>
<GridColumn>
<GridColumn Field=@nameof(EmployeeData.OrderID) TextAlign="TextAlign.Center"
HeaderText="Order ID" Width="120"></GridColumn>

```

```

<GridColumn Field=@nameof(EmployeeData.CustomerID)
TextAlign="TextAlign.Center" HeaderText="Customer Name"
Width="130"></GridColumn>
<GridColumn Field=@nameof(EmployeeData.EmployeeID)
TextAlign="TextAlign.Center" HeaderText="Employee ID"
Width="120"></GridColumn>
</GridColumn>
</SfGrid>
@code{
public class EmployeeData
{
public int OrderID { get; set; }
public string CustomerID { get; set; }
public int EmployeeID { get; set; }
}
}

```

Return the complete list from server-side when using **Offline** property.

DataGrid

<!-- markdownlint-disable MD024 -->

Getting Started with Blazor DataGrid Component

This section briefly explains about how to include a [Blazor DataGrid](#) Component in your Blazor Server-Side and Client-Side application. You can refer to our Getting Started with [Blazor Server-Side DataGrid](#) and [Blazor WebAssembly DataGrid](#) documentation pages for configuration specifications.

To get start quickly with Blazor DataGrid component, you can check on this video.

{% youtube

"youtube:https://www.youtube.com/watch?v=CIAlwPUv0_s" %}

Importing Syncfusion Blazor component in the application

1. Install the **Syncfusion.Blazor.Grid** NuGet package to the application by using the **NuGet Package Manager**.
2. You can add the client-side resources through [CDN](#) or from [NuGet](#) package in the **HEAD** element of the **~/Pages/_Host.cshtml** page.

HTML

```

<head>
<environment include="Development">
....
....
<link href="_content/Syncfusion.Blazor.Themes/fabric.css" rel="stylesheet"
/>
</environment>
</head>

```

For Internet Explorer 11 kindly refer the polyfills. Refer the [documentation](#) for more information.

Adding component package to the application

Open `~/_Imports.razor` file and import the **Syncfusion.Blazor.Grids** package.

ASPX-CS

```
@using Syncfusion.Blazor.Grids
```

Add SyncfusionBlazor service in Startup.cs

Open the **Startup.cs** file and add services required by Syncfusion components using **services.AddSyncfusionBlazor()** method. Add this method in the **ConfigureServices** function as follows.

C#

```
using Syncfusion.Blazor;
namespace BlazorApplication
{
    public class Startup
    {
        ....
        ....
        public void ConfigureServices(IServiceCollection services)
        {
            ....
            ....
            services.AddSyncfusionBlazor();
        }
    }
}
```

Add DataGrid Component

To initialize the DataGrid component add the below code to your **Index.razor** view page which is present under `~/Pages` folder.

CSHARP

```
<SfGrid>
</SfGrid>
```

Defining Row Data

To bind data for the DataGrid component, you can assign a **IEnumerable** object to the [DataSource](#) property. The list data source can also be provided as an instance of the **DataManager**. You can assign the data source through the **OnInitialized** life cycle of the page.

ASPX-CS

```
@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders">
</SfGrid>
@code{
    public List<Order> Orders { get; set; }
    protected override void OnInitialized()
    {
        Orders = Enumerable.Range(1, 75).Select(x => new Order()
        {
            OrderID = 1000 + x,
```

```

CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
OrderDate = DateTime.Now.AddDays(-x),
}).ToList();
}
public class Order {
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
}

```

Defining Columns

The columns are automatically generated when columns declaration is empty or undefined while initializing the datagrid.

The DataGrid has an option to define columns using [GridColumns](#) component. In [GridColumns](#) component we have properties to customize columns.

Let's check the properties used here:

- We have added [Field](#) to map with a property name in IEnumerable object.
- We have added [HeaderText](#) to change the title of columns.
- We have used [TextAlign](#) to change the alignment of columns. By default, columns will be left aligned. To change columns to right align, we need to define [TextAlign](#) as **Right**.
- Also, we have used another useful property, [Format](#). Using this, we can format number and date values to standard or custom formats.

ASPX-CS

```

@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders">
  <GridColumns>
    <GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
    TextAlign="TextAlign.Right" Width="120"></GridColumn>
    <GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
    Width="150"></GridColumn>
    <GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
    Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
    Width="130"></GridColumn>
    <GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
    TextAlign="TextAlign.Right" Width="120"></GridColumn>
  </GridColumns>
</SfGrid>
@code{
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,

```

```

CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
OrderDate = DateTime.Now.AddDays(-x),
}).ToList();
}

public class Order {
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
}

```

Enable Paging

The paging feature enables users to view the datagrid record in a paged view. It can be enabled by setting the [AllowPaging](#) property to true. Pager can be customized using the [GridPageSettings](#) component.

ASPX-CS

```

@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders" AllowPaging="true">
<GridPageSettings PageSize="5"></GridPageSettings>
<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumns>
</SfGrid>
@code{
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
OrderDate = DateTime.Now.AddDays(-x),
}).ToList();
}
public class Order {
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
}

```

Enable Sorting

The sorting feature enables you to order the records. It can be enabled by setting the [AllowSorting](#) property as true. Sorting feature can be customized using the [GridSortSettings](#) component.

ASPX-CS

```
@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders" AllowPaging="true" AllowSorting="true">
  <GridPageSettings PageSize="5"></GridPageSettings>
  <GridColumns>
    <GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
      TextAlign="TextAlign.Right" Width="120"></GridColumn>
    <GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
      Width="150"></GridColumn>
    <GridColumn Field=@nameof(Order.OrderDate) HeaderText="Order Date"
      Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
      Width="130"></GridColumn>
    <GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
      TextAlign="TextAlign.Right" Width="120"></GridColumn>
  </GridColumns>
</SfGrid>
@code{
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
  Orders = Enumerable.Range(1, 75).Select(x => new Order()
  {
    OrderID = 1000 + x,
    CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
  })[new Random().Next(5)],
    Freight = 2.1 * x,
    OrderDate = DateTime.Now.AddDays(-x),
  }).ToList();
}
public class Order {
  public int? OrderID { get; set; }
  public string CustomerID { get; set; }
  public DateTime? OrderDate { get; set; }
  public double? Freight { get; set; }
}
}
```

Enable Filtering

The filtering feature enables you to view reduced amount of records based on filter criteria. It can be enabled by setting the [AllowFiltering](#) property as true. Filtering feature can be customized using the [GridFilterSettings](#) component.

ASPX-CS

```
@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders" AllowPaging="true" AllowSorting="true"
  AllowFiltering="true">
  <GridPageSettings PageSize="5"></GridPageSettings>
  <GridColumns>
```

```

<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumns>
</SfGrid>
@code{
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
OrderDate = DateTime.Now.AddDays(-x),
}).ToList();
}
public class Order {
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
}

```

Enable Grouping

The grouping feature enables you to view the datagrid record in a grouped view. It can be enabled by setting the [AllowGrouping](#) property as true. Grouping feature can be customized using the [GridGroupSettings](#) component.

ASPX-CS

```

@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders" AllowPaging="true" AllowSorting="true"
AllowFiltering="true" AllowGrouping="true">
<GridPageSettings PageSize="5"></GridPageSettings>
<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumns>
</SfGrid>
@code{

```

```

public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
    Orders = Enumerable.Range(1, 75).Select(x => new Order()
    {
        OrderID = 1000 + x,
        CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
    })[new Random().Next(5)],
        Freight = 2.1 * x,
        OrderDate = DateTime.Now.AddDays(-x),
    }).ToList();
}
public class Order {
    public int? OrderID { get; set; }
    public string CustomerID { get; set; }
    public DateTime? OrderDate { get; set; }
    public double? Freight { get; set; }
}
}

```

The following image represents datagrid with paging, sorting, filtering and grouping.

Drag a column header here to group its column			
Order ID	Customer Name	Order Date	Freight
1001	BOLID	7/21/2019	\$2.10
1002	ANTON	7/20/2019	\$4.20
1003	BOLID	7/19/2019	\$6.30
1004	BLONP	7/18/2019	\$8.40
1005	ALFKI	7/17/2019	\$10.50
<div> K < 1 2 3 4 5 6 7 8 9 10 ... > X </div> <div>1 of 15 pages (75 items)</div>			

Handling exceptions

Exceptions occurred during grid actions can be handled without stopping application. These error messages or exception details can be acquired using the [OnActionFailure](#) event.

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

We recommend you to bind **OnActionFailure** event during your application development phase, this helps you to find any exceptions. You can pass these exception details to our support team to get solution as early as possible.

The following sample code demonstrates notifying user when server-side exception has occurred during data operation,

ASPX-CS

```

@using Syncfusion.Blazor
@using Syncfusion.Blazor.Data
@using Syncfusion.Blazor.Grids
<span class="error">@ErrorDetails</span>
<SfGrid TValue="Order" AllowPaging="true">
<GridEvents TValue="Order" OnActionFailure="@ActionFailure"></GridEvents>
<GridPageSettings PageSize="10"></GridPageSettings>
<SfDataManager Url="https://some.com/invalidUrl"
Adaptor="Adaptors.WebApiAdaptor"></SfDataManager>
<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
IsPrimaryKey="true" TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumns>
</SfGrid>
<style>
.error {
color: red;
}
</style>
@code{
public string ErrorDetails = "";
public class Order
{
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
public void ActionFailure(FailureEventArgs args)
{
this.ErrorDetails = "Server exception: 404 Not found";
StateHasChanged();
}
}

```

See Also

- [Getting started with Syncfusion Data Grid in Blazor Server Side App using .NET Core CLI](#)
- [Getting Started with Syncfusion Blazor for Client-Side in .NET Core CLI](#)
- [Getting Started with Syncfusion DataGrid in Blazor WebAssembly using Visual Studio 2019](#)

Data Binding in Blazor DataGrid Component

The DataGrid uses [SfDataManager](#), which supports both RESTful JSON data services binding and IEnumerable binding. The [DataSource](#) value can be assigned either with the property values from [SfDataManager](#) or list of business objects.

It supports the following kinds of data binding method:

- List binding
- Remote data

When using [DataSource](#) as `IEnumerable<T>`, component type(`TValue`) will be inferred from its value. When using [SfDataManager](#) for data binding then the **TValue** must be provided explicitly in the datagrid component.

List binding

To bind list binding to the datagrid, you can assign a `IEnumerable` object to the [DataSource](#) property. The list data source can also be provided as an instance of the [SfDataManager](#) or by using [SfDataManager](#) component.

ASPX-CS

```
@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders" AllowPaging="true">
  <GridPageSettings PageSize="5"></GridPageSettings>
  <GridColumns>
    <GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
      TextAlign="TextAlign.Right" Width="120"></GridColumn>
    <GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
      Width="150"></GridColumn>
    <GridColumn Field=@nameof(Order.OrderDate) HeaderText="Order Date"
      Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
      Width="130"></GridColumn>
    <GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
      TextAlign="TextAlign.Right" Width="120"></GridColumn>
  </GridColumns>
</SfGrid>

@code{
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
  Orders = Enumerable.Range(1, 75).Select(x => new Order()
  {
    OrderID = 1000 + x,
    CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
  })[new Random().Next(5)],
    Freight = 2.1 * x,
    OrderDate = DateTime.Now.AddDays(-x),
  }).ToList();
}
public class Order {
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
}
```

By default, [SfDataManager](#) uses **BlazorAdaptor** for list data-binding.

ExpandoObject binding

Grid is a generic component which is strongly bound to a model type. There are cases when the model type is unknown during compile time. In such cases you can bound data to the grid as list of **ExpandoObject**.

To know about **ExpandoObject** data binding in Blazor DataGrid component, you can check on this video.

{% youtube

"youtube:https://www.youtube.com/watch?v=Xhaw3DdHmJk"%}

ExpandoObject can be bound to datagrid by assigning to the [DataSource](#) property. Grid can also perform all kind of supported data operations and editing in ExpandoObject.

ASPX-CS

```
@using Syncfusion.Blazor.Grids
@using System.Dynamic
<SfGrid DataSource="@Orders" AllowPaging="true" Toolbar="@ToolbarItems">
  <GridEditSettings AllowAdding="true" AllowDeleting="true"
  AllowEditing="true"></GridEditSettings>
  <GridColumns>
    <GridColumn Field="OrderID" HeaderText="Order ID" IsPrimaryKey="true"
    TextAlign="TextAlign.Right" Width="120"></GridColumn>
    <GridColumn Field="CustomerID" HeaderText="Customer Name"
    Width="120"></GridColumn>
    <GridColumn Field="Freight" HeaderText="Freight" Format="C2"
    TextAlign="TextAlign.Right" Width="120"></GridColumn>
    <GridColumn Field="OrderDate" HeaderText="Order Date" Format="d"
    TextAlign="TextAlign.Right" Width="130" Type="ColumnType.Date"></GridColumn>
    <GridColumn Field="ShipCountry" HeaderText="Ship Country"
    EditType="EditType.DropDownEdit" Width="150"></GridColumn>
    <GridColumn Field="Verified" HeaderText="Active" DisplayAsCheckBox="true"
    Width="150"></GridColumn>
  </GridColumns>
</SfGrid>
@code {
  public List<ExpandoObject> Orders { get; set; } = new List<ExpandoObject>();
  private List<string> ToolbarItems = new List<string>() { "Add", "Edit",
  "Delete", "Update", "Cancel" };
  protected override void OnInitialized()
  {
    Orders = Enumerable.Range(1, 75).Select((x) =>
    {
      dynamic d = new ExpandoObject();
      d.OrderID = 1000 + x;
      d.CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
      })[new Random().Next(5)];
      d.Freight = (new double[] { 2, 1, 4, 5, 3 })[new Random().Next(5)] * x;
      d.OrderDate = (new DateTime[] { new DateTime(2010, 11, 5), new
      DateTime(2018, 10, 3), new DateTime(1995, 9, 9), new DateTime(2012, 8, 2),
      new DateTime(2015, 4, 11) })[new Random().Next(5)];
      d.ShipCountry = (new string[] { "USA", "UK" })[new Random().Next(2)];
      d.Verified = (new bool[] { true, false })[new Random().Next(2)];
      return d;
    }).Cast<ExpandoObject>().ToList<ExpandoObject>();
  }
}
```

ExpandoObject Complex data binding

You can achieve ExpandoObject complex data binding in the datagrid by using the dot(.) operator in the column.field. In the below examples `CustomerID.Name` and `ShipCountry.Country` are complex data.

ASPX-CS

```
@using Syncfusion.Blazor.Grids
@using System.Dynamic
<SfGrid DataSource="@Orders" AllowPaging="true" AllowFiltering="true"
AllowSorting="true" AllowGrouping="true" Toolbar="@ToolbarItems">
<GridEditSettings AllowAdding="true" AllowDeleting="true"
AllowEditing="true"></GridEditSettings>
<GridColumns>
<GridColumn Field="OrderID" HeaderText="Order ID" IsPrimaryKey="true"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field="CustomerID.Name" HeaderText="Customer Name"
Width="120"></GridColumn>
<GridColumn Field="Freight" HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field="OrderDate" HeaderText=" Order Date" Format="d"
TextAlign="TextAlign.Right" Width="130" Type="ColumnType.Date"></GridColumn>
<GridColumn Field="ShipCountry.Country" HeaderText="Ship Country"
Width="150"></GridColumn>
<GridColumn Field="Verified" HeaderText="Active" DisplayAsCheckBox="true"
Width="150"></GridColumn>
</GridColumns>
</SfGrid>
@code {
public List<ExpandoObject> Orders { get; set; } = new List<ExpandoObject>();
private List<string> ToolbarItems = new List<string>() { "Add", "Edit",
"Delete", "Update", "Cancel" };
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select((x) =>
{
dynamic d = new ExpandoObject();
dynamic customerName = new ExpandoObject();
dynamic countryName = new ExpandoObject();
d.OrderID = 1000 + x;
customerName.Name = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP",
"BOLID" })[new Random().Next(5)];
d.CustomerID = customerName;
d.Freight = (new double[] { 2, 1, 4, 5, 3 })[new Random().Next(5)] * x;
d.OrderDate = (new DateTime[] { new DateTime(2010, 11, 5), new
DateTime(2018, 10, 3), new DateTime(1995, 9, 9), new DateTime(2012, 8, 2),
new DateTime(2015, 4, 11) })[new Random().Next(5)];
countryName.Country = (new string[] { "USA", "UK" })[new Random().Next(2)];
d.ShipCountry = countryName;
d.Verified = (new bool[] { true, false })[new Random().Next(2)];
return d;
}).Cast<ExpandoObject>().ToList<ExpandoObject>();
}
}
```

You can perform the Data operations and CRUD operations for Complex ExpandoObject binding fields too.

The following image represents ExpandoObject complex data binding,

Drag a column header here to group its column						
<div> + Add Edit Delete Update Cancel </div>						
Order ID	Customer Name	Freight	Order Date	Ship Country	Active	
<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	
1001	ALFKI	\$5.00	4/11/2015	USA	<input checked="" type="checkbox"/>	
1002	BOLID	\$6.00	4/11/2015	USA	<input type="checkbox"/>	
1003	BOLID	\$15.00	8/2/2012	UK	<input type="checkbox"/>	
1004	BLONP	\$8.00	10/3/2018	USA	<input checked="" type="checkbox"/>	
1005	ANANTR	\$5.00	10/3/2018	USA	<input type="checkbox"/>	
1006	BLONP	\$30.00	9/9/1995	USA	<input type="checkbox"/>	
1007	ANTON	\$28.00	8/2/2012	UK	<input type="checkbox"/>	
1008	ANTON	\$24.00	8/2/2012	UK	<input type="checkbox"/>	

DynamicObject binding

Grid is a generic component which is strongly bound to a model type. There are cases when the model type is unknown during compile type. In such cases you can bound data to the grid as list of **DynamicObject**.

To know about **DynamicObject** data binding in Blazor DataGrid component, you can check on this video.

{% youtube

"youtube:https://www.youtube.com/watch?v=Xhaw3DdHmJk"%}

DynamicObject can be bound to datagrid by assigning to the [DataSource](#) property. Grid can also perform all kind of supported data operations and editing in DynamicObject.

The [GetDynamicMemberNames](#) method of DynamicObject class must be overridden and return the property names to perform data operation and editing while using DynamicObject.

ASPX-CS

```
@using Syncfusion.Blazor.Grids
@using System.Dynamic
<SfGrid DataSource="@Orders" AllowPaging="true" Toolbar="@ToolbarItems">
  <GridEditSettings AllowAdding="true" AllowDeleting="true"
    AllowEditing="true"></GridEditSettings>
  <GridColumns>
    <GridColumn Field="OrderID" HeaderText="Order ID" IsPrimaryKey="true"
      TextAlign="TextAlign.Right" Width="120"></GridColumn>
    <GridColumn Field="CustomerID" HeaderText="Customer Name"
      Width="150"></GridColumn>
    <GridColumn Field="OrderDate" HeaderText="Order Date" Format="d"
      Type="ColumnType.Date" TextAlign="TextAlign.Right"
      EditType="EditType.DatePickerEdit" Width="130"></GridColumn>
```

```

<GridColumn Field="Freight" HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumn>
</SfGrid>
@code {
private List<string> ToolbarItems = new List<string>() {
"Add", "Edit", "Delete", "Update", "Cancel"};
public List<DynamicDictionary> Orders = new List<DynamicDictionary>() { };
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 1075).Select((x) =>
{
dynamic d = new DynamicDictionary();
d.OrderID = 1000 + x;
d.CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)];
d.Freight = (new double[] { 2, 1, 4, 5, 3 })[new Random().Next(5)] * x;
d.OrderDate = DateTime.Now.AddDays(-x);
return d;
}).Cast<DynamicDictionary>().ToList<DynamicDictionary>();
}
public class DynamicDictionary : DynamicObject
{
Dictionary<string, object> dictionary = new Dictionary<string, object>();
public override bool TryGetMember(GetMemberBinder binder, out object result)
{
string name = binder.Name;
return dictionary.TryGetValue(name, out result);
}
public override bool TrySetMember(SetMemberBinder binder, object value)
{
dictionary[binder.Name] = value;
return true;
}
public override System.Collections.Generic.IEnumerable<string>
GetDynamicMemberNames()
{
return this.dictionary?.Keys;
}
}
}

```

DynamicObject Complex data binding

You can achieve DynamicObject complex data binding in the datagrid by using the dot(.) operator in the column.field. In the below examples `CustomerID.Name` and `ShipCountry.Country` are complex data.

ASPX-CS

```

@using Syncfusion.Blazor.Grids
@using System.Dynamic
<SfGrid DataSource="@Orders" AllowPaging="true" AllowFiltering="true"
AllowSorting="true" AllowGrouping="true" Toolbar="@ToolbarItems">
<GridEditSettings AllowAdding="true" AllowDeleting="true"
AllowEditing="true"></GridEditSettings>
<GridColumn>

```

```

<GridColumn Field="OrderID" HeaderText="Order ID" IsPrimaryKey="true"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field="CustomerID.Name" HeaderText="Customer Name"
Width="150"></GridColumn>
<GridColumn Field="OrderDate" HeaderText="Order Date" Format="d"
Type="ColumnType.Date" TextAlign="TextAlign.Right"
EditType="EditType.DatePickerEdit" Width="130"></GridColumn>
<GridColumn Field="Freight" HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field="ShipCountry.Country" HeaderText="Ship Country"
Width="150"></GridColumn>
</GridColumn>
</SfGrid>
@code {
private List<string> ToolbarItems = new List<string>() { "Add", "Edit",
"Delete", "Update", "Cancel" };
public List<DynamicDictionary> Orders = new List<DynamicDictionary>() { };
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 1075).Select((x) =>
{
dynamic d = new DynamicDictionary();
dynamic combo = new DynamicDictionary();
dynamic countryName = new DynamicDictionary();
d.OrderID = 1000 + x;
combo.Name = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)];
d.CustomerID = combo;
d.Freight = (new double[] { 2, 1, 4, 5, 3 })[new Random().Next(5)] * x;
d.OrderDate = DateTime.Now.AddDays(-x);
countryName.Country = (new string[] { "USA", "UK" })[new Random().Next(2)];
d.ShipCountry = countryName;
return d;
}).Cast<DynamicDictionary>().ToList<DynamicDictionary>();
}
public class DynamicDictionary : DynamicObject
{
Dictionary<string, object> dictionary = new Dictionary<string, object>();
public override bool TryGetMember(GetMemberBinder binder, out object result)
{
string name = binder.Name;
return dictionary.TryGetValue(name, out result);
}
public override bool TrySetMember(SetMemberBinder binder, object value)
{
dictionary[binder.Name] = value;
return true;
}
public override System.Collections.Generic.IEnumerable<string>
GetDynamicMemberNames()
{
return this.dictionary?.Keys;
}
}
}

```

* you can perform the Data operations and CRUD operations for Complex DynamicObject binding fields too.

The following image represents DynamicObject complex data binding

Drag a column header here to group its column

+ Add ✎ Edit 🗑 Delete 🔄 Update ✕ Cancel

Order ID	Customer Name	Order Date	Freight	Ship Country
<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>
1001	BLONP	3/30/2021	\$3.00	UK
1002	BLONP	3/29/2021	\$4.00	UK
1003	ANANTR	3/28/2021	\$6.00	UK
1004	ANANTR	3/27/2021	\$16.00	USA
1005	BLONP	3/26/2021	\$10.00	USA
1006	ALFKI	3/25/2021	\$18.00	USA
1007	ANTON	3/24/2021	\$21.00	UK
1008	BLONP	3/23/2021	\$8.00	UK

Remote data

To bind remote data to datagrid component, assign service data as an instance of [SfDataManager](#) to the [DataSource](#) property or by using [SfDataManager](#) component. To interact with remote data source, provide the endpoint **Url**.

When using [SfDataManager](#) for data binding then the **TValue** must be provided explicitly in the datagrid component.

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

Binding with OData services

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

ASPX-CS

```
@using Syncfusion.Blazor
@using Syncfusion.Blazor.Data
@using Syncfusion.Blazor.Grids
<SfGrid TValue="Order" AllowPaging="true">
  <SfDataManager
    Url="https://js.syncfusion.com/ejServices/Wcf/Northwind.svc/Orders"
    Adaptor="Adaptors.ODataAdaptor"></SfDataManager>
  <GridColumn>
    <GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
      IsPrimaryKey="true" TextAlign="TextAlign.Right" Width="120"></GridColumn>
    <GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
      Width="150"></GridColumn>
```

```

<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.ShipCountry) HeaderText="Ship Country"
Width="150"></GridColumn>
</GridColumn>
</SfGrid>
@code{
public class Order {
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
public string ShipCountry { get; set; }
}
}

```

Binding with OData v4 services

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

ASPX-CS

```

@using Syncfusion.Blazor
@using Syncfusion.Blazor.Data
@using Syncfusion.Blazor.Grids
<SfGrid TValue="Order" AllowPaging="true">
<SfDataManager
Url="https://services.odata.org/V4/Northwind/Northwind.svc/Orders/"
Adaptor="Adaptors.ODataV4Adaptor"></SfDataManager>
<GridColumn>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
IsPrimaryKey="true" TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumn>
</SfGrid>
@code{
public class Order {
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
}

```

Web API

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

ASPX-CS

```
@using Syncfusion.Blazor
@using Syncfusion.Blazor.Data
@using Syncfusion.Blazor.Grids
<SfGrid TValue="Order" AllowPaging="true">
  <SfDataManager Url="https://ej2services.syncfusion.com/production/web-
  services/api/Orders" Adaptor="Adaptors.WebApiAdaptor"></SfDataManager>
  <GridColumn>
    <GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
    IsPrimaryKey="true" TextAlign="TextAlign.Right" Width="120"></GridColumn>
    <GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
    Width="150"></GridColumn>
    <GridColumn Field=@nameof(Order.OrderDate) HeaderText="Order Date"
    Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
    Width="130"></GridColumn>
    <GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
    TextAlign="TextAlign.Right" Width="120"></GridColumn>
  </GridColumn>
</SfGrid>
@code{
public class Order {
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
}
```

The response object from the Web API should contain properties, **Items** and **Count**, whose values are a collection of entities and total count of the entities, respectively.

The sample response object should look like this:

JAVASCRIPT

```
{
  Items: [{...}, {...}, {...}, ...],
  Count: 830
}
```

Enable SfDataManager after initial rendering

It is possible to render the datasource in DataGrid after initial rendering. This can be achieved by conditionally enabling the [SfDataManager](#) component after datagrid rendering.

The following sample code demonstrates enabling data manager condition in the DataGrid on button click,

ASPX-CS

```
@using Syncfusion.Blazor
@using Syncfusion.Blazor.Buttons
@using Syncfusion.Blazor.Data
```



```

@using Syncfusion.Blazor.Grids
<SfButton OnClick="Enable" CssClass="e-primary" IsPrimary="true"
Content="Enable data manager"></SfButton>
<SfGrid TValue="Order" AllowPaging="true">
<GridPageSettings PageSize="10"></GridPageSettings>
@if(IsInitialRender)
{
<SfDataManager Url="https://ej2services.syncfusion.com/production/web-
services/api/Orders" Adaptor="Adaptors.WebApiAdaptor"></SfDataManager>
}
<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
IsPrimaryKey="true" TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumns>
</SfGrid>
@code{
public bool IsInitialRender = false;
public class Order
{
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
public void Enable()
{
// Enabling condition to render the data manager
this.IsInitialRender = true;
}
}

```

The following GIF represents dynamically rendering data manager in DataGrid,

Enable data manager

Order ID	Customer Name	Order Date	Freight
No records to display			
<div> <div> <div></div> <div></div> <div></div> <div></div> </div> </div>			0 of 0 pages (0 items)

Sending additional parameters to the server

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

The following sample code demonstrates sending additional parameters using the `Query` property,

ASPX-CS

```
@using Syncfusion.Blazor
@using Syncfusion.Blazor.Data
@using Syncfusion.Blazor.Grids
<SfGrid TValue="Order" AllowPaging="true" Query=@GridQuery>
<GridPageSettings PageSize="10"></GridPageSettings>
<SfDataManager Url="https://ej2services.syncfusion.com/production/web-services/api/Orders" Adaptor="Adaptors.WebApiAdaptor"></SfDataManager>
<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
IsPrimaryKey="true" TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumns>
</SfGrid>
@code{
public string ParamValue = "true";
```

```

public Query GridQuery { get; set; }
protected override void OnInitialized() {
    GridQuery = new Query().AddParams("ej2grid", ParamValue);
}
public class Order
{
    public int? OrderID { get; set; }
    public string CustomerID { get; set; }
    public DateTime? OrderDate { get; set; }
    public double? Freight { get; set; }
}
}

```

Configuring HttpClient

SfDataManager uses the **HttpClient** instance to make HTTP requests to data services. **SfDataManager** checks whether a **HttpClient** is already registered in the service container if it's found then the **SfDataManager** will use **HttpClient** from the service container else it will create and add **HttpClient** to the service container and use that instance for making requests to the server.

When registering your **HttpClient**, the registration should be done before calling **AddSyncfusionBlazor()** method in **Startup.cs/Program.cs**, so that **SfDataManager** will not create its own **HttpClient** and uses the pre-configured **HttpClient**. This helps **SfDataManager** to use **HttpClient** instance pre-configured with base address, authentication, default headers, etc.

You could also pass **HttpClient** to the **SfDataManager** component as a parameter using **HttpClientInstance** property. This will be useful when the application has more than one pre-configured **HttpClient**s. You can use this approach to use the named **HttpClient** with **SfDataManager**.

To troubleshoot the requests and responses made using **HttpClient**, a custom HTTP message handler can be used. More information about registering the custom HTTP message handler can be found [here](#).

Using Typed **HttpClient** with **SfDataManager** is not supported. The [custom binding](#) feature has to be used to achieve this requirement.

Handling HTTP error

During server interaction from the datagrid, sometimes server-side exceptions might occur. These error messages or exception details can be acquired in client-side using the [OnActionFailure](#) event.

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

The following sample code demonstrates notifying user when server-side exception has occurred,

ASPX-CS

```

@using Syncfusion.Blazor
@using Syncfusion.Blazor.Data
@using Syncfusion.Blazor.Grids
<span class="error">@ErrorDetails</span>
<SfGrid TValue="Order" AllowPaging="true">
    <GridEvents TValue="Order" OnActionFailure="@ActionFailure"></GridEvents>
    <GridPageSettings PageSize="10"></GridPageSettings>
    <SfDataManager Url="https://some.com/invalidUrl"
        Adaptor="Adaptors.WebApiAdaptor"></SfDataManager>
    <GridColumns>
        <GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
            IsPrimaryKey="true" TextAlign="TextAlign.Right" Width="120"></GridColumn>
    </GridColumns>
</SfGrid>

```

```

<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumn>
</SfGrid>
<style>
.error {
color: red;
}
</style>
@code{
public string ErrorDetails = "";
public class Order
{
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
public void ActionFailure(FailureEventArgs args)
{
this.ErrorDetails = "Server exception: 404 Not found";
StateHasChanged();
}
}

```

Authorization and Authentication

It is common to have authorization in the server of origin to prevent anonymous access to the data services. **SfDataManager** can consume data from such protected remote data services with the proper bearer token. The access token or bearer token can be used by **SfDataManager** in one of the following ways.

- By using the pre-configured HttpClient with the access token or authentication message handler, SfDataManager can access protected remote services. When registering your HttpClient, the registration should be done before calling `AddSyncfusionBlazor()` method in **Startup.cs/Program.cs**, so that SfDataManager will not create its own HttpClient and uses the already configured HttpClient.
- Setting access token in the default header of the HttpClient by injecting it in the page. See here for adding default headers to HttpClient.

CSHARP

```

@Inject HttpClient _httpClient
. . . .
@code {
. . . .
protected override async Task OnInitializedAsync()
{
. . . .
}

```

```

_httpClient.DefaultRequestHeaders.Add("Authorization", $"Bearer
{tokenValue}");
await base.OnInitializedAsync();
}
}

```

- Setting the access token in the **Headers** property of the **SfDataManager**. See [here](#) for adding headers.

Getting the bearer token may vary with access token providers. More information on configuring HttpClient with authentication can be found on the official page [here](#).

Setting custom headers

To add a custom headers to the data request, use the [Headers](#) property of the [SfDataManager](#).

The following sample code demonstrates adding custom headers to the **SfDataManager** request,

ASPX-CS

```

@using Syncfusion.Blazor
@using Syncfusion.Blazor.Data
@using Syncfusion.Blazor.Grids
<SfGrid TValue="Order" AllowPaging="true">
  <GridPageSettings PageSize="10"></GridPageSettings>
  <SfDataManager Headers=@HeaderData
    Url="https://ej2services.syncfusion.com/production/web-services/api/Orders"
    Adaptor="Adaptors.WebApiAdaptor"></SfDataManager>
  <GridColumn>
    <GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
      IsPrimaryKey="true" TextAlign="TextAlign.Right" Width="120"></GridColumn>
    <GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
      Width="150"></GridColumn>
    <GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
      Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
      Width="130"></GridColumn>
    <GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
      TextAlign="TextAlign.Right" Width="120"></GridColumn>
  </GridColumn>
</SfGrid>
@code{
  private IDictionary<string, string> HeaderData = new Dictionary<string,
  string>();
  public class Order
  {
    public int? OrderID { get; set; }
    public string CustomerID { get; set; }
    public DateTime? OrderDate { get; set; }
    public double? Freight { get; set; }
  }
}

```

Change Query parameter value dynamically

It is possible to dynamically modify datagrid's [Query](#) property value.

The following sample code demonstrates achieving this,

ASPX-CS

```
@using Syncfusion.Blazor
@using Syncfusion.Blazor.Buttons
@using Syncfusion.Blazor.Data
@using Syncfusion.Blazor.Grids
<SfButton Content="Modify query data" OnClick="BtnClick"></SfButton>
<SfGrid TValue="Order" @ref="GridObj" AllowPaging="true" Query="@QueryData">
  <GridPageSettings PageSize="10"></GridPageSettings>
  <SfDataManager
    Url="https://services.odata.org/V4/Northwind/Northwind.svc/Orders"
    Adaptor="Adaptors.ODataV4Adaptor">
  </SfDataManager>
  <GridColumns>
    <GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
      IsPrimaryKey="true" TextAlign="TextAlign.Right" Width="120"></GridColumn>
    <GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
      Width="150"></GridColumn>
    <GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
      TextAlign="TextAlign.Right" Width="120"></GridColumn>
  </GridColumns>
</SfGrid>
@code{
public SfGrid<Order> GridObj;
private Query QueryData = new Query().Where("CustomerID", "equal", "VINET");
private Query UpdatedQueryData = new Query().Where("CustomerID", "equal",
"HANAR");
public class Order
{
  public int? OrderID { get; set; }
  public string CustomerID { get; set; }
  public double? Freight { get; set; }
}
public void BtnClick()
{
  QueryData = UpdatedQueryData;
}
```

The following GIF represents dynamically modifying the query property in DataGrid,

Modify query data		
Order ID	Customer Name	Freight
10248	VINET	\$32.38
10274	VINET	\$6.01
10295	VINET	\$1.15
10737	VINET	\$7.79
10739	VINET	\$11.08
K < 1 > X		1 of 1 pages (5 items)

SQL Server data binding(SQL Client)

The following examples demonstrate, how to consume data from SQL Server using Microsoft SqlClient and bound it to Blazor DataGrid. You can achieve this requirement by using [Custom Adaptor](#).

Before the implementation, add required NuGet like **Microsoft.Data.SqlClient** and **Syncfusion.Blazor** in your application. In the below sample, Custom Adaptor can be created as a Component. In custom adaptor **Read** method you can get grid action details like paging, filtering, sorting information etc., using **DataManagerRequest**.

Based on the DataManagerRequest, you can form SQL query string (to perform paging) and execute the SQL query and retrieve the data from database using **SqlDataAdapter**. The Fill method of the **DataAdapter** is used to populate a **DataSet** with the results of the **SelectCommand** of the DataAdapter, then converted the DataSet into List and return **Result** and **Count** pair object in **Read** method to bind the data to Grid.

ASPX-CS

```
@using Syncfusion.Blazor.Grids;
@using Syncfusion.Blazor.Data;
@using Syncfusion.Blazor;
<SfGrid TValue="Order" AllowPaging="true">
  <SfDataManager Adaptor="Adaptors.CustomAdaptor">
    <CustomAdaptorComponent></CustomAdaptorComponent>
  </SfDataManager>
  <GridColumns>
    <GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
      IsIdentity="true" IsPrimaryKey="true" TextAlign="TextAlign.Right"
      Width="120">
    </GridColumn>
```

```
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>
</GridColumn>
</SfGrid>
@code{
SfGrid<Order> Grid { get; set; }
public static List<Order> Orders { get; set; }
public class Order
{
public int? OrderID { get; set; }
public string CustomerID { get; set; }
}
}
```

ASPX-CS

```
@using Syncfusion.Blazor;
@using Syncfusion.Blazor.Data;
@using Newtonsoft.Json
@using static EFGrid.Pages.Index;
@using Microsoft.Data.SqlClient;
@using System.Data;
@using System.IO;
@using Microsoft.AspNetCore.Hosting;
@inject IHostingEnvironment _env
@inherits DataAdaptor<Order>
<CascadingValue Value="@this">
@ChildContent
</CascadingValue>
@code {
[Parameter]
[JsonIgnore]
public RenderFragment ChildContent { get; set; }
public static DataSet CreateCommand(string queryString, string
connectionString)
{
using (SqlConnection connection = new SqlConnection(
connectionString))
{
SqlDataAdapter adapter = new SqlDataAdapter(queryString, connection);
DataSet dt = new DataSet();
try
{
connection.Open();
adapter.Fill(dt); // using sqlDataAdapter we process the query string and
fill the data into dataset
}
catch (SqlException se)
{
Console.WriteLine(se.ToString());
}
finally
{
connection.Close();
}
return dt;
}
```



```

}
}
// Performs data Read operation
public override object Read(DataManagerRequest dm, string key = null)
{
    string appdata = _env.ContentRootPath;
    string path = Path.Combine(appdata, "App_Data\\NORTHWND.MDF");
    string str = $"Data
Source=(LocalDB)\\MSSQLLocalDB;AttachDbFilename='{path}';Integrated
Security=True;Connect Timeout=30";
    // based on the skip and take count from DataManagerRequest here we formed
    SQL query string
    string qs = "SELECT OrderID, CustomerID FROM dbo.Orders ORDER BY OrderID
OFFSET " + dm.Skip + " ROWS FETCH NEXT " + dm.Take + " ROWS ONLY;";
    DataSet data = CreateCommand(qs, str);
    Orders = data.Tables[0].AsEnumerable().Select(r => new Order
    {
        OrderID = r.Field<int>("OrderID"),
        CustomerID = r.Field<string>("CustomerID")
    }).ToList(); // here we convert dataset into list
    IEnumerable<Order> DataSource = Orders;
    SqlConnection conn = new SqlConnection(str);
    conn.Open();
    SqlCommand comm = new SqlCommand("SELECT COUNT(*) FROM dbo.Orders", conn);
    Int32 count = (Int32)comm.ExecuteScalar();
    return dm.RequiresCounts ? new DataResult() { Result = DataSource, Count =
count } : (object)DataSource;
}
}

```

* In this [sample](#), paging action is handled for Blazor grid based on your need you can extend the given logic for other operations.

* For performing data manipulation, you can override available methods such as **Insert**, **Update** and **Remove** of the Custom Adaptor.

Entity Framework

You need to follow the below steps to consume data from the **Entity Framework** in the datagrid component.

Create DbContext class

The first step is to create a DbContext class called **OrderContext** to connect to a Microsoft SQL Server database.

CSHARP

```

using Microsoft.EntityFrameworkCore;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using EFGrid.Shared.Models;
namespace EFGrid.Shared.DataAccess
{
    public class OrderContext : DbContext
    {

```

```
public virtual DbSet<Order> Orders { get; set; }
protected override void OnConfiguring(DbContextOptionsBuilder
optionsBuilder)
{
    if (!optionsBuilder.IsConfigured)
    {
        optionsBuilder.UseSqlServer(@"Data
Source=(LocalDB)\MSSQLLocalDB;AttachDbFilename='F:\blazor\EFGrid - CRUD -
P6\EFGrid.Shared\App_Data\NORTHWND.MDF';Integrated Security=True;Connect
Timeout=30");
    }
}
}
```

Create data access layer to perform data operation

Now you need to create a class named **OrderDataAccessLayer**, which act as data access layer for retrieving the records and also insert, update and delete the records from the database table.

CSHARP

```
using Microsoft.EntityFrameworkCore;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using EFGrid.Shared.Models;
namespace EFGrid.Shared.DataAccess
{
    public class OrderDataAccessLayer
    {
        OrderContext db = new OrderContext();
        public DbSet<Order> GetAllOrders()
        {
            try
            {
                return db.Orders;
            }
            catch
            {
                throw;
            }
        }
    }
}
```

Creating Web API Controller

A Web API Controller has to be created which allows datagrid directly to consume data from the Entity Framework.

CSHARP

```
using System;
using System.Collections;
using System.Collections.Generic;
```

```

using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Primitives;
using EFGGrid.Shared.DataAccess;
using EFGGrid.Shared.Models;
namespace WebApplication1.Server.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class DefaultController : ControllerBase
    {
        OrderDataAccessLayer db = new OrderDataAccessLayer();
        [HttpGet]
        public object Get()
        {
            IQueryable<Order> data = db.GetAllOrders().AsQueryable();
            var count = data.Count();
            var queryString = Request.Query;
            if (queryString.Keys.Contains("$inlinecount"))
            {
                StringValues Skip;
                StringValues Take;
                int skip = (queryString.TryGetValue("$skip", out Skip)) ?
                Convert.ToInt32(Skip[0]) : 0;
                int top = (queryString.TryGetValue("$top", out Take)) ?
                Convert.ToInt32(Take[0]) : data.Count();
                return new { Items = data.Skip(skip).Take(top), Count = count };
            }
            else
            {
                return data;
            }
        }
    }
}

```

Configure datagrid component using Web API adaptor

Now you can configure the datagrid using the '**SfDataManager**' to interact with the created Web API and consume the data appropriately. To interact with web api, you need to use WebApiAdaptor.

ASPX-CS

```

@using Syncfusion.Blazor
@using Syncfusion.Blazor.Grids
<SfGrid TValue="Order" AllowPaging="true" >
    <SfDataManager Url="api/Default"
    Adaptor="Adaptors.WebApiAdaptor"></SfDataManager>
    <GridColumn>
        <GridColumn Field="OrderID" HeaderText="Order ID" IsPrimaryKey="true"
        IsIdentity="true" TextAlign="@Syncfusion.Blazor.Grids.TextAlign.Right"
        Width="90"></GridColumn>
        <GridColumn Field="CustomerID" HeaderText="Customer ID"
        Width="90"></GridColumn>
    </GridColumn>
</SfGrid>

```

```
<GridColumn Field="EmployeeID" HeaderText="Employee ID"
Width="90"></GridColumn>
</GridColumn>
</SfGrid>
@code{
public class Order {
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
}
```

To perform datagrid CRUD operation using Entity Framework. You can refer [here](#).

You can find the fully working sample [here](#).

HTTP client

It is possible to call web api from the blazor WebAssembly(client-side) app. This can be used for sending HTTP requests to fetch data from web api and bind them in the DataGrid's data source. The requests are sent using [HttpClient](#) service.

This can be achieved by initially injecting the `HttpClient` instance in the app.

ASPX-CS

```
@using System.Net.Http
@Inject HttpClient Http
```

After that the data to be fetched is defined in the api controller of the blazor WebAssembly app.

CSHARP

```
[Route("api/[controller]")]
[ApiController]
public class EmployeeController : ControllerBase
{
List<Employee> employee = new List<Employee>();
[HttpGet]
public object Get()
{
employee.Add(new Employee { EmployeeID = 1, FirstName = "Andrew", LastName =
"Fuller", Title = "Branch Manager" });
employee.Add(new Employee { EmployeeID = 2, FirstName = "Nancy", LastName =
"Leverling", Title = "Product Manager" });
employee.Add(new Employee { EmployeeID = 3, FirstName = "Anne", LastName =
"Dodsworth", Title = "Team Lead" });
employee.Add(new Employee { EmployeeID = 4, FirstName = "Laura", LastName =
"Callahan", Title = "Product Manager" });
employee.Add(new Employee { EmployeeID = 5, FirstName = "Michael", LastName =
"Suyama", Title = "Team Lead" });
employee.Add(new Employee { EmployeeID = 6, FirstName = "Robert", LastName =
"King", Title = "Developer" });
employee.Add(new Employee { EmployeeID = 7, FirstName = "Janet", LastName =
"Peacock", Title = "Developer" });
}
```

```

employee.Add(new Employee { EmployeeID = 8, FirstName = "Steven", LastName =
"Buchanan", Title = "Developer" });
employee.Add(new Employee { EmployeeID = 9, FirstName = "Margaret", LastName
= "Davolio", Title = "Developer" });
return employee;
}
}
public class Employee
{
public int? EmployeeID { get; set; }
public string FirstName { get; set; }
public string LastName { get; set; }
public string Title { get; set; }
}

```

Then using the `GetJsonAsync` method request is sent to the api controller for fetching data which is bounded to the DataGrid's data source

ASPX-CS

```

@using Syncfusion.Blazor.Grids
@inject HttpClient Http
<SfGrid DataSource="@Empdata" AllowPaging="true">
<GridPageSettings PageSize="7"></GridPageSettings>
<GridColumns>
<GridColumn Field=@nameof(Employee.EmployeeID) HeaderText="Employee ID"
IsPrimaryKey="true" TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Employee.FirstName) HeaderText="First Name"
Width="120"></GridColumn>
<GridColumn Field=@nameof(Employee.LastName) HeaderText="Last Name"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Employee.Title) HeaderText="Title"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumns>
</SfGrid>
@code{
public List<Employee> Empdata { get; set; }
protected override async Task OnInitializedAsync()
{
Empdata = await Http.GetJsonAsync<List<Employee>>("api/Employee");
}
public class Employee
{
public int? EmployeeID { get; set; }
public string FirstName { get; set; }
public string LastName { get; set; }
public string Title { get; set; }
}
}

```

The above steps are processed in the blazor WebAssembly app which has the preconfigured `HttpClient` service. For blazor server apps, web api calls are created using [IHttpClientFactory](#). More information on making requests using `IHttpClientFactory` is available in this [link](#).

Observable Collection

This [ObservableCollection](#) (dynamic data collection) provides notifications when items added, removed and moved. The implement [INotifyCollectionChanged](#) notifies when dynamic changes of add, remove, move and clear the collection. The implement [INotifyPropertyChanged](#) notifies when property value has changed in client side.

Here, Order class implements the interface of **INotifyPropertyChanged** and it raises the event when CustomerID property value was changed.

CSHARP

```
@using Syncfusion.Blazor.Grids
@using System.Collections.ObjectModel
@using System.ComponentModel
<button id="add" @onclick="AddRecords">Add Data</button>
<button id="del" @onclick="DelRecords">Delete Data</button>
<button id="update" @onclick="UpdateRecords">Update Data</button>
<SfGrid DataSource="@GridData" AllowReordering="true">
<GridColumns>
<GridColumn Field=@nameof(OrdersDetailsObserveData.OrderID)
HeaderText="Order ID" IsPrimaryKey="true" TextAlign="@TextAlign.Center"
HeaderTextAlign="@TextAlign.Center" Width="140"></GridColumn>
<GridColumn Field=@nameof(OrdersDetailsObserveData.CustomerID)
HeaderText="Customer Name" Width="150"></GridColumn>
<GridColumn Field=@nameof(OrdersDetailsObserveData.Freight)
HeaderText="Freight" EditType="EditType.NumericEdit" Format="C2" Width="140"
TextAlign="@TextAlign.Right"
HeaderTextAlign="@TextAlign.Right"></GridColumn>
<GridColumn Field=@nameof(OrdersDetailsObserveData.OrderDate)
HeaderText="Order Date" EditType="EditType.DatePickerEdit" Format="d"
Type="ColumnType.Date" Width="160"></GridColumn>
<GridColumn Field=@nameof(OrdersDetailsObserveData.ShipCountry)
HeaderText="Ship Country" EditType="EditType.DropDownEdit"
Width="150"></GridColumn>
</GridColumns>
</SfGrid>
@code{
public ObservableCollection<OrdersDetailsObserveData> GridData { get; set; }
public int Count = 32341;
protected override void OnInitialized()
{
GridData = OrdersDetailsObserveData.GetRecords();
}
public void AddRecords()
{
GridData.Add(new OrdersDetailsObserveData(Count++, "ALFKI", 4343, 2.3 * 43,
false, new DateTime(1991, 05, 15), "Berlin", "Simons bistro", "Denmark", new
DateTime(1996, 7, 16), "Kirchgasse 6"));
}
public void DelRecords()
{
GridData.Remove(GridData.First());
}
public void UpdateRecords()
{
var a = GridData.First();
```

```

a.CustomerID = "Update";
}
public class OrdersDetailsObserveData : INotifyPropertyChanged
{
    public OrdersDetailsObserveData()
    {
    }
    public OrdersDetailsObserveData(int OrderID, string CustomerId, int
EmployeeId, double Freight, bool Verified, DateTime OrderDate, string
ShipCity, string ShipName, string ShipCountry, DateTime ShippedDate, string
ShipAddress)
    {
        this.OrderID = OrderID;
        this.CustomerID = CustomerId;
        this.EmployeeID = EmployeeId;
        this.Freight = Freight;
        this.ShipCity = ShipCity;
        this.Verified = Verified;
        this.OrderDate = OrderDate;
        this.ShipName = ShipName;
        this.ShipCountry = ShipCountry;
        this.ShippedDate = ShippedDate;
        this.ShipAddress = ShipAddress;
    }
    public static ObservableCollection<OrdersDetailsObserveData> GetRecords()
    {
        ObservableCollection<OrdersDetailsObserveData> order = new
ObservableCollection<OrdersDetailsObserveData>();
        int code = 10000;
        for (int i = 1; i < 2; i++)
        {
            order.Add(new OrdersDetailsObserveData(code + 1, "ALFKI", i + 0, 2.3 * i,
false, new DateTime(1991, 05, 15), "Berlin", "Simons bistro", "Denmark", new
DateTime(1996, 7, 16), "Kirchgasse 6"));
            order.Add(new OrdersDetailsObserveData(code + 2, "ANATR", i + 2, 3.3 * i,
true, new DateTime(1990, 04, 04), "Madrid", "Queen Cozinha", "Brazil", new
DateTime(1996, 9, 11), "Avda. Azteca 123"));
            order.Add(new OrdersDetailsObserveData(code + 3, "ANTON", i + 1, 4.3 * i,
true, new DateTime(1957, 11, 30), "Cholchester", "Frankenversand",
"Germany", new DateTime(1996, 10, 7), "Carrera 52 con Ave. Bolívar #65-98
Llano Largo"));
            order.Add(new OrdersDetailsObserveData(code + 4, "BLONP", i + 3, 5.3 * i,
false, new DateTime(1930, 10, 22), "Marseille", "Ernst Handel", "Austria",
new DateTime(1996, 12, 30), "Magazinweg 7"));
            order.Add(new OrdersDetailsObserveData(code + 5, "BOLID", i + 4, 6.3 * i,
true, new DateTime(1953, 02, 18), "Tsawassen", "Hanari Carnes",
"Switzerland", new DateTime(1997, 12, 3), "1029 - 12th Ave. S.));
            code += 5;
        }
        return order;
    }
    public int OrderID { get; set; }
    public string CustomerID
    {
        get { return customerID; }
        set

```

```

{
    customerID = value;
    NotifyPropertyChanged("CustomerID");
}
}

public string customerID { get; set; }
public int? EmployeeID { get; set; }
public double? Freight { get; set; }
public string ShipCity { get; set; }
public bool Verified { get; set; }
public DateTime? OrderDate { get; set; }
public event PropertyChangedEventHandler PropertyChanged;
public string ShipName { get; set; }
public string ShipCountry { get; set; }
public DateTime ShippedDate { get; set; }
public string ShipAddress { get; set; }
private void NotifyPropertyChanged(String propertyName)
{
    if (PropertyChanged != null)
    {
        PropertyChanged(this, new PropertyChangedEventArgs(propertyName));
    }
}
}
}

```

The following screenshot represents the DataGrid with **Observable Collection**.

ADD DATA DELETE DATA UPDATE DATA				
Order ID	Customer Name	Freight	Order Date	
10001	ANTON	\$2.10	10/13/2019	
10002	ANTON	\$4.20	10/12/2019	
10003	BLONP	\$6.30	10/11/2019	
10004	BOLID	\$8.40	10/10/2019	
10005	ANTON	\$10.50	10/9/2019	
10006	BOLID	\$12.60	10/8/2019	
10007	BLONP	\$14.70	10/7/2019	
10008	ALFKI	\$16.80	10/6/2019	
10009	ANANTR	\$18.90	10/5/2019	
10010	BLONP	\$21.00	10/4/2019	

1 of 1 pages (10 items)

Troubleshoot: DataGrid renders without data even though server returns with correct data

In ASP.NET Core, by default the JSON results are returned in **camelCase** format. So datagrid field names are also changed in **camelCase**.

To avoid this problem, you need to add [DefaultContractResolver](#) in **Startup.cs** file.

CSHARP

```

public void ConfigureServices(IServiceCollection services)
{
    services.AddRazorPages();
    services.AddServerSideBlazor().AddCircuitOptions();
    services.AddSingleton<WeatherForecastService>();
}

```



```
services.AddMvc().AddNewtonsoftJson(options =>
{
options.SerializerSettings.ContractResolver = new DefaultContractResolver();
});
}
```

Handling exceptions

Exceptions occurred during grid actions can be handled without stopping application. These error messages or exception details can be acquired using the [OnActionFailure](#) event.

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

We recommend you to bind **OnActionFailure** event during your application development phase, this helps you to find any exceptions. You can pass these exception details to our support team to get solution as early as possible.

The following sample code demonstrates notifying user when server-side exception has occurred during data operation,

ASPX-CS

```
@using Syncfusion.Blazor
@using Syncfusion.Blazor.Data
@using Syncfusion.Blazor.Grids
<span class="error">@ErrorDetails</span>
<SfGrid TValue="Order" AllowPaging="true">
<GridEvents TValue="Order" OnActionFailure="@ActionFailure"></GridEvents>
<GridPageSettings PageSize="10"></GridPageSettings>
<SfDataManager Url="https://some.com/invalidUrl"
Adaptor="Adaptors.WebApiAdaptor"></SfDataManager>
<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
IsPrimaryKey="true" TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumns>
</SfGrid>
<style>
.error {
color: red;
}
</style>
@code{
public string ErrorDetails = "";
public class Order
{
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
```

```
public void ActionFailure(FailureEventArgs args)
{
    this.ErrorDetails = "Server exception: 404 Not found";
    StateHasChanged();
}
}
```

You can refer to our [Blazor DataGrid](#) feature tour page for its groundbreaking feature representations. You can also explore our [Blazor DataGrid example](#) to understand how to present and manipulate data.

Custom Binding in Blazor DataGrid Component

The [SfDataManager](#) has custom adaptor support which allows you to perform manual operations on the data. This can be utilized for implementing custom data binding and editing operations in the DataGrid component.

For implementing custom data binding in DataGrid, the **DataAdaptor** class is used. This abstract class acts as a base class for the custom adaptor.

The **DataAdaptor** abstract class has both synchronous and asynchronous method signatures which can be overridden in the custom adaptor. Following are the method signatures present in this class,

CSHARP

```
public abstract class DataAdaptor
{
    /// <summary>
    /// Performs data Read operation synchronously.
    /// </summary>
    public virtual object Read(DataManagerRequest dataManagerRequest, string key
    = null)
    /// <summary>
    /// Performs data Read operation asynchronously.
    /// </summary>
    public virtual Task<object> ReadAsync(DataManagerRequest dataManagerRequest,
    string key = null)
    /// <summary>
    /// Performs Insert operation synchronously.
    /// </summary>
    public virtual object Insert(DataManager dataManager, object data, string
    key)
    /// <summary>
    /// Performs Insert operation asynchronously.
    /// </summary>
    public virtual Task<object> InsertAsync(DataManager dataManager, object
    data, string key)
    /// <summary>
    /// Performs Remove operation synchronously.
    /// </summary>
    public virtual object Remove(DataManager dataManager, object data, string
    keyField, string key)
    /// <summary>
    /// Performs Remove operation asynchronously.
    /// </summary>
    public virtual Task<object> RemoveAsync(DataManager dataManager, object
    data, string keyField, string key)
    /// <summary>
```

```

/// Performs Update operation synchronously.
/// </summary>
public virtual object Update (DataManager dataManager, object data, string
keyField, string key)
/// <summary>
/// Performs Update operation asynchronously.
/// </summary>
public virtual Task<object> UpdateAsync (DataManager dataManager, object
data, string keyField, string key)
/// <summary>
/// Performs Batch CRUD operations synchronously.
/// </summary>
public virtual object BatchUpdate (DataManager dataManager, object
changedRecords, object addedRecords, object deletedRecords, string keyField,
string key, int? dropIndex)
/// <summary>
/// Performs Batch CRUD operations asynchronously.
/// </summary>
public virtual Task<object> BatchUpdateAsync (DataManager dataManager, object
changedRecords, object addedRecords, object deletedRecords, string keyField,
string key, int? dropIndex)
}

```

Data binding

The custom data binding can be performed in the DataGrid component by providing the custom adaptor class and overriding the **Read** or **ReadAsync** method of the **DataAdaptor** abstract class.

The following sample code demonstrates implementing custom data binding using custom adaptor,

ASPX-CS

```

@using Syncfusion.Blazor
@using Syncfusion.Blazor.Data
@using Syncfusion.Blazor.Grids
<SfGrid TValue="Order" ID="Grid" AllowSorting="true" AllowFiltering="true"
AllowPaging="true">
<SfDataManager AdaptorInstance="@typeof(CustomAdaptor)"
Adaptor="Adaptors.CustomAdaptor"></SfDataManager>
<GridPageSettings PageSize="8"></GridPageSettings>
<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
IsPrimaryKey="true" TextAlign="@TextAlign.Center" Width="140"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
Width="150"></GridColumn>
</GridColumns>
</SfGrid>
@code{
public static List<Order> Orders { get; set; }
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,

```

```
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
}).ToList();
}
public class Order
{
public int OrderID { get; set; }
public string CustomerID { get; set; }
public double Freight { get; set; }
}
// Implementing custom adaptor by extending the DataAdaptor class
public class CustomAdaptor : DataAdaptor
{
// Performs data Read operation
public override object Read(DataManagerRequest dm, string key = null)
{
IEnumerable<Order> DataSource = Orders;
if (dm.Search != null && dm.Search.Count > 0)
{
// Searching
DataSource = DataOperations.PerformSearching(DataSource, dm.Search);
}
if (dm.Sorted != null && dm.Sorted.Count > 0)
{
// Sorting
DataSource = DataOperations.PerformSorting(DataSource, dm.Sorted);
}
if (dm.Where != null && dm.Where.Count > 0)
{
// Filtering
DataSource = DataOperations.PerformFiltering(DataSource, dm.Where,
dm.Where[0].Operator);
}
int count = DataSource.Cast<Order>().Count();
if (dm.Skip != 0)
{
//Paging
DataSource = DataOperations.PerformSkip(DataSource, dm.Skip);
}
if (dm.Take != 0)
{
DataSource = DataOperations.PerformTake(DataSource, dm.Take);
}
DataResult DataObject = new DataResult();
if (dm.Aggregates != null) // Aggregation
{
DataObject.Result = DataSource;
DataObject.Count = count;
DataObject.Aggregates = DataUtil.PerformAggregation(DataSource,
dm.Aggregates);
return dm.RequiresCounts ? DataObject : (object)DataSource;
}
return dm.RequiresCounts ? new DataResult() { Result = DataSource, Count =
count } : (object)DataSource;
}
}
```

```
}

```

If the **DataManagerRequest.RequiresCounts** value is **true**, then the Read/ReadAsync return value must be of **DataRow** with properties **Result** whose value is a collection of records and **Count** whose value is the total number of records. If the **DataManagerRequest.RequiresCounts** is **false**, then simply send the collection of records.

The following image shows the custom bound data displayed in the DataGrid component,

Order ID	Customer Name	Freight
1001	ANTON	2.1
1002	BLONP	4.2
1003	ANTON	6.3000000000000001
1004	BOLID	8.4
1005	ANTON	10.5
1006	ANTON	12.6000000000000001
1007	ANANTR	14.7000000000000001
1008	ALFKI	16.8
K < 1 2 3 4 5 6 7 8 9 10 > X		1 of 10 pages (75 items)

If the Read/ReadAsync method is not overridden in the custom adaptor, then it will be handled by the default read handler.

Inject service into Custom Adaptor

If you want to inject some of your service into Custom Adaptor and use the service, then you can achieve your requirement by using below way.

Initially, you need to add CustomAdaptor class as AddScoped in **Startup.cs** file.

CSHARP

```
public void ConfigureServices(IServiceCollection services)
{
    ...
    services.AddSingleton<OrderDataAccessLayer>();
    services.AddScoped<CustomAdaptor>();
    services.AddScoped<ServiceClass>();
}
```

The following sample code demonstrates injecting service into Custom Adaptor,

ASPX-CS

```

@using Syncfusion.Blazor.Data
@using Syncfusion.Blazor.Grids
@using Syncfusion.Blazor
<SfGrid TValue="Order" ID="Grid" AllowSorting="true" AllowFiltering="true"
AllowPaging="true">
<SfDataManager AdaptorInstance="@typeof(CustomAdaptor)"
Adaptor="Adaptors.CustomAdaptor"></SfDataManager>
<GridPageSettings PageSize="8"></GridPageSettings>
<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
IsPrimaryKey="true" TextAlign="@TextAlign.Center" Width="140"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight"
Width="150"></GridColumn>
</GridColumns>
</SfGrid>
@code{
public class CustomAdaptor : DataAdaptor
{
//Here, you can inject your service
public OrderDataAccessLayer context { get; set; };
public CustomAdaptor(OrderDataAccessLayer _context)
{
context = _context;
}
// Performs data Read operation
public override object Read(DataManagerRequest dm, string key = null)
{
IEnumerable<Order> DataSource = context.GetAllOrders();
if (dm.Search != null && dm.Search.Count > 0)
{
// Searching
DataSource = DataOperations.PerformSearching(DataSource, dm.Search);
}
if (dm.Sorted != null && dm.Sorted.Count > 0)
{
// Sorting
DataSource = DataOperations.PerformSorting(DataSource, dm.Sorted);
}
if (dm.Where != null && dm.Where.Count > 0)
{
// Filtering
DataSource = DataOperations.PerformFiltering(DataSource, dm.Where,
dm.Where[0].Operator);
}
int count = DataSource.Cast<Order>().Count();
if (dm.Skip != 0)
{
//Paging
DataSource = DataOperations.PerformSkip(DataSource, dm.Skip);
}
if (dm.Take != 0)
{
DataSource = DataOperations.PerformTake(DataSource, dm.Take);
}
}
}

```

```

}
return dm.RequiresCounts ? new DataResult() { Result = DataSource, Count =
count } : (object)DataSource;
}
}
}

```

Custom adaptor as Component

Custom Adaptor can be created as a component when `DataAdaptor` is extended from `OwningComponentBase`. You can create Custom Adaptor from any of the two versions of the class, `DataAdaptor` and `DataAdaptor<T>`.

Ensure to register your service in **Startup.cs** file.

CSHARP

```

public void ConfigureServices(IServiceCollection services)
{
    ...
    services.AddScoped<Order>();
}

```

The following sample code demonstrates creating Custom Adaptor as a component,

ASPX-CS

```

@using Syncfusion.Blazor
@using Syncfusion.Blazor.Data
@using Syncfusion.Blazor.Grids
<SfGrid TValue="Order" ID="Grid" AllowSorting="true" AllowFiltering="true"
AllowPaging="true">
<SfDataManager Adaptor="Adaptors.CustomAdaptor">
<CustomAdaptorComponent></CustomAdaptorComponent>
</SfDataManager>
<GridPageSettings PageSize="8"></GridPageSettings>
<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
IsPrimaryKey="true" TextAlign="@TextAlign.Center" Width="140"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight"
TextAlign="@TextAlign.Center" Width="140"></GridColumn>
</GridColumns>
</SfGrid>
@code{
public static List<Order> Orders { get; set; }
public class Order
{
public List<Order> GetAllRecords()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],

```

```
Freight = 2.1 * x,
}).ToList();
return Orders;
}
public int OrderID { get; set; }
public string CustomerID { get; set; }
public double Freight { get; set; }
}
}
```

The following sample code demonstrates `DataAdaptor` extended from `OwningComponentBase<T>`. This provides a single service of type `T`. We can access this service by using the `Service` property.

CSHARP

```
[CustomAdaptorComponent.razor]
@using Syncfusion.Blazor;
@using Syncfusion.Blazor.Data;
@using Newtonsoft.Json
@using static BlazorApp1.Pages.Index;
@inherits DataAdaptor<Order>
<CascadingValue Value="@this">
@ChildContent
</CascadingValue>
@code {
    [Parameter]
    [JsonIgnore]
    public RenderFragment ChildContent { get; set; }
    // Performs data Read operation
    public override object Read(DataManagerRequest dm, string key = null)
    {
        IEnumerable<Order> DataSource = (IEnumerable<Order>)Service.GetAllRecords();
        if (dm.Search != null && dm.Search.Count > 0)
        {
            // Searching
            DataSource = DataOperations.PerformSearching(DataSource, dm.Search);
        }
        if (dm.Sorted != null && dm.Sorted.Count > 0)
        {
            // Sorting
            DataSource = DataOperations.PerformSorting(DataSource, dm.Sorted);
        }
        if (dm.Where != null && dm.Where.Count > 0)
        {
            // Filtering
            DataSource = DataOperations.PerformFiltering(DataSource, dm.Where,
            dm.Where[0].Operator);
        }
        int count = DataSource.Cast<Order>().Count();
        if (dm.Skip != 0)
        {
            //Paging
            DataSource = DataOperations.PerformSkip(DataSource, dm.Skip);
        }
        if (dm.Take != 0)
        {

```



```

DataSource = DataOperations.PerformTake(DataSource, dm.Take);
}
return dm.RequiresCounts ? new DataResult() { Result = DataSource, Count =
count } : (object)DataSource;
}
}

```

The following sample code demonstrates `DataAdaptor` extended from `OwningComponentBase`. This provides the possibility to request multiple services.

CSHARP

```

[CustomAdaptorComponent.razor]
@using Syncfusion.Blazor;
@using Syncfusion.Blazor.Data;
@using static BlazorApp1.Pages.Index
@using Newtonsoft.Json
@inherits DataAdaptor
<CascadingValue Value="@this">
@ChildContent
</CascadingValue>
@code {
[Parameter]
[JsonIgnore]
public RenderFragment ChildContent { get; set; }
Order orderdata;
public override object Read(DataManagerRequest dm, string key = null)
{
orderdata = (Order)ScopedServices.GetService(typeof(Order));
IEnumerable<Order> DataSource =
(IEnumerable<Order>)orderdata.GetAllRecords().Take(10);
if (dm.Search != null && dm.Search.Count > 0)
{
// Searching
DataSource = DataOperations.PerformSearching(DataSource, dm.Search);
}
if (dm.Sorted != null && dm.Sorted.Count > 0)
{
// Sorting
DataSource = DataOperations.PerformSorting(DataSource, dm.Sorted);
}
if (dm.Where != null && dm.Where.Count > 0)
{
// Filtering
DataSource = DataOperations.PerformFiltering(DataSource, dm.Where,
dm.Where[0].Operator);
}
int count = DataSource.Cast<Order>().Count();
if (dm.Skip != 0)
{
//Paging
DataSource = DataOperations.PerformSkip(DataSource, dm.Skip);
}
if (dm.Take != 0)
{
DataSource = DataOperations.PerformTake(DataSource, dm.Take);
}
}
}

```

```

}
return dm.RequiresCounts ? new DataResult() { Result = DataSource, Count =
count } : (object)DataSource;
}
}

```

CRUD operation

The CRUD operations for the custom bound data in the DataGrid component can be implemented by overriding the following CRUD methods of the **DataAdaptor** abstract class,

- **Insert/InsertAsync**
- **Remove/RemoveAsync**
- **Update/UpdateAsync**
- **BatchUpdate/BatchUpdateAsync**

While using batch editing in DataGrid, use BatchUpdate/BatchUpdateAsync method to handle the corresponding CRUD operation.

The following sample code demonstrates implementing CRUD operations for the custom bound data,

ASPX-CS

```

@using Syncfusion.Blazor
@using Syncfusion.Blazor.Data
@using Syncfusion.Blazor.Grids
<SfGrid TValue="Order" ID="Grid" AllowSorting="true" AllowFiltering="true"
AllowPaging="true" Toolbar="@ (new List<string>() { "Add", "Delete",
"Update", "Cancel" })">
<SfDataManager AdaptorInstance="@typeof(CustomAdaptor)"
Adaptor="Adaptors.CustomAdaptor"></SfDataManager>
<GridPageSettings PageSize="8"></GridPageSettings>
<GridEditSettings AllowEditing="true" AllowDeleting="true"
AllowAdding="true" Mode="@EditMode.Normal"></GridEditSettings>
<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
IsPrimaryKey="true" TextAlign="@TextAlign.Center" Width="140"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight"
Width="150"></GridColumn>
</GridColumns>
</SfGrid>
@code{
public static List<Order> Orders { get; set; }
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
}).ToList();
}
}
public class Order

```

```
{
public int OrderID { get; set; }
public string CustomerID { get; set; }
public double Freight { get; set; }
}
// Implementing custom adaptor by extending the DataAdaptor class
public class CustomAdaptor : DataAdaptor
{
// Performs data Read operation
public override object Read(DataManagerRequest dm, string key = null)
{
IEnumerable<Order> DataSource = Orders;
if (dm.Search != null && dm.Search.Count > 0)
{
// Searching
DataSource = DataOperations.PerformSearching(DataSource, dm.Search);
}
if (dm.Sorted != null && dm.Sorted.Count > 0)
{
// Sorting
DataSource = DataOperations.PerformSorting(DataSource, dm.Sorted);
}
if (dm.Where != null && dm.Where.Count > 0)
{
// Filtering
DataSource = DataOperations.PerformFiltering(DataSource, dm.Where,
dm.Where[0].Operator);
}
int count = DataSource.Cast<Order>().Count();
if (dm.Skip != 0)
{
//Paging
DataSource = DataOperations.PerformSkip(DataSource, dm.Skip);
}
if (dm.Take != 0)
{
DataSource = DataOperations.PerformTake(DataSource, dm.Take);
}
return dm.RequiresCounts ? new DataResult() { Result = DataSource, Count =
count } : (object)DataSource;
}
// Performs Insert operation
public override object Insert(DataManager dm, object value, string key)
{
Orders.Insert(0, value as Order);
return value;
}
// Performs Remove operation
public override object Remove(DataManager dm, object value, string keyField,
string key)
{
Orders.Remove(Orders.Where(or => or.OrderID ==
int.Parse(value.ToString()))).FirstOrDefault());
return value;
}
// Performs Update operation
```

```
public override object Update(DataManager dm, object value, string keyField,
string key)
{
    var data = Orders.Where(or => or.OrderID == (value as
Order).OrderID).FirstOrDefault();
    if (data != null)
    {
        data.OrderID = (value as Order).OrderID;
        data.CustomerID = (value as Order).CustomerID;
        data.Freight = (value as Order).Freight;
    }
    return value;
}
// Performs BatchUpdate operation
public override object BatchUpdate(DataManager dm, object Changed, object
Added, object Deleted, string KeyField, string Key, int? dropIndex)
{
    if (Changed != null)
    {
        foreach (var rec in (IEnumerable<Order>)Changed)
        {
            Order val = Orders.Where(or => or.OrderID == rec.OrderID).FirstOrDefault();
            val.OrderID = rec.OrderID;
            val.CustomerID = rec.CustomerID;
            val.Freight = rec.Freight;
        }
    }
    if (Added != null)
    {
        foreach (var rec in (IEnumerable<Order>)Added)
        {
            Orders.Add(rec);
        }
    }
    if (Deleted != null)
    {
        foreach (var rec in (IEnumerable<Order>)Deleted)
        {
            Orders.Remove(Orders.Where(or => or.OrderID ==
rec.OrderID).FirstOrDefault());
        }
    }
    return Orders;
}
}
```

The following GIF displays the CRUD operations performed on the custom bound data displayed in the DataGrid component,

<div> + Add 🗑 Delete 💾 Update ✕ Cancel </div>		
Order ID	Customer Name	Freight
<input type="text"/>	<input type="text"/>	<input type="text"/>
1001	ANTON	2.1
1002	BLONP	4.2
1003	ANTON	6.3000000000000001
1004	BLONP	8.4
1005	ANTON	10.5
1006	BOLID	12.6000000000000001
1007	ALFKI	14.7000000000000001
1008	ALFKI	16.8
<div> K < 1 2 3 4 5 6 7 8 9 10 > ⌂ </div>		1 of 10 pages (75 items)

You can refer to the [Blazor DataGrid](#) feature tour page for its groundbreaking feature representations. You can also explore [Blazor DataGrid example](#) to understand how to present and manipulate data.

Handling Aggregates in Custom Adaptor

When using Custom Adaptor, the aggregates must be handled in the Read/ReadAsync method of Custom adaptor.

The following sample code demonstrates implementing the aggregates for the custom bound data,

ASPX-CS

```
@using Syncfusion.Blazor.Grids
@using Syncfusion.Blazor.Data
@using Syncfusion.Blazor
<SfGrid TValue="Order" AllowPaging="true">
<SfDataManager AdaptorInstance="@typeof(CustomAdaptor) "
Adaptor="Adaptors.CustomAdaptor"></SfDataManager>
<GridPageSettings PageSize="8"></GridPageSettings>
<GridAggregates>
<GridAggregate>
<GridAggregateColumns>
<GridAggregateColumn Field=@nameof(Order.Freight) Type="AggregateType.Sum"
Format="C2">
<FooterTemplate>
@{
var aggregate = (context as AggregateTemplateContext);
<div>
<p>Sum: @aggregate.Sum</p>
</div>
}
```

```

</FooterTemplate>
</GridAggregateColumn>
</GridAggregateColumns>
</GridAggregate>
<GridAggregate>
<GridAggregateColumns>
<GridAggregateColumn Field=@nameof(Order.Freight)
Type="AggregateType.Average" Format="C2">
<FooterTemplate>
@{
var aggregate = (context as AggregateTemplateContext);
<div>
<p>Average: @aggregate.Average</p>
</div>
}
</FooterTemplate>
</GridAggregateColumn>
</GridAggregateColumns>
</GridAggregate>
</GridAggregates>
<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumns>
</SfGrid>
@code{
public static List<Order> Orders { get; set; }
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
OrderDate = DateTime.Now.AddDays(-x),
}).ToList();
}
public class Order
{
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
public class CustomAdaptor : DataAdaptor
{
// Performs data Read operation
public override object Read(DataManagerRequest dm, string key = null)
{

```

```
IEnumerable<Order> DataSource = Orders;
if (dm.Search != null && dm.Search.Count > 0)
{
    // Searching
    DataSource = DataOperations.PerformSearching(DataSource, dm.Search);
}
if (dm.Sorted != null && dm.Sorted.Count > 0)
{
    // Sorting
    DataSource = DataOperations.PerformSorting(DataSource, dm.Sorted);
}
if (dm.Where != null && dm.Where.Count > 0)
{
    // Filtering
    DataSource = DataOperations.PerformFiltering(DataSource, dm.Where,
    dm.Where[0].Operator);
}
int count = DataSource.Cast<Order>().Count();
if (dm.Skip != 0)
{
    //Paging
    DataSource = DataOperations.PerformSkip(DataSource, dm.Skip);
}
if (dm.Take != 0)
{
    DataSource = DataOperations.PerformTake(DataSource, dm.Take);
}
DataResult DataObject = new DataResult();
if (dm.Aggregates != null) // Aggregation
{
    DataObject.Result = DataSource;
    DataObject.Count = count;
    DataObject.Aggregates = DataUtil.PerformAggregation(DataSource,
    dm.Aggregates);
    return dm.RequiresCounts ? DataObject : (object)DataSource;
}
return dm.RequiresCounts ? new DataResult() { Result = DataSource, Count =
count } : (object)DataSource;
}
}
```

Handling Grouping in Custom Adaptor

When using Custom Adaptor, the grouping operation has to be handled in the Read/ReadAsync method of Custom adaptor.

The following sample code demonstrates implementing the grouping operation for the custom bounded data,

ASPX-CS

```
@using Syncfusion.Blazor
@using Syncfusion.Blazor.Data
@using Syncfusion.Blazor.Grids
@using System.Collections
```

```

<SfGrid TValue="Order" ID="Grid" AllowSorting="true" AllowFiltering="true"
AllowPaging="true" AllowGrouping="true">
<SfDataManager AdaptorInstance="@typeof(CustomAdaptor)"
Adaptor="Adaptors.CustomAdaptor"></SfDataManager>
<GridPageSettings PageSize="8"></GridPageSettings>
<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
IsPrimaryKey="true" TextAlign="@TextAlign.Center" Width="140"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
Width="150"></GridColumn>
</GridColumns>
</SfGrid>
@code{
public static List<Order> Orders { get; set; }
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
}).ToList();
}
public class Order
{
public int OrderID { get; set; }
public string CustomerID { get; set; }
public double Freight { get; set; }
}
// Implementing custom adaptor by extending the DataAdaptor class
public class CustomAdaptor : DataAdaptor
{
// Performs data Read operation
public override object Read(DataManagerRequest dm, string key = null)
{
IEnumerable<Order> DataSource = Orders;
if (dm.Search != null && dm.Search.Count > 0)
{
// Searching
DataSource = DataOperations.PerformSearching(DataSource, dm.Search);
}
if (dm.Sorted != null && dm.Sorted.Count > 0)
{
// Sorting
DataSource = DataOperations.PerformSorting(DataSource, dm.Sorted);
}
if (dm.Where != null && dm.Where.Count > 0)
{
// Filtering
DataSource = DataOperations.PerformFiltering(DataSource, dm.Where,
dm.Where[0].Operator);
}
int count = DataSource.Cast<Order>().Count();
if (dm.Skip != 0)

```



```

{
//Paging
DataSource = DataOperations.PerformSkip(DataSource, dm.Skip);
}
if (dm.Take != 0)
{
DataSource = DataOperations.PerformTake(DataSource, dm.Take);
}
DataResult DataObject = new DataResult();
if (dm.Group != null)
{
IEnumerable ResultData = DataSource.ToList();
// Grouping
foreach (var group in dm.Group)
{
ResultData = DataUtil.Group<Order>(ResultData, group, dm.Aggregates, 0,
dm.GroupByFormatter);
}
DataObject.Result = ResultData;
DataObject.Count = count;
return dm.RequiresCounts ? DataObject : (object)ResultData;
}
return dm.RequiresCounts ? new DataResult() { Result = DataSource, Count =
count } : (object)DataSource;
}
}
}

```

Data Annotation in Blazor DataGrid Component

Data Annotations helps us to define rules to the model classes or properties to perform data validation and display suitable messages to end users.

The Data Annotation can be enabled by referencing the **System.ComponentModel.DataAnnotations** namespace which maps the data annotations to the corresponding DataGrid Column property.

The list of data annotation attributes that are supported in DataGrid component are provided below,

Attribute Name	Functionality
DisplayFormat	Sets 'DisplayFormat' property for the DataGrid column to be rendered.
DisplayName	Sets the display name for the DataGrid column.
ReadOnly	Sets AllowEditing for a particular column
Key	Sets PrimaryKey in DataGrid Columns
Validations are, 1. RequiredAttribute 2. StringLengthAttribute 3. RangeAttribute 4. RegularExpressionAttribute 5. MinLengthAttribute 6. MaxLengthAttribute 7. EmailAddressAttribute 8. CompareAttribute 9. DataTypeAttribute 10. DataType.Custom 11. DataType.Date 12. DataType.DateTime 13. DataType.EmailAddress 14. DataType.ImageUrl 15. DataType.Url	

DataGrid Property has more priority than Data Annotation. For Instance, if DisplayName Attribute is set to a Field in DataGrid Model class and also HeaderText is set to the same DataGrid column property, then the value of HeaderText property will be considered and shown in the DataGrid header.

The following sample code demonstrates data annotations implemented in the DataGrid component,

ASPX-CS

```
@using Syncfusion.Blazor.Grids
@using System.ComponentModel.DataAnnotations;
<SfGrid TValue="Order" DataSource="@Orders" Height="315" AllowPaging="true"
Toolbar="@ (new List<string>() { "Add", "Edit", "Delete", "Update", "Cancel"
}) ">
<GridEditSettings AllowAdding="true" AllowEditing="true"
AllowDeleting="true"></GridEditSettings>
<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) TextAlign="TextAlign.Right"
Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate)
EditType="EditType.DatePickerEdit" Format="d" TextAlign="TextAlign.Right"
Width="130" Type="ColumnType.Date"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight"
TextAlign="TextAlign.Right" Format="C2" Width="120"></GridColumn>
</GridColumns>
</SfGrid>
@code{
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
OrderDate = DateTime.Now.AddDays(-x),
}).ToList();
}
public class Order
{
// Sets column as primary key
[Key]
// Sets column as required and error message to be displayed when empty
[Required(ErrorMessage = "Order ID should not be empty")]
// Sets header text to the column
[DisplayName = "Order ID"]
public int? OrderID { get; set; }
[DisplayName = "Customer Name"]
// Sets column as required and error message to be displayed when empty
[Required(ErrorMessage = "Field should not be empty")]
public string CustomerID { get; set; }
// Sets data type of column as Date
[DataType(DataType.Date)]
[DisplayName = "Order Date"]
// Sets column as read only
[Editable(false)]
```

```
public DateTime? OrderDate { get; set; }
[Display(Name = "Freight")]
public double? Freight { get; set; }
}
```

The following image represents data annotations enabled in the DataGrid columns,

+ Add Edit Delete Update Cancel				
Order ID	Customer Name	Order Date		Freight
1001	<input type="text"/>	8/29/2019		2.1
1002	Field should not be empty	8/28/2019		\$4.20
1003		8/27/2019		\$6.30
1004	BLONP	8/26/2019		\$8.40
1005	ANTON	8/25/2019		\$10.50
1006	BLONP	8/24/2019		\$12.60
1007	ANANTR	8/23/2019		\$14.70
1008	ANTON	8/22/2019		\$16.80
1009	ALFKI	8/21/2019		\$18.90
K < 1 2 3 4 5 6 7 > X 1 of 7 pages (75 items)				

You can refer to our [Blazor DataGrid](#) feature tour page for its groundbreaking feature representations. You can also explore our [Blazor DataGrid example](#) to understand how to present and manipulate data.

Columns in Blazor DataGrid Component

The column definitions are used as the **DataSource** schema in the DataGrid. This plays a vital role in rendering column values in the required format. The DataGrid operations such as sorting, filtering and grouping etc. are performed based on column definitions. The [Field](#) property of **GridColumn** is necessary to map the datasource values in DataGrid columns.

1. If the column [Field](#) is not specified in the dataSource, the column values will be empty.
2. If the [Field](#) name contains “dot” operator, it is considered as complex binding.
3. It is must to define the [Field](#) property for a [Template](#) column, to perform CRUD or Data Operations such as filtering, searching etc.

Auto generation

The [Columns](#) are automatically generated when [Columns](#) declaration is empty or undefined while initializing the datagrid. All the columns in the **DataSource** are bound as datagrid columns.

ASPX-CS

```

@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders"></SfGrid>
@code{
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
    Orders = Enumerable.Range(1, 75).Select(x => new Order()
    {
        OrderID = 1000 + x,
        CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
    })[new Random().Next(5)],
        Freight = 2.1 * x,
        OrderDate = DateTime.Now.AddDays(-x),
    }).ToList();
}
public class Order
{
    public int? OrderID { get; set; }
    public string CustomerID { get; set; }
    public DateTime? OrderDate { get; set; }
    public double? Freight { get; set; }
}
}

```

Dynamic column building

It is possible to dynamically build and customize each of the DataGrid column using the type of the model.

You can refer the following code example to achieve this.

ASPX-CS

```

@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@OrderData">
<GridEditSettings AllowEditing="true" AllowAdding="true"
AllowDeleting="true"></GridEditSettings>
<GridColumns>
@foreach (var prop in typeof(Order).GetProperties())
{
    <GridColumn Field="@prop.Name" IsPrimaryKey="@ (prop.Name == "OrderID") "
AllowEditing="@prop.CanWrite"></GridColumn>
}
</GridColumns>
</SfGrid>
@code{
public List<Order> OrderData = new List<Order>
{
    new Order() { OrderID = 10248, CustomerID = "VINET", EmployeeID = 4 },
    new Order() { OrderID = 10249, CustomerID = "TOMSP", EmployeeID = 5 },
    new Order() { OrderID = 10250, CustomerID = "HANAR", EmployeeID = 6 }
};
public class Order
{
    public int? OrderID { get; set; }
    public string CustomerID { get; set; }
    public int? EmployeeID { get; set; }
}
}

```

```
}
}
```

The following image represents DataGrid with dynamically build columns,

OrderID	CustomerID	EmployeeID
10248	VINET	4
10249	TOMSP	5
10250	HANAR	6

Complex data binding

You can achieve complex data binding in the DataGrid by using the dot(.) operator in the column.field. In the following examples, **Name.FirstName** and **Name.LastName** are complex data.

ASPX-CS

```
@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Employees" Height="315">
  <GridColumn>
    <GridColumn Field=@nameof(EmployeeData.EmployeeID) HeaderText="EmployeeID"
      TextAlign="TextAlign.Right" Width="120"></GridColumn>
    <GridColumn Field="Name.FirstName" HeaderText="First Name"
      Width="150"></GridColumn>
    <GridColumn Field="Name.LastName" HeaderText="Last
      Name"Width="130"></GridColumn>
    <GridColumn Field=@nameof(EmployeeData.Title) HeaderText="Title" Format="C2"
      TextAlign="TextAlign.Right" Width="120"></GridColumn>
  </GridColumn>
</SfGrid>
@code{
public List<EmployeeData> Employees { get; set; }
protected override void OnInitialized()
{
Employees = Enumerable.Range(1, 9).Select(x => new EmployeeData()
{
EmployeeID = x,
Name = new EmployeeName() {
FirstName = (new string[] { "Nancy", "Andrew", "Janet", "Margaret", "Steven"
})[new Random().Next(5)],
LastName = (new string[] { "Davolio", "Fuller", "Leverling", "Peacock",
"Buchanan" })[new Random().Next(5)]
},
},
},
```

```

Title = (new string[] { "Sales Representative", "Vice President, Sales",
"Sales Manager",
"Inside Sales Coordinator" })[new Random().Next(4)],
}).ToList();
}
public class EmployeeData
{
public int? EmployeeID { get; set; }
public EmployeeName Name { get; set; }
public string Title { get; set; }
}
public class EmployeeName
{
public string FirstName { get; set; }
public string LastName { get; set; }
}
}

```

The following image represents complex data binding

EmployeeID	First Name	Last Name	Title
1	Steven	Buchanan	Sales Manager
2	Janet	Buchanan	Vice President, Sales
3	Steven	Leverling	Sales Representative
4	Margaret	Peacock	Sales Manager
5	Margaret	Davolio	Sales Representative
6	Andrew	Leverling	Sales Manager
7	Steven	Buchanan	Inside Sales Coordinator
8	Steven	Leverling	Vice President, Sales
9	Steven	Buchanan	Sales Representative

For OData and ODataV4 adaptors, you need to add expand query to the query property (of DataGrid) to load the complex data.

ExpandoObject Complex data binding

Before proceeding this, learn about [ExpandoObject Binding](#). You can achieve ExpandoObject complex data binding in the DataGrid by using the dot(.) operator in the column.field. In the following examples, `CustomerID.Name` and `ShipCountry.Country` are complex data.

ASPX-CS

```

@using Syncfusion.Blazor.Grids
@using System.Dynamic
<SfGrid DataSource="@Orders" AllowPaging="true" AllowFiltering="true"
AllowSorting="true" AllowGrouping="true" Toolbar="@ToolbarItems">
<GridEditSettings AllowAdding="true" AllowDeleting="true"
AllowEditing="true"></GridEditSettings>
<GridColumn>

```

```

<GridColumn Field="OrderID" HeaderText="Order ID" IsPrimaryKey="true"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field="CustomerID.Name" HeaderText="Customer Name"
Width="120"></GridColumn>
<GridColumn Field="Freight" HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field="OrderDate" HeaderText=" Order Date" Format="d"
TextAlign="TextAlign.Right" Width="130" Type="ColumnType.Date"></GridColumn>
<GridColumn Field="ShipCountry.Country" HeaderText="Ship Country"
Width="150"></GridColumn>
<GridColumn Field="Verified" HeaderText="Active" DisplayAsCheckBox="true"
Width="150"></GridColumn>
</GridColumn>
</SfGrid>
@code {
public List<ExpandoObject> Orders { get; set; } = new List<ExpandoObject>();
private List<string> ToolbarItems = new List<string>() { "Add", "Edit",
"Delete", "Update", "Cancel" };
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select((x) =>
{
dynamic d = new ExpandoObject();
dynamic customerName = new ExpandoObject();
dynamic countryName = new ExpandoObject();
d.OrderID = 1000 + x;
customerName.Name = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP",
"BOLID" })[new Random().Next(5)];
d.CustomerID = customerName;
d.Freight = (new double[] { 2, 1, 4, 5, 3 })[new Random().Next(5)] * x;
d.OrderDate = (new DateTime[] { new DateTime(2010, 11, 5), new
DateTime(2018, 10, 3), new DateTime(1995, 9, 9), new DateTime(2012, 8, 2),
new DateTime(2015, 4, 11) })[new Random().Next(5)];
countryName.Country = (new string[] { "USA", "UK" })[new Random().Next(2)];
d.ShipCountry = countryName;
d.Verified = (new bool[] { true, false })[new Random().Next(2)];
return d;
}).Cast<ExpandoObject>().ToList<ExpandoObject>();
}
}

```

* you can perform the Data operations and CRUD operations for Complex ExpandoObject binding fields too.

The following image represents ExpandoObject complex data binding

Drag a column header here to group its column						
<div> <div>+ Add</div> <div> Edit</div> <div> Delete</div> <div> Update</div> <div> Cancel</div> </div>						
Order ID	Customer Name	Freight	Order Date	Ship Country	Active	
<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	
1001	ALFKI	\$5.00	4/11/2015	USA	<input checked="" type="checkbox"/>	
1002	BOLID	\$6.00	4/11/2015	USA	<input type="checkbox"/>	
1003	BOLID	\$15.00	8/2/2012	UK	<input type="checkbox"/>	
1004	BLONP	\$8.00	10/3/2018	USA	<input checked="" type="checkbox"/>	
1005	ANANTR	\$5.00	10/3/2018	USA	<input type="checkbox"/>	
1006	BLONP	\$30.00	9/9/1995	USA	<input type="checkbox"/>	
1007	ANTON	\$28.00	8/2/2012	UK	<input type="checkbox"/>	
1008	ANTON	\$24.00	8/2/2012	UK	<input type="checkbox"/>	

DynamicObject Complex data binding

Before proceeding this, learn about [DynamicObject Binding](#). You can achieve DynamicObject complex data binding in the datagrid by using the dot(.) operator in the column.field. In the following examples, `CustomerID.Name` and `ShipCountry.Country` are complex data.

ASPX-CS

```
@using Syncfusion.Blazor.Grids
@using System.Dynamic

<SfGrid DataSource="@Orders" AllowPaging="true" AllowFiltering="true"
AllowSorting="true" AllowGrouping="true" Toolbar="@ToolbarItems">
  <GridEditSettings AllowAdding="true" AllowDeleting="true"
AllowEditing="true"></GridEditSettings>
  <GridColumns>
    <GridColumn Field="OrderID" HeaderText="Order ID" IsPrimaryKey="true"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
    <GridColumn Field="CustomerID.Name" HeaderText="Customer Name"
Width="150"></GridColumn>
    <GridColumn Field="OrderDate" HeaderText="Order Date" Format="d"
Type="ColumnType.Date" TextAlign="TextAlign.Right"
EditType="EditType.DatePickerEdit" Width="130"></GridColumn>
    <GridColumn Field="Freight" HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
    <GridColumn Field="ShipCountry.Country" HeaderText="Ship Country"
Width="150"></GridColumn>
  </GridColumns>
</SfGrid>

@code {
private List<string> ToolbarItems = new List<string>() { "Add", "Edit",
"Delete", "Update", "Cancel" };
public List<DynamicDictionary> Orders = new List<DynamicDictionary>() { };
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 1075).Select((x) =>
{
dynamic d = new DynamicDictionary();
```



```
dynamic combo = new DynamicDictionary();
dynamic countryName = new DynamicDictionary();
d.OrderID = 1000 + x;
combo.Name = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)];
d.CustomerID = combo;
d.Freight = (new double[] { 2, 1, 4, 5, 3 })[new Random().Next(5)] * x;
d.OrderDate = DateTime.Now.AddDays(-x);
countryName.Country = (new string[] { "USA", "UK" })[new Random().Next(2)];
d.ShipCountry = countryName;
return d;
}).Cast<DynamicDictionary>().ToList<DynamicDictionary>();
}
public class DynamicDictionary : DynamicObject
{
    Dictionary<string, object> dictionary = new Dictionary<string, object>();
    public override bool TryGetMember(GetMemberBinder binder, out object result)
    {
        string name = binder.Name;
        return dictionary.TryGetValue(name, out result);
    }
    public override bool TrySetMember(SetMemberBinder binder, object value)
    {
        dictionary[binder.Name] = value;
        return true;
    }
    public override System.Collections.Generic.IEnumerable<string>
    GetDynamicMemberNames()
    {
        return this.dictionary?.Keys;
    }
}
}
```

* you can perform the Data operations and CRUD operations for Complex DynamicObject binding fields too.

The following image represents DynamicObject complex data binding

Drag a column header here to group its column				
<div> <div>+</div> Add <div></div> Edit <div></div> Delete <div></div> Update <div></div> Cancel </div>				
Order ID	Customer Name	Order Date	Freight	Ship Country
<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>
1001	BLONP	3/30/2021	\$3.00	UK
1002	BLONP	3/29/2021	\$4.00	UK
1003	ANANTR	3/28/2021	\$6.00	UK
1004	ANANTR	3/27/2021	\$16.00	USA
1005	BLONP	3/26/2021	\$10.00	USA
1006	ALFKI	3/25/2021	\$18.00	USA
1007	ANTON	3/24/2021	\$21.00	UK
1008	BLONP	3/23/2021	\$8.00	UK

Foreign key column

Foreign key column can be enabled by using [ForeignKeyDataSource](#), [ForeignKeyField](#) and [ForeignKeyValue](#) properties of **GridForeignKeyColumn** directive.

- [ForeignKeyDataSource](#) - Defines the foreign data.
- [ForeignKeyField](#) - Defines the mapping column name to the foreign data.
- [ForeignKeyValue](#) - Defines the display field from the foreign data.

Foreign key column - Local Data

In the following example, **Employee Name** is a foreign column which shows **FirstName** column from foreign data.

ASPX-CS

```
@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders" Height="315">
  <GridColumns>
    <GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
      TextAlign="TextAlign.Right" Width="120"></GridColumn>
    <GridForeignKeyColumn Field=@nameof(Order.EmployeeID) HeaderText="Employee
      Name" ForeignKeyValue="FirstName" ForeignKeyDataSource="@Employees"
      Width="150"></GridForeignKeyColumn>
    <GridColumn Field=@nameof(Order.OrderDate) HeaderText="Order Date"
      Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
      Width="130"></GridColumn>
    <GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
      TextAlign="TextAlign.Right" Width="120"></GridColumn>
  </GridColumns>
</SfGrid>
@code{
  public List<Order> Orders { get; set; }
  public List<EmployeeData> Employees { get; set; }
  protected override void OnInitialized()
  {
```

```

Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
    OrderID = 1000 + x,
    EmployeeID = x,
    Freight = 2.1 * x,
    OrderDate = DateTime.Now.AddDays(-x),
}).ToList();
Employees = Enumerable.Range(1, 75).Select(x => new EmployeeData()
{
    EmployeeID = x,
    FirstName = (new string[] { "Nancy", "Andrew", "Janet", "Margaret", "Steven"
    })[new Random().Next(5)],
}).ToList();
}
}
public class Order
{
    public int? OrderID { get; set; }
    public int? EmployeeID { get; set; }
    public DateTime? OrderDate { get; set; }
    public double? Freight { get; set; }
}
public class EmployeeData
{
    public int? EmployeeID { get; set; }
    public string FirstName { get; set; }
}
}

```

ForeignKey Column - Remote Data

In the following example, **Employee Name** is a foreign column which shows **FirstName** column from foreign data.

ASPX-CS

```

@using Syncfusion.Blazor
@using Syncfusion.Blazor.Grids
@using Syncfusion.Blazor.Grids.Internal
<SfGrid DataSource="@Orders" Height="250">
    <GridColumns>
        <GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
        TextAlign="TextAlign.Right" Width="120"></GridColumn>
        <GridForeignKeyColumn TValue="EmployeeData" Field=@nameof(Order.EmployeeID)
        HeaderText="Employee Name" ForeignKeyValue="FirstName" Width="150">
            <Syncfusion.Blazor.Data.SfDataManager
            Url="https://js.syncfusion.com/demos/ejServices/Wcf/Northwind.svc/Employees"
            CrossDomain="true" Adaptor="Adaptors.ODataAdaptor">
            </Syncfusion.Blazor.Data.SfDataManager>
        </GridForeignKeyColumn>
        <GridColumn Field=@nameof(Order.OrderDate) HeaderText="Order Date"
        Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
        Width="130"></GridColumn>
        <GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
        TextAlign="TextAlign.Right" Width="120"></GridColumn>
    </GridColumns>
</SfGrid>
@code{

```

```

public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
    Orders = Enumerable.Range(1, 75).Select(x => new Order()
    {
        OrderID = 1000 + x,
        EmployeeID = x,
        Freight = 2.1 * x,
        OrderDate = DateTime.Now.AddDays(-x),
    }).ToList();
}
public class Order
{
    public int? OrderID { get; set; }
    public int? EmployeeID { get; set; }
    public DateTime? OrderDate { get; set; }
    public double? Freight { get; set; }
}
public class EmployeeData
{
    public int? EmployeeID { get; set; }
    public string FirstName { get; set; }
}
}

```

The following image represents foreign key column

Order ID	Employee Name	Order Date	Freight
1001	Janet	7/30/2019	\$2.10
1002	Margaret	7/29/2019	\$4.20
1003	Steven	7/28/2019	\$6.30
1004	Nancy	7/27/2019	\$8.40
1005	Janet	7/26/2019	\$10.50
1006	Margaret	7/25/2019	\$12.60
1007	Janet	7/24/2019	\$14.70
1008	Margaret	7/23/2019	\$16.80
1009	Andrew	7/22/2019	\$18.90

* For remote data, the sorting and grouping is done based on [ForeignKeyField](#) instead of [ForeignKeyValue](#).

* If [ForeignKeyField](#) is not defined, then the column uses [Field](#) property of **GridColumn** tag helper.

Header text

By default, column header title is displayed from column [Field](#) value. To override the default header title, you have to define the **HeaderText** value in the [HeaderText](#) property of **GridColumn** directive.

ASPX-CS

```

@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders" Height="315">
  <GridColumn>
    <GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
      TextAlign="TextAlign.Right" Width="120"></GridColumn>
    <GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
      Width="150"></GridColumn>
    <GridColumn Field=@nameof(Order.OrderDate) HeaderText="Order Date"
      Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
      Width="130"></GridColumn>
    <GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
      TextAlign="TextAlign.Right" Width="120"></GridColumn>
  </GridColumn>
</SfGrid>
@code{
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
  Orders = Enumerable.Range(1, 75).Select(x => new Order()
  {
    OrderID = 1000 + x,
    CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
  })[new Random().Next(5)],
    Freight = 2.1 * x,
    OrderDate = DateTime.Now.AddDays(-x),
  }).ToList();
}
public class Order {
  public int? OrderID { get; set; }
  public string CustomerID { get; set; }
  public DateTime? OrderDate { get; set; }
  public double? Freight { get; set; }
}
}

```

The Output image for header text is as follows

Order ID	Customer Name	Order Date	Freight
1001	BOLID	7/30/2019	\$2.10
1002	BOLID	7/29/2019	\$4.20
1003	ALFKI	7/28/2019	\$6.30
1004	ANANTR	7/27/2019	\$8.40
1005	BOLID	7/26/2019	\$10.50
1006	ANANTR	7/25/2019	\$12.60
1007	ANTON	7/24/2019	\$14.70
1008	ANTON	7/23/2019	\$16.80
1009	ALFKI	7/22/2019	\$18.90

* If both the [Field](#) and [HeaderText](#)

are not defined in the column, the column renders with “empty” header text.

Header template

Before adding header template to the datagrid, we strongly recommend you to go through the [template](#) section topic to configure the template.

To know about **Header Template** in Blazor DataGrid Component, you can check this video.

{% youtube

"youtube:https://www.youtube.com/watch?v=9YF9HnFY5Ew"%}

The Header Template has options to display custom element value or content in the header. You can use the [HeaderTemplate](#) of the [GridColumn](#) component to specify the custom content.



ASPX-CS

```
@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Employees" AllowPaging="true" Height="315">
  <GridColumn>
    <GridColumn Field=@nameof(EmployeeData.EmployeeID)
      TextAlign="TextAlign.Right" Width="120">
      <HeaderTemplate>
        <div>
          <span class="e-icon-userlogin e-icons employee"></span> Emp ID
        </div>
      </HeaderTemplate>
    </GridColumn>
    <GridColumn Field=@nameof(EmployeeData.FirstName) HeaderText="First Name"
      Width="130"></GridColumn>
    <GridColumn Field=@nameof(EmployeeData.OrderDate) HeaderText="Last Name"
      Format="d" TextAlign="TextAlign.Right" Width="120">
      <HeaderTemplate>
        <div>
          <span class="e-icon-calender e-icons headericon"></span> Order Date
        </div>
      </HeaderTemplate>
    </GridColumn>
    <GridColumn Field=@nameof(EmployeeData.City) HeaderText="City"
      Width="150"></GridColumn>
    <GridColumn Field=@nameof(EmployeeData.Country) HeaderText="Country"
      TextAlign="TextAlign.Right" Width="150"></GridColumn>
  </GridColumn>
</SfGrid>
<style>
@@font-face {
font-family: 'ej2-headertemplate';
src: url(data:application/x-font-ttf;charset=utf-
8;base64,AAEAAAKAIAAAwAgTlMvMjlvTFIAAAEoAAAAVmNtYXDS2c5qAAABjAAAAEBnbHlmQmN
FrQAAAdQAAANoAGVhZBRdbkIAAADQAAANmhoZWEIUQQEAAAArAAAACRobXR4DAAAAAAAAAYAAAA
MbG9jYQCOAbQAAAHMAAACG1heHABHgENAAABCAAAACBuYW1lohGZJQAABTWAAAKpcG9zdA2o3w0
AAAFoAAAAQAABAAAEAAAAAFwEAAAAAAD9AABAAAAAAAAAAAAAAAAAAAAAwABAAAAQAATBXy9l8
PPPUACwQAAAAANillKkAAAAA2KWUqQAAAAAD9APzAAAACAACAAAAAAAAAAAAEAAAADAQEAEQAAAA
AAgAAAAoACgAAAP8AAAAAAAAAAQQAABQAAAAokCzAAAAI8CiQLMAAAB6wAyAQgAAAIABQMAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAUGZFZABA5wLpYAQAAAAAXAQAAAAAAAAABAAAAAAAAABAAAAQAAAA
EAAAAAAAAAgAAAAAMAAAAUAAAAQAAABQABAAsAAAAABgAEAAEAucC6WD//wAA5wLpYP//AAAAAA
BAAYABgAAAAIAAQAAAAAAjgG0ABEAAAAA8kD8wADAACACwAPABMAFwAbAB8AIwAnACsALwAzADc
AOwBPAGsAACUVIzUjFSM1IxUjNSMVIzUjFSM1JRujNSMVIzUjFSM1IxUjNSMVIzU1FSM1IxUjNSM
```

```
VizUjFSM1IxUjNQMVHwYhPwcRITCjDwghNS8HIzUjFSE1Iwn2U1NTU1RTU1NTAuxTUI1NTVFNTU1MC7FNTU1NUU1NTU1QCAWUGBgIA0QICAcHBQQBAVxsp30ICAcHAgUDAQED1AECBAUHBwgIfVP+YFOzU1NTU1NTU1NTU6dUVFRUVFRUVFRUp1NTU1NTU1NTU1P+NqQIBwcGBAMCAQIEBQCCHAwGcdPoBAGQFAwChCIF8CQGHBgUEAgFTU1MABAAAAAAD9APeADQAbQCuAQAAAAAEfCDclLwclPwIPBy8HHww3HwQPATMPVpc1Lx0jdWmfAgUDar8GBTUZLxQjDxD0BFxUFEdSBPxAlNyc1LXIPEhUCCg8OGxcVeXANCgMBAQMDCQQDAgECAxESEhmTEExUUFRQTfBISeHEHCWYHCAkKcW0NDw8SuQQGAwIBagRxlgSKCQCGBAMBAGMDAwUFBQcGBwgICQkKCgsKDAsMDQwNDQ4NDg45BQUDAQEEA/1eAgUGBwkKcWhjeggdKhEUfxS1FRMSEA4NCwoJBwcJBjwODg0ODQ0MDQwLDAoLCgoJCQgIBWyHBQUFAwMDAgEBAQEACAgYICg0ODXiSFBUXFfwMDA0MDQwMFxcvFBISDw4MCwgGAgIBAQCICAWcJcW0PERITFRUXDAwMDA0NDawMDASXFRQTEQ8ODQoIBgICAVQEAWgJCgsMCwwbEBAREREZEAE8VDawKCgoIBwyFAwIBAQIDBQYHCAoUFQWLcwsLCgoJCACGMwsWFhUVHB3QAQIEBggICgueDg4ODg0NDAMCwsLCwoKCQgJBwgGBwUFBAQDAWECCw8NDxETCrilawsKCAGGBAIB0AoLCwoLCQgNCakLDA0NDg4ODg4ZFAIBAwMEBAUGBgYIBwkICQoKCwsLDawMDA0ODQ4ODgH7DQwMDBgWFRQTERAOdaOIBgICAQEACAgYICgwOEbetTFBUWGAWMDA0MDQwMCxcWFRMSEA8NDakHAwIBAQBEBACAWMICcwOEbetTFBUFWwMDQAAAAASAN4AAQAAAAAAAAABAAAAQA
AAAAAAAAQASAAEAQAQAAAAAAAgAHABMAAQAAAAAAAwASABoAAQAAAAABAASACwAAQAAAAABQALAD4
AAQAAAAABgASAEKAQAQAAAAAACgASAfSAQAQAAAAACWASAicAAwABBakAAAACAjkaAwABBakAAQA
kAJsaAwABBakAAgAOL8AAwABBakAAwAkAM0AAwABBakABAakAPEAAwABBakABQAWARUAawABBak
ABgAKasAAwABBakACgBYAU8AAwABBakACwAkAacgzWoyLWhlYWRLcnRlbXBsYXRlUmVndWxhcmlV
qMilozWFkZXJ0ZWlwbgGF0ZWVqMilozWFkZXJ0ZWlwbgGF0ZVZlcnlbnpb24gMS4wZWoyLWhlYWRLcnR
lbXBsYXRlRm9udCBnZW5lcmlF0ZWQgdXNpbmcgU3luY2Zlc2lvbiBNZXRYbyBTdHVkaW93d3cuc3l
uY2Zlc2lvbi5jb20AIABlAGoAMgAtAGgAZQBhAGQAZQBByAHQAZQBTATHAAAbABhAHQAZQBSAGUAZWB
lAGWAYQByAGUAagAyAC0AaABlAGEAZABlAHlAdABlAG0AcABsAGEAdABlAGUAagAyAC0AaABlAGE
AZABlAHlAdABlAG0AcABsAGEAdABlAFYAZQBByAHMAaQBVag4AIAAxAC4AMABlAGoAMgAtAGgAZQB
hAGQAZQBByAHQAZQBTATHAAAbABhAHQAZQBGAG8Abgb0ACAAZwBlAG4AZQBByAGEAdABlAGQAIABlAHM
AaqBuAGCAIABTAHkAbgBjAGYAdQBzAGkAbwBuACAATQBlAHQAcgBvACAAUwB0AHUAZABpAG8AdwB
3AhCaLgBzAHkAbgBjAGYAdQBzAGkAbwBuAC4AYWVvBAG0AAAAAAGAAAAAAAKAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAADAQIBAwEEAatCVF9DYWxlbmRhcghlbXBsb3llZQAQ)
format('trueType');
font-weight: normal;
font-style: normal;
}
.e-grid .e-icon-userlogin::before {
font-family: 'ej2-headertemplate';
width: 15px !important;
content: '\e702';
}
.e-grid .e-icon-calender::before {
font-family: 'ej2-headertemplate';
width: 15px !important;
content: '\e960';
}
</style>
@code{
public List<EmployeeData> Employees { get; set; }
protected override void OnInitialized()
{
Employees = Enumerable.Range(1, 9).Select(x => new EmployeeData()
{
EmployeeID = x,
FirstName = (new string[] { "Nancy", "Andrew", "Janet", "Margaret", "Steven"
})[new Random().Next(5)],
OrderDate = DateTime.Now.AddDays(-x),
City = (new string[] { "Seattle", "Tacoma", "Redmond", "Kirkland", "London"
})[new Random().Next(5)],
Country = (new string[] { "USA", "UK"})[new Random().Next(2)],
}).ToList();
}
}
```

```
public class EmployeeData
{
    public int? EmployeeID { get; set; }
    public string FirstName { get; set; }
    public DateTime? OrderDate { get; set; }
    public string City { get; set; }
    public string Country { get; set; }
}
```

The following screenshot represents the Header Template.

 Emp ID	First Name	 Order Date	City	Country
1	Andrew	7/24/2019	Kirkland	UK
2	Margaret	7/23/2019	Seattle	USA
3	Steven	7/22/2019	Kirkland	UK
4	Steven	7/21/2019	Tacoma	UK
5	Andrew	7/20/2019	Kirkland	USA
6	Steven	7/19/2019	Redmond	UK
7	Janet	7/18/2019	Redmond	UK
8	Janet	7/17/2019	Redmond	USA
9	Andrew	7/16/2019	Tacoma	UK

Column type

Column type can be specified using the [Type](#) property. It specifies the type of data the column binds.

If the format is defined for a column, the column uses type to select the appropriate format option (number or date).

DataGrid column supports the following types:

- String
- Number
- Boolean
- Date
- DateTime
- CheckBox

If the [Type](#) is not defined, it will be determined from the first record of the [DataSource](#). In case, if the first record of the [DataSource](#) is null/blank value for a column then it is necessary to define the [Type](#) for that column.

Difference between Boolean type and CheckBox type column

- Use GridColumn [Type](#) as Boolean if you want to bind boolean values from your datasource and/or edit Boolean property value from your Type.

- Use [CheckBox](#) as [GridColumn Type](#) for the purpose of selection/deselection of the whole row.

See also section [Render boolean values as checkbox](#) to render boolean values as checkbox in [GridColumn](#).

Format

To format cell values based on specific culture, use the [Format](#) property of **GridColumn** component . The DataGrid uses **Internalization** library to format **number** and **date**.

values.

ASPX-CS

```
@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders" Height="315">
  <GridColumn>
    <GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
      TextAlign="TextAlign.Right" Width="120"></GridColumn>
    <GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
      Width="150"></GridColumn>
    <GridColumn Field=@nameof(Order.OrderDate) HeaderText="Order Date"
      Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
      Width="130"></GridColumn>
    <GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
      TextAlign="TextAlign.Right" Width="120"></GridColumn>
  </GridColumn>
</SfGrid>

@code{
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
  Orders = Enumerable.Range(1, 75).Select(x => new Order()
  {
    OrderID = 1000 + x,
    CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
  })[new Random().Next(5)],
    Freight = 2.1 * x,
    OrderDate = DateTime.Now.AddDays(-x),
  }).ToList();
}

public class Order {
  public int? OrderID { get; set; }
  public string CustomerID { get; set; }
  public DateTime? OrderDate { get; set; }
  public double? Freight { get; set; }
}
```

By default, the **number** and **date** values are formatted in **en-US** locale.

Number formatting

The number or integer values can be formatted using the following format strings.

Format	Description	Remarks
--------	-------------	---------

N | Denotes numeric type. | The numeric format is followed by integer value as N2, N3. etc which denotes the number of precision to be allowed.

C | Denotes currency type. | The currency format is followed by integer value as C2, C3. etc which denotes the number of precision to be allowed.

P | Denotes percentage type | The percentage format expects the input value to be in the range of 0 to 1. For example the cell value **0.2** is formatted as **20%**. The percentage format is followed by integer value as P2, P3. etc which denotes the number of precision to be allowed.

Date formatting

You can format date values either using built-in date format string.

For built-in date format, you can specify the [Format](#) property of **GridColumn** as string.

Visibility

You can hide any particular column in DataGrid before rendering by defining the [Visible](#) property of **GridColumn** as false. In the following sample, **Freight** column is defined as visible false.

ASPX-CS

```
@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders" Height="315">
  <GridColumn>
    <GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
      TextAlign="TextAlign.Right" Width="120"></GridColumn>
    <GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
      Width="150"></GridColumn>
    <GridColumn Field=@nameof(Order.OrderDate) HeaderText="Order Date"
      Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
      Width="130"></GridColumn>
    <GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
      TextAlign="TextAlign.Right" Visible="false" Width="120"></GridColumn>
  </GridColumn>
</SfGrid>

@code{
  public List<Order> Orders { get; set; }
  protected override void OnInitialized()
  {
    Orders = Enumerable.Range(1, 75).Select(x => new Order()
    {
      OrderID = 1000 + x,
      CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
    })[new Random().Next(5)],
      Freight = 2.1 * x,
      OrderDate = DateTime.Now.AddDays(-x),
    }).ToList();
  }

  public class Order {
    public int? OrderID { get; set; }
    public string CustomerID { get; set; }
    public DateTime? OrderDate { get; set; }
    public double? Freight { get; set; }
  }
}
```

The following screenshot represents the DataGrid with Freight column set to visible false.

Order ID	Customer Name	Order Date
1001	BLONP	7/21/2019
1002	ALFKI	7/20/2019
1003	ALFKI	7/19/2019
1004	BOLID	7/18/2019
1005	BLONP	7/17/2019
1006	ANTON	7/16/2019
1007	BLONP	7/15/2019
1008	ANANTR	7/14/2019
1009	BLONP	7/13/2019

K < 1 2 3 4 5 6 7 > X
 1 of 7 pages (75 items)

Width

The column width can be set using the **Width** property of the **GridColumn**. The unit of the width value can be either in pixel (px) or percentage (%). By default, the grid tables use `table-layout:fixed` to speed up the table rendering.

- Columns will respect the width value irrespective of its cell content width.
- Columns with no width set will share the available space equally.
- When all columns are provided with a width value and the cumulative width of all columns is greater than the grid element width, a horizontal scrollbar will be shown.
- When only some columns are provided with the width value and if the cumulative width of the columns is greater than the grid element width then columns with no width might be invisible as their width is zero.
- When only some columns are provided with the width value and if the cumulative width of the columns is lesser than the grid element width then columns with no width will share the available space evenly.
- When no width is provided in a column and MinWidth property is defined, if the cumulative width of the column is greater than the grid element width then MinWidth would be used as the column width to avoid it from becoming invisible.

Autofit

You can auto fit a column interactively by double clicking the right border of the header cells.

Autofit columns at initial rendering

AutoFit resizes the column to fit the widest cell's content without wrapping. To enable AutoFit for specific columns, you need to set the AutoFit property to true.

ASPX-CS

```
@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders" Height="315">
```

```

<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150" AutoFit="true"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText="Order Date"
Format="d" AutoFit="true" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumns>
</SfGrid>
@code{
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
OrderDate = DateTime.Now.AddDays(-x),
}).ToList();
}
public class Order
{
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
}

```

In the following Image, **Autofit** Property is set to true for CustomerName and OrderDate.

Order ID	Customer Name	Order Date	Freight	
1001	BLONP	7/30/2019	\$2.10	
1002	BLONP	7/29/2019	\$4.20	
1003	ALFKI	7/28/2019	\$6.30	
1004	BLONP	7/27/2019	\$8.40	
1005	BLONP	7/26/2019	\$10.50	
1006	ANTON	7/25/2019	\$12.60	
1007	ANANTR	7/24/2019	\$14.70	
1008	BOLID	7/23/2019	\$16.80	
1009	ANANTR	7/22/2019	\$18.90	

Autofit columns by method

The **AutoFitColumns** method resizes the column to fit the widest cell's content without wrapping. You can autofit a specific column at initial rendering by invoking the **AutoFitColumns** method in the [DataBound](#) event.

ASPX-CS

```
@using Syncfusion.Blazor.Grids
<SfGrid @ref="Grid" DataSource="@Orders" Height="315" Width="540">
  <GridEvents DataBound="DataboundHandler" TValue="Order"></GridEvents>
  <GridColumns>
    <GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
      TextAlign="TextAlign.Right" Width="120"></GridColumn>
    <GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
      Width="150"></GridColumn>
    <GridColumn Field=@nameof(Order.OrderDate) HeaderText="Order Date"
      Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
      Width="130"></GridColumn>
    <GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
      TextAlign="TextAlign.Right" Width="120"></GridColumn>
  </GridColumns>
</SfGrid>
@code{
private SfGrid<Order> Grid;
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
  Orders = Enumerable.Range(1, 75).Select(x => new Order()
  {
    OrderID = 1000 + x,
    CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
  })[new Random().Next(5)],
    Freight = 2.1 * x,
    OrderDate = DateTime.Now.AddDays(-x),
  }).ToList();
}
public class Order
{
  public int? OrderID { get; set; }
  public string CustomerID { get; set; }
  public DateTime? OrderDate { get; set; }
  public double? Freight { get; set; }
}
public void DataboundHandler(object args)
{
  this.Grid.AutoFitColumns(new string[] { "OrderDate", "Freight" });
}
}
```

The following image represents AutoFit column by method

Order ID	Customer Name	Order Date	Freight
1001	ALFKI	7/31/2019	\$2.10
1002	ALFKI	7/30/2019	\$4.20
1003	ALFKI	7/29/2019	\$6.30
1004	BOLID	7/28/2019	\$8.40
1005	ALFKI	7/27/2019	\$10.50
1006	ALFKI	7/26/2019	\$12.60
1007	ANANTR	7/25/2019	\$14.70
1008	ANANTR	7/24/2019	\$16.80
1009	ANTON	7/23/2019	\$18.90

You can autofit all the columns by invoking the **AutoFitColumns** method without column names.

Responsive columns

You can toggle column visibility based on media queries. This can be achieved by defining Media Queries in the [HideAtMedia](#) Column property. The [HideAtMedia](#) accepts valid [Media Queries](#).

In the following sample code, for OrderID column - HideAtMedia property value is set as (min-width: 700px). This hides the OrderID column when the browser screen width is less than 700px.

ASPX-CS

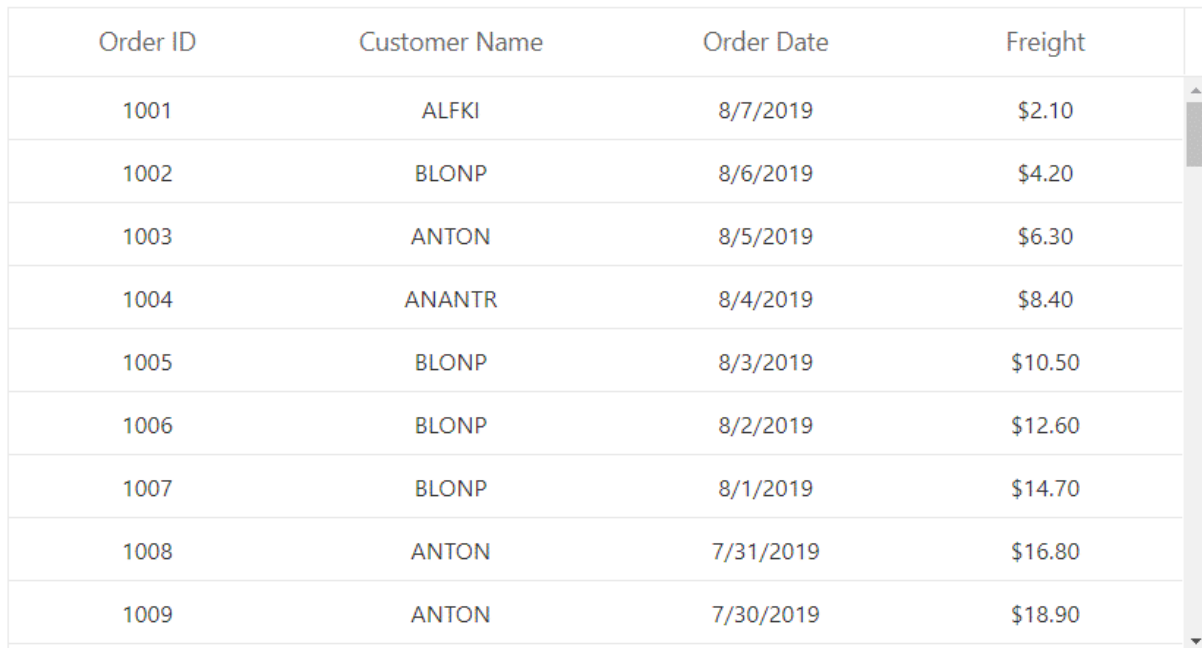
```
@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders" Height="315">
  <GridColumns>
    <GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
      HideAtMedia="(min-width: 700px)" TextAlign="TextAlign.Center"
      Width="120"></GridColumn> // Column hides when browser screen width is less
      than 700px
    <GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
      HideAtMedia="(max-width: 500px)" Width="150"
      TextAlign="TextAlign.Center"></GridColumn> // Column shows when browser
      screen width is less than or equal to 500px
    <GridColumn Field=@nameof(Order.OrderDate) HeaderText="Order Date"
      HideAtMedia="(min-width: 500px)" Format="d" Type="ColumnType.Date"
      TextAlign="TextAlign.Center" Width="130"></GridColumn> // Column hides when
      browser screen width is less than 500px
    <GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
      TextAlign="TextAlign.Center" Width="120"></GridColumn> // Column is always
      displayed
  </GridColumns>
</SfGrid>
@code{
  public List<Order> Orders { get; set; }
  protected override void OnInitialized()
  {
    Orders = Enumerable.Range(1, 75).Select(x => new Order()
    {
      OrderID = 1000 + x,
```

```

CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
OrderDate = DateTime.Now.AddDays(-x),
}).ToList();
}
public class Order
{
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
}

```

The following GIF shows the responsive columns behavior when the window is resized,



Order ID	Customer Name	Order Date	Freight
1001	ALFKI	8/7/2019	\$2.10
1002	BLONP	8/6/2019	\$4.20
1003	ANTON	8/5/2019	\$6.30
1004	ANANTR	8/4/2019	\$8.40
1005	BLONP	8/3/2019	\$10.50
1006	BLONP	8/2/2019	\$12.60
1007	BLONP	8/1/2019	\$14.70
1008	ANTON	7/31/2019	\$16.80
1009	ANTON	7/30/2019	\$18.90

Controlling datagrid actions

You can enable or disable datagrid action for a particular column by using the [AllowFiltering](#), [AllowGrouping](#), [AllowSorting](#), [AllowReordering](#), and [AllowEditing](#) properties.

The following sample code shows DataGrid actions disabled for particular columns,

ASPX-CS

```

@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders" Height="315" AllowFiltering="true"
AllowGrouping="true" AllowSorting="true" AllowReordering="true">
<GridEditSettings AllowAdding="true" AllowDeleting="true"
AllowEditing="true" Mode=EditMode.Normal></GridEditSettings>
<GridColumns>

```

```
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
TextAlign="TextAlign.Center" IsPrimaryKey="true" Width="120"
AllowGrouping="false"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150" TextAlign="TextAlign.Center"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText="Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Center" Width="130"
AllowEditing="false" AllowFiltering="false"
AllowReordering="false"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Center" Width="120" AllowSorting="false"></GridColumn>
</GridColumns>
</SfGrid>
@code{
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
OrderDate = DateTime.Now.AddDays(-x),
}).ToList();
}
public class Order
{
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
}
```

The following GIF shows the DataGrid actions for different columns,

Drag a column header here to group its column			
Order ID	Customer Name	Order Date	Freight
1001	ANTON	8/7/2019	\$2.10
1002	BOLID	8/6/2019	\$4.20
1003	BOLID	8/5/2019	\$6.30
1004	BLONP	8/4/2019	\$8.40
1005	ANANTR	8/3/2019	\$10.50
1006	BOLID	8/2/2019	\$12.60
1007	ALFKI	8/1/2019	\$14.70
1008	ANTON	7/31/2019	\$16.80
1009	BOLID	7/30/2019	\$18.90

Show/hide columns by external button

You can show or hide DataGrid columns dynamically using external buttons by invoking the [ShowColumns](#) or [HideColumns](#) method.

The following sample code demonstrates showing and hiding of columns using their header texts ("Order Date", "Freight") on button click,

ASPX-CS

```
@using Syncfusion.Blazor.Buttons
@using Syncfusion.Blazor.Grids
<SfButton OnClick="Show" CssClass="e-primary" Content="Show"></SfButton>
<SfButton OnClick="Hide" CssClass="e-primary" Content="Hide"></SfButton>
<SfGrid @ref="DefaultGrid" DataSource="@Orders" Height="315">
  <GridColumn>
    <GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
      TextAlign="TextAlign.Center" Width="150"></GridColumn>
    <GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
      Width="150" TextAlign="TextAlign.Center"></GridColumn>
    <GridColumn Field=@nameof(Order.OrderDate) HeaderText="Order Date"
      Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Center"
      Width="130"></GridColumn>
    <GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
      TextAlign="TextAlign.Center" Width="120"></GridColumn>
  </GridColumn>
</SfGrid>
@code{
  private SfGrid<Order> DefaultGrid;
  public List<Order> Orders { get; set; }
  public string[] ColumnItems = new string[] { "Order Date", "Freight" };
```

```
protected override void OnInitialized()
{
    Orders = Enumerable.Range(1, 75).Select(x => new Order()
    {
        OrderID = 1000 + x,
        CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
    })[new Random().Next(5)],
        Freight = 2.1 * x,
        OrderDate = DateTime.Now.AddDays(-x),
    }).ToList();
}

public class Order
{
    public int? OrderID { get; set; }
    public string CustomerID { get; set; }
    public DateTime? OrderDate { get; set; }
    public double? Freight { get; set; }
}

public void Show()
{
    // Show columns by its header text
    this.DefaultGrid.ShowColumns(ColumnItems);
}

public void Hide()
{
    // Hide columns by its header text
    this.DefaultGrid.HideColumns(ColumnItems);
}
}
```

The following GIF represents the showing/hiding DataGrid columns on button click,

Show
Hide

Order ID	Customer Name	Order Date	Freight
1001	ANTON	8/8/2019	\$2.10
1002	BOLID	8/7/2019	\$4.20
1003	BOLID	8/6/2019	\$6.30
1004	ANTON	8/5/2019	\$8.40
1005	ANTON	8/4/2019	\$10.50
1006	ALFKI	8/3/2019	\$12.60
1007	ANANTR	8/2/2019	\$14.70
1008	BLONP	8/1/2019	\$16.80
1009	ANANTR	7/31/2019	\$18.90

<!-- ValueAccessor

The [ValueAccessor](#) property is used to access/manipulate the value of the displayed data. You can achieve custom value formatting by using the [ValueAccessor](#) property.

The following sample code demonstrates the displayed data manipulated for two columns using the [ValueAccessor](#) property,

ASPX-CS

```
@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders" Height="315">
  <GridColumn>
    <GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
      TextAlign="TextAlign.Center" Width="150"></GridColumn>
    <GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
      Width="150" TextAlign="TextAlign.Center"></GridColumn>
    <GridColumn Field=@nameof(Order.OrderDate) HeaderText="Order Date"
      Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Center"
      Width="130"></GridColumn>
    <GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
      TextAlign="TextAlign.Center" Width="120"
      ValueAccessor=@CurrencyFormatter></GridColumn>
  </GridColumn>
</SfGrid>
@code{
  public List<Order> Orders { get; set; }
  protected override void OnInitialized()
  {
    Orders = Enumerable.Range(1, 75).Select(x => new Order()
    {
      OrderID = 1000 + x,
      CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
    })[new Random().Next(5)],
      Freight = 2.1 * x,
      OrderDate = DateTime.Now.AddDays(-x),
    }).ToList();
  }
  public class Order
  {
    public int? OrderID { get; set; }
    public string CustomerID { get; set; }
    public DateTime? OrderDate { get; set; }
    public double? Freight { get; set; }
  }
  public string CurrencyFormatter(String Field, Object Data, Object Column)
  {
    return "€" + new List<string> Data["Freight"];
  }
}
```

The following image represents the Grid with manipulated column data. -->

<!-- Display array type columns

You can bind an array of objects in a column by using the [ValueAccessor](#) property.

In the following sample code, the name field has an array of two objects, FirstName and LastName are joined and bound to a column using the [ValueAccessor](#) property.

ASPX-CS

```
@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Employees">
<GridColumns>
<GridColumn Field=@nameof(EmployeeData.EmployeeID) HeaderText="Employee ID"
TextAlign="TextAlign.Center" Width="120"></GridColumn>
<GridColumn Field=@nameof(EmployeeData.Name) HeaderText="Full Name"
TextAlign="TextAlign.Center" ValueAccessor=@ValueAccess
Width="130"></GridColumn>
<GridColumn Field=@nameof(EmployeeData.Title) HeaderText="Title"
TextAlign="TextAlign.Center" Width="120"></GridColumn>
</GridColumns>
</SfGrid>
@code{
List<EmployeeData> Employees = new List<EmployeeData>
{
new EmployeeData() { EmployeeID = 1, Name = { FirstName = "Nancy", LastName
= "Davolio" }, Title = "Vice President, Sales" },
new EmployeeData() { EmployeeID = 2, Name = { FirstName = "Andrew", LastName
= "Fuller" }, Title = "Sales Representative" },
new EmployeeData() { EmployeeID = 3, Name = { FirstName = "Steven", LastName
= "Buchanan" }, Title = "Sales Representative" },
new EmployeeData() { EmployeeID = 4, Name = { FirstName = "Michael",
LastName = "Suyama" }, Title = "Inside Sales Coordinator" },
new EmployeeData() { EmployeeID = 5, Name = { FirstName = "Laura", LastName
= "Callahan" }, Title = "Sales Manager" },
new EmployeeData() { EmployeeID = 6, Name = { FirstName = "Anne", LastName =
"Dodsworth" }, Title = "Sales Representative" },
new EmployeeData() { EmployeeID = 7, Name = { FirstName = "Robert", LastName
= "King" }, Title = "Sales Representative" },
new EmployeeData() { EmployeeID = 8, Name = { FirstName = "Margaret",
LastName = "Peacock" }, Title = "Sales Representative" }
};
public class EmployeeData
{
public int EmployeeID { get; set; }
public EmployeeName Name { get; set; }
public string Title { get; set; }
}
public class EmployeeName
{
public string FirstName { get; set; }
public string LastName { get; set; }
}
public string ValueAccess(string Field, object Data, object Column)
{
return Data[Field].Select(s => s.LastName || s.FirstName).Join("");
}
}
```

The following image represents the bound column data in Grid using ValueAccessor property -->

<!-- Expression column

You can achieve the expression column by using the [ValueAccessor](#) property.

The following sample code demonstrates expression column achieved using [ValueAccessor](#) for **Calories** column,

ASPX-CS

```
@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@FoodInformation">
  <GridColumns>
    <GridColumn Field=@nameof(FoodData.FoodName) HeaderText="Food Name"
    TextAlign="TextAlign.Center" Width="120"></GridColumn>
    <GridColumn Field=@nameof(FoodData.Time) HeaderText="Intake Time"
    TextAlign="TextAlign.Center" Width="130"></GridColumn>
    <GridColumn Field=@nameof(FoodData.Protein) HeaderText="Protein"
    TextAlign="TextAlign.Center" Width="130"></GridColumn>
    <GridColumn Field=@nameof(FoodData.Fat) HeaderText="Fat"
    TextAlign="TextAlign.Center" Width="120"></GridColumn>
    <GridColumn Field=@nameof(FoodData.Carbohydrate) HeaderText="Carbohydrate"
    TextAlign="TextAlign.Center" Width="120"></GridColumn>
    <GridColumn HeaderText="Calorie" TextAlign="TextAlign.Center"
    ValueAccessor=@TotalCalories Width="120"></GridColumn>
  </GridColumns>
</SfGrid>

@code{
List<FoodData> FoodInformation = new List<FoodData>
{
    new FoodData() { FoodId = 1, Time = "8:40 AM", FoodName = "CHEESE BURGER",
    Protein = 15, Fat = 15, Carbohydrate = 28 },
    new FoodData() { FoodId = 2, Time = "10:30 AM", FoodName = "PIZZA", Protein
    = 15, Fat = 9, Carbohydrate = 8 },
    new FoodData() { FoodId = 3, Time = "12:45 PM", FoodName = "CHICKEN NOODLE",
    Protein = 4, Fat = 2, Carbohydrate = 9 },
    new FoodData() { FoodId = 4, Time = "5:30 PM", FoodName = "YOGURT", Protein
    = 10, Fat = 2, Carbohydrate = 43 },
    new FoodData() { FoodId = 5, Time = "6:30 PM", FoodName = "BEEF SANDWICH",
    Protein = 22, Fat = 13, Carbohydrate = 34 },
    new FoodData() { FoodId = 6, Time = "7:00 PM", FoodName = "CHICKEN BURGER",
    Protein = 15, Fat = 10, Carbohydrate = 25 },
    new FoodData() { FoodId = 7, Time = "8:00 PM", FoodName = "VEG BURGER",
    Protein = 14, Fat = 5, Carbohydrate = 53 },
    new FoodData() { FoodId = 8, Time = "9:00 PM", FoodName = "VEG SANDWICH",
    Protein = 20, Fat = 15, Carbohydrate = 24 },
    new FoodData() { FoodId = 9, Time = "11:00 PM", FoodName = "EGG BURGER",
    Protein = 25, Fat = 16, Carbohydrate = 35 },
};
public class FoodData
{
    public int FoodId { get; set; }
    public string Time { get; set; }
    public string FoodName { get; set; }
    public int Protein { get; set; }
    public int Fat { get; set; }
    public int Carbohydrate { get; set; }
}
public int TotalCalories(string Field, FoodData Data, object Column)
```

```
{
return Data.Protein * 4 + Data.Fat * 9 + Data.Carbohydrate * 4;
}
}
```

The following image represents expression column achieved in Grid -->

Render boolean values as checkbox

To render boolean values as checkbox in columns, you need to set [DisplayAsCheckBox](#) property as true.

The following sample code demonstrates [DisplayAsCheckBox](#) property enabled for **Verified** column,

ASPX-CS

```
@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@OrderData">
<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
TextAlign="TextAlign.Center" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer ID"
TextAlign="TextAlign.Center" Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight"
TextAlign="TextAlign.Center" Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.ShipName) HeaderText="Ship Name"
TextAlign="TextAlign.Center" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.Verified) HeaderText="Verified"
TextAlign="TextAlign.Center" DisplayAsCheckBox="true"
Width="120"></GridColumn>
</GridColumns>
</SfGrid>
@code{
List<Order> OrderData = new List<Order>
{
new Order() { OrderID = 10248, CustomerID = "VINET", Freight = 32.38,
ShipName = "Vins et alcools Chevalier", Verified = true },
new Order() { OrderID = 10249, CustomerID = "TOMSP", Freight = 11.61,
ShipName = "Toms Spezialitäten", Verified = false },
new Order() { OrderID = 10250, CustomerID = "HANAR", Freight = 65.83,
ShipName = "Hanari Carnes", Verified = true },
new Order() { OrderID = 10251, CustomerID = "VICTE", Freight = 41.34,
ShipName = "Victuailles en stock", Verified = false },
new Order() { OrderID = 10252, CustomerID = "SUPRD", Freight = 51.3,
ShipName = "Suprêmes délices", Verified = false },
new Order() { OrderID = 10253, CustomerID = "HANAR", Freight = 58.17,
ShipName = "Hanari Carnes", Verified = false },
new Order() { OrderID = 10254, CustomerID = "CHOPS", Freight = 22.98,
ShipName = "Chop-suey Chinese", Verified = true },
new Order() { OrderID = 10255, CustomerID = "RICSU", Freight = 148.33,
ShipName = "Richter Supermarket", Verified = true },
new Order() { OrderID = 10256, CustomerID = "WELLI", Freight = 13.97,
ShipName = "Wellington Importadora", Verified = false },
new Order() { OrderID = 10257, CustomerID = "HILAA", Freight = 81.91,
ShipName = "HILARION-Abastos", Verified = true }
};
public class Order
{
public int? OrderID { get; set; }
```

```
public string CustomerID { get; set; }
public double? Freight { get; set; }
public string ShipName { get; set; }
public bool Verified { get; set; }
}
```

Need to define [EditType](#) as **EditType.BooleanEdit** to GridColumn to render checkbox while editing a boolean value.

ASPX-CS

```
<GridColumn Field=@nameof(Order.Verified) HeaderText="Verified"
Type="ColumnType.Boolean" EditType="EditType.BooleanEdit"
TextAlign="TextAlign.Center" DisplayAsCheckBox="true"
Width="120"></GridColumn>
```

The following image represents the DisplayAsCheckBox enabled for a DataGrid column,

Order ID	Customer ID	Freight	Ship Name	Verified
10248	VINET	32.38	Vins et alcools Chev...	<input checked="" type="checkbox"/>
10249	TOMSP	11.61	Toms Spezialitäten	<input type="checkbox"/>
10250	HANAR	65.83	Hanari Carnes	<input checked="" type="checkbox"/>
10251	VICTE	41.34	Victuailles en stock	<input type="checkbox"/>
10252	SUPRD	51.3	Suprêmes délices	<input type="checkbox"/>
10253	HANAR	58.17	Hanari Carnes	<input type="checkbox"/>
10254	CHOPS	22.98	Chop-suey Chinese	<input checked="" type="checkbox"/>
10255	RICSU	148.33	Richter Supermarket	<input checked="" type="checkbox"/>
10256	WELLI	13.97	Wellington Importa...	<input type="checkbox"/>
10257	HILAA	81.91	HILARION-Abastos	<input checked="" type="checkbox"/>

You can refer to the [Blazor DataGrid](#) feature tour page for its groundbreaking feature representations. You can also explore [Blazor DataGrid example](#) to understand how to present and manipulate data.

Row in Blazor DataGrid Component

The row represents record details fetched from data source.

Row template

Before adding row template to the datagrid, we strongly recommend you to go through the [template](#) section topic to configure the template.

To know about **Row Template** in Blazor DataGrid Component, you can check this video.

{% youtube

"youtube:https://www.youtube.com/watch?v=Dft0kerEGUQ"%}

The **RowTemplate** has an option to customize the look and behavior of the datagrid rows. The **RowTemplate** should be wrapped around a component named [GridTemplates](#) as follows. The **RowTemplate** content must be **TD** elements and the number of **TD** elements must match the number of datagrid columns.

ASPX-CS

```
@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Employees" Height="335px">
  <GridTemplates>
    <RowTemplate Context="emp">
      @{
        var employee = (emp as EmployeeData);
        <td class="photo">
          
        </td>
        <td class="details">
          <table class="CardTable" cellpadding="3" cellspacing="2">
            <colgroup>
              <col width="50%">
              <col width="50%">
            </colgroup>
            <tbody>
              <tr>
                <td class="CardHeader">First Name </td>
                <td>@employee.FirstName </td>
              </tr>
              <tr>
                <td class="CardHeader">Last Name</td>
                <td>@employee.LastName </td>
              </tr>
              <tr>
                <td class="CardHeader">
                  Title
                </td>
                <td>
                  @employee.Title
                </td>
              </tr>
              <tr>
                <td class="CardHeader">
                  Country
                </td>
                <td>
                  @employee.Country
                </td>
              </tr>
            </tbody>
          </table>
        </td>
      </td>
    </RowTemplate>
  </GridTemplates>
</GridColumns>
```



```

<GridColumn HeaderText="Employee Image" Width="250"
TextAlign="TextAlign.Center"> </GridColumn>
<GridColumn HeaderText="Employee Details" Width="300"
TextAlign="TextAlign.Left"></GridColumn>
</GridColumns>
</SfGrid>
<style type="text/css" class="cssStyles">
.photo img {
width: 100px;
height: 100px;
border-radius: 50px;
box-shadow: inset 0 0 1px #e0e0e0, inset 0 0 14px rgba(0, 0, 0, 0.2);
}
.photo,
.details {
border-color: #e0e0e0;
border-style: solid;
}
.photo {
border-width: 1px 0px 0px 0px;
text-align: center;
}
.details {
border-width: 1px 0px 0px 0px;
padding-left: 18px;
}
.e-bigger .details {
padding-left: 25px;
}
.e-device .details {
padding-left: 8px;
}
.details > table {
width: 100%;
}
.CardHeader {
font-weight: 600;
}
td {
padding: 2px 2px 3px 4px;
}
</style>
@code {
public List<EmployeeData> Employees { get; set; }
protected override void OnInitialized()
{
Employees = Enumerable.Range(1, 9).Select(x => new EmployeeData()
{
EmployeeID = x,
FirstName = (new string[] { "Nancy", "Andrew", "Janet", "Margaret", "Steven"
})[new Random().Next(5)],
LastName = (new string[] { "Davolio", "Fuller", "Leverling", "Peacock",
"Buchanan" })[new Random().Next(5)],
Title = (new string[] { "Sales Representative", "Vice President, Sales",
"Sales Manager",
"Inside Sales Coordinator" })[new Random().Next(4)],
Country = (new string[] { "USA", "UK", "UAE", "NED",





```

```

"BER" })[new Random().Next(4)],
}).ToList();
}
public class EmployeeData
{
public int? EmployeeID { get; set; }
public string FirstName { get; set; }
public string LastName { get; set; }
public string Title { get; set; }
public string Country { get; set; }
}
}

```

The output will be as follows.

Employee Image	Employee Details	
	First Name	Nancy
	Last Name	Buchanan
	Title	Vice President, Sales
	Country	USA
	First Name	Margaret
	Last Name	Buchanan
	Title	Inside Sales Coordinator
	Country	USA
	First Name	Steven
	Last Name	Buchanan
	Title	Sales Manager
	Country	UK
	First Name	Margaret

Row template with formatting

If the [RowTemplate](#) is used, the value cannot be formatted inside the template using the [Columns.Format](#) property. In that case, C# custom formats can be used.

Here [Custom DateTime](#) format is used for below sample.

ASPX-CS

```

@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Employees" Height="335px">
<GridTemplates>
<RowTemplate>
@{
var employee = (context as EmployeeData);
<td class="photo">

</td>
<td class="details">
<table class="CardTable" cellpadding="3" cellspacing="2">
<colgroup>

```

```

<col width="50%">
<col width="50%">
</colgroup>
<tbody>
<tr>
<td class="CardHeader">First Name </td>
<td>@employee.FirstName </td>
</tr>
<tr>
<td class="CardHeader">Last Name</td>
<td>@employee.LastName </td>
</tr>
<tr>
<td class="CardHeader">
Title
</td>
<td>
@employee.Title
</td>
</tr>
<tr>
<td class="CardHeader">
Birth Date
</td>
<td>
@employee.HireDate.Value.ToString("MM/dd/yyyy")
</td>
</tr>
<tr>
<td class="CardHeader">
Country
</td>
<td>
@employee.Country
</td>
</tr>
</tbody>
</table>
</td>
}
</RowTemplate>
</GridTemplates>
<GridColumns>
<GridColumn HeaderText="Employee Image" Width="250"
TextAlign="TextAlign.Center"> </GridColumn>
<GridColumn HeaderText="Employee Details" Width="300"
TextAlign="TextAlign.Left"></GridColumn>
</GridColumns>
</SfGrid>
<style type="text/css" class="cssStyles">
.photo img {
width: 100px;
height: 100px;
border-radius: 50px;
box-shadow: inset 0 0 1px #e0e0e0, inset 0 0 14px rgba(0, 0, 0, 0.2);
}
.photo,




```

```

.details {
border-color: #e0e0e0;
border-style: solid;
}
.photo {
border-width: 1px 0px 0px 0px;
text-align: center;
}
.details {
border-width: 1px 0px 0px 0px;
padding-left: 18px;
}
.e-bigger .details {
padding-left: 25px;
}
.e-device .details {
padding-left: 8px;
}
.details > table {
width: 100%;
}
.CardHeader {
font-weight: 600;
}
td {
padding: 2px 2px 3px 4px;
}
</style>
@code {
public List<EmployeeData> Employees { get; set; }
protected override void OnInitialized()
{
Employees = Enumerable.Range(1, 9).Select(x => new EmployeeData()
{
EmployeeID = x,
FirstName = (new string[] { "Nancy", "Andrew", "Janet", "Margaret", "Steven"
})[new Random().Next(5)],
LastName = (new string[] { "Davolio", "Fuller", "Leverling", "Peacock",
"Buchanan" })[new Random().Next(5)],
Title = (new string[] { "Sales Representative", "Vice President, Sales",
"Sales Manager",
"Inside Sales Coordinator" })[new Random().Next(4)],
HireDate = DateTime.Now.AddDays(-x),
Country = (new string[] { "USA", "UK", "USA", "UK", "USA" })[new
Random().Next(5)],
}).ToList();
}
public class EmployeeData
{
public int? EmployeeID { get; set; }
public string FirstName { get; set; }
public string LastName { get; set; }
public string Title { get; set; }
public DateTime? HireDate { get; set; }
public string Country { get; set; }
}
}

```

The output will be as follows.

Employee Image	Employee Details	
	First Name	Nancy
	Last Name	Fuller
	Title	Sales Manager
	Birth Date	08/08/2019
	Country	UK
	First Name	Janet
	Last Name	Peacock
	Title	Vice President, Sales
	Birth Date	08/07/2019
	Country	UK
	First Name	Andrew
	Last Name	Peacock
	Title	Sales Representative
	Birth Date	08/06/2019

Limitations

Row template feature is not compatible with all the features which are available in datagrid and it has limited features support. Here we have listed out the features which are compatible with row template feature.

- Filtering
- Paging
- Sorting
- Scrolling
- Searching
- Rtl
- Export
- Context Menu
- State Persistence

Detail Template

Before adding detail template to the datagrid, we strongly recommend you to go through the [template](#) section topic to configure the template.

To know about **Detail Template** in Blazor DataGrid Component, you can check this video.

{% youtube

"youtube:https://www.youtube.com/watch?v=Dft0kerEGUQ"%}

The detail template provides additional information about a particular row by expanding or collapsing detail content. The **DetailTemplate** should be wrapped around a component named [GridTemplates](#) as follows.

ASPX-CS

```

@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Employees">
<GridTemplates>
<DetailTemplate>
@{
var employee = (context as EmployeeData);
<table class="detailtable" width="100%">
<colgroup>
<col width="35%">
<col width="35%">
<col width="30%">
</colgroup>
<tbody>
<tr>
<td rowspan="4" style="text-align: center;">

</td>
<td>
<span style="font-weight: 500;">Employee ID: </span> @employee.FirstName
</td>
<td>
<span style="font-weight: 500;">Hire Date: </span>
@employee.HireDate.Value.ToShortDateString()
</td>
</tr>
<tr>
<td>
<span style="font-weight: 500;">Last Name: </span> @employee.LastName
</td>
<td>
<span style="font-weight: 500;">City: </span> @employee.City
</td>
</tr>
<tr>
<td>
<span style="font-weight: 500;">Title: </span> @employee.Title
</td>
<td>
<span style="font-weight: 500;">Country: </span> @employee.Country
</td>
</tr>
</tbody>
</table>
}
</DetailTemplate>
</GridTemplates>
<GridColumn>
<GridColumn Field=@nameof(EmployeeData.FirstName) HeaderText="First Name"
Width="110"> </GridColumn>
<GridColumn Field=@nameof(EmployeeData.LastName) HeaderText="Last Name"
Width="110"></GridColumn>
<GridColumn Field=@nameof(EmployeeData.Title) HeaderText="Title"
Width="110"></GridColumn>
<GridColumn Field=@nameof(EmployeeData.Country) HeaderText="Country"
Width="110"></GridColumn>



```

```

</GridColumns>
</SfGrid>
<style type="text/css" class="cssStyles">
.detailtable td {
font-size: 13px;
padding: 4px;
max-width: 0;
overflow: hidden;
text-overflow: ellipsis;
white-space: nowrap;
}
.photo {
width: 100px;
height: 100px;
border-radius: 50px;
box-shadow: inset 0 0 1px #e0e0e0, inset 0 0 14px rgba(0,0,0,0.2);
}
</style>
@code{
public List<EmployeeData> Employees { get; set; }
protected override void OnInitialized()
{
Employees = Enumerable.Range(1, 9).Select(x => new EmployeeData()
{
EmployeeID = x,
FirstName = (new string[] { "Nancy", "Andrew", "Janet", "Margaret", "Steven"
})[new Random().Next(5)],
LastName = (new string[] { "Davolio", "Fuller", "Leverling", "Peacock",
"Buchanan" })[new Random().Next(5)],
Title = (new string[] { "Sales Representative", "Vice President, Sales",
"Sales Manager",
"Inside Sales Coordinator" })[new Random().Next(4)],
HireDate = DateTime.Now.AddDays(-x),
City = (new string[] { "Seattle", "Tacoma", "Redmond", "Kirkland", "London"
})[new Random().Next(5)],
Country = (new string[] { "USA", "UK" })[new Random().Next(2)],
}).ToList();
}
public class EmployeeData
{
public int? EmployeeID { get; set; }
public string FirstName { get; set; }
public string LastName { get; set; }
public string Title { get; set; }
public DateTime? HireDate { get; set; }
public string City { get; set; }
public string Country { get; set; }
}
}

```

The output will be as follows.

First Name	Last Name	Title	Country
^ Nancy	Fuller	Inside Sales Coordinator	UK
<div>  <div> <p>Employee ID: Nancy</p> <p>Hire Date: 8/8/2019</p> <p>Last Name: Fuller</p> <p>City: Tacoma</p> <p>Title: Inside Sales Coordinator</p> <p>Country: UK</p> </div> </div>			
^ Andrew	Davolio	Vice President, Sales	USA
<div>  <div> <p>Employee ID: Andrew</p> <p>Hire Date: 8/7/2019</p> <p>Last Name: Davolio</p> <p>City: Redmond</p> <p>Title: Vice President, Sales</p> <p>Country: USA</p> </div> </div>			
v Nancy	Peacock	Sales Manager	UK
v Margaret	Fuller	Sales Manager	UK
v Andrew	Davolio	Sales Representative	UK
v Steven	Peacock	Sales Manager	UK
v Nancy	Buchanan	Vice President, Sales	UK
v Janet	Leverling	Sales Representative	USA
v Steven	Peacock	Inside Sales Coordinator	USA

Rendering custom component

To render the custom component inside the detail row, define the template in the [DetailTemplate](#) and render the custom component in any of the detail row element.

In the below sample, a datagrid component is rendered as custom component using detailed row details.

ASPX-CS

```
@using Syncfusion.Blazor.Grids
@using Syncfusion.Blazor.Data
<SfGrid DataSource="@Employees" Height="315px">
<GridTemplates>
<DetailTemplate>
@{
var employee = (context as EmployeeData);
<SfGrid DataSource="@Orders" Query="@ (new Query().Where("EmployeeID",
"equal", employee.EmployeeID)) ">
<GridColumn>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="First Name"
Width="110"> </GridColumn>
<GridColumn Field=@nameof(Order.CustomerName) HeaderText="Last Name"
Width="110"></GridColumn>
<GridColumn Field=@nameof(Order.ShipCountry) HeaderText="Title"
Width="110"></GridColumn>
</GridColumn>
</GridColumns>
```



```

</SfGrid>
}
</DetailTemplate>
</GridTemplates>
<GridColumns>
<GridColumn Field=@nameof(EmployeeData.FirstName) HeaderText="First Name"
Width="110"> </GridColumn>
<GridColumn Field=@nameof(EmployeeData.LastName) HeaderText="Last Name"
Width="110"></GridColumn>
<GridColumn Field=@nameof(EmployeeData.Title) HeaderText="Title"
Width="110"></GridColumn>
<GridColumn Field=@nameof(EmployeeData.Country) HeaderText="Country"
Width="110"></GridColumn>
</GridColumns>
</SfGrid>
@code{
List<EmployeeData> Employees = new List<EmployeeData>
{
new EmployeeData() {EmployeeID = 1001, FirstName="Nancy", LastName="Fuller",
Title="Sales Representative", Country="USA"},
new EmployeeData() {EmployeeID = 1002, FirstName="Andrew",
LastName="Davolio", Title="Vice President", Country="UK"},
new EmployeeData() {EmployeeID = 1003, FirstName="Janet",
LastName="Leverling", Title="Sales", Country="UK"},
new EmployeeData() {EmployeeID = 1004, FirstName="Margaret",
LastName="Peacock", Title="Sales Manager", Country="UAE"},
new EmployeeData() {EmployeeID = 1005, FirstName="Steven",
LastName="Buchanan", Title="Inside Sales Coordinator", Country="USA"},
new EmployeeData() {EmployeeID = 1006, FirstName="Smith", LastName="Steven",
Title="HR Manager", Country="UAE"},
};
List<Order> Orders = new List<Order> {
new Order() {EmployeeID = 1001, OrderID=001, CustomerName="Nancy",
ShipCountry="USA"},
new Order() {EmployeeID = 1001, OrderID=002, CustomerName="Steven",
ShipCountry="UR"},
new Order() {EmployeeID = 1002, OrderID=003, CustomerName="Smith",
ShipCountry="UK"},
new Order() {EmployeeID = 1002, OrderID=004, CustomerName="Smith",
ShipCountry="USA"},
new Order() {EmployeeID = 1003, OrderID=005, CustomerName="Nancy",
ShipCountry="ITA"},
new Order() {EmployeeID = 1003, OrderID=006, CustomerName="Nancy",
ShipCountry="UK"},
new Order() {EmployeeID = 1003, OrderID=007, CustomerName="Steven",
ShipCountry="GER"},
new Order() {EmployeeID = 1004, OrderID=008, CustomerName="Andrew",
ShipCountry="GER"},
new Order() {EmployeeID = 1005, OrderID=009, CustomerName="Fuller",
ShipCountry="USA"},
new Order() {EmployeeID = 1006, OrderID=010, CustomerName="Leverling",
ShipCountry="UAE"},
new Order() {EmployeeID = 1006, OrderID=011, CustomerName="Margaret",
ShipCountry="KEN"},
new Order() {EmployeeID = 1007, OrderID=012, CustomerName="Buchanan",
ShipCountry="GER"},

```

```

new Order() {EmployeeID = 1008, OrderID=013, CustomerName="Nancy",
ShipCountry="USA"},
new Order() {EmployeeID = 1006, OrderID=014, CustomerName="Andrew",
ShipCountry="UAE"},
};
public class EmployeeData
{
public int? EmployeeID { get; set; }
public string FirstName { get; set; }
public string LastName { get; set; }
public string Title { get; set; }
public string Country { get; set; }
}
public class Order
{
public int? EmployeeID { get; set; }
public int? OrderID { get; set; }
public string CustomerName { get; set; }
public string ShipCountry { get; set; }
}
}

```

The output will be as follows.

	First Name	Last Name	Title	Country
^	Nancy	Fuller	Sales Representative	USA
	EmployeeID	OrderID	CustomerName	ShipCountry
	1001	1	Nancy	USA
	1001	2	Steven	UR
✓	Andrew	Davolio	Vice President	UK
✓	Janet	Leverling	Sales	UK

[Expand by external button](#)

By default, detail rows render in collapsed state. You can expand a detail row by invoking the **Expand** method using the external button.

ASPX-CS

```

@using Syncfusion.Blazor.Buttons
@using Syncfusion.Blazor.Grids
<SfButton Content="Expand" OnClick="BtnClick"></SfButton>
<SfGrid @ref="GridObj" DataSource="@Employees">
<GridTemplates>
<DetailTemplate>
@{
var employee = (context as EmployeeData);
<table class="detailtable" width="100%">
<colgroup>
<col width="35%">
<col width="35%">

```

```

<col width="30%">
</colgroup>
<tbody>
<tr>
<td rowspan="4" style="text-align: center;">

</td>
<td>
<span style="font-weight: 500;">Employee ID: </span> @employee.FirstName
</td>
<td>
<span style="font-weight: 500;">Hire Date: </span>
@employee.HireDate.Value.ToShortDateString()
</td>
</tr>
<tr>
<td>
<span style="font-weight: 500;">Last Name: </span> @employee.LastName
</td>
<td>
<span style="font-weight: 500;">City: </span> @employee.City
</td>
</tr>
<tr>
<td>
<span style="font-weight: 500;">Title: </span> @employee.Title
</td>
<td>
<span style="font-weight: 500;">Country: </span> @employee.Country
</td>
</tr>
</tbody>
</table>
}
</DetailTemplate>
</GridTemplates>
<GridColumn>
<GridColumn Field=@nameof(EmployeeData.FirstName) HeaderText="First Name"
Width="110"> </GridColumn>
<GridColumn Field=@nameof(EmployeeData.LastName) HeaderText="Last Name"
Width="110"></GridColumn>
<GridColumn Field=@nameof(EmployeeData.Title) HeaderText="Title"
Width="110"></GridColumn>
<GridColumn Field=@nameof(EmployeeData.Country) HeaderText="Country"
Width="110"></GridColumn>
</GridColumn>
</SfGrid>
<style type="text/css" class="cssStyles">
.detailtable td {
font-size: 13px;
padding: 4px;
max-width: 0;
overflow: hidden;
text-overflow: ellipsis;
white-space: nowrap;

```

```

}
.photo {
width: 100px;
height: 100px;
border-radius: 50px;
box-shadow: inset 0 0 1px #e0e0e0, inset 0 0 14px rgba(0,0,0,0.2);
}
</style>
@code{
public SfGrid<EmployeeData> GridObj;
public List<EmployeeData> Employees { get; set; }
protected override void OnInitialized()
{
Employees = Enumerable.Range(1, 9).Select(x => new EmployeeData()
{
EmployeeID = x,
FirstName = (new string[] { "Nancy", "Andrew", "Janet", "Margaret", "Steven"
})[new Random().Next(5)],
LastName = (new string[] { "Davolio", "Fuller", "Leverling", "Peacock",
"Buchanan" })[new Random().Next(5)],
Title = (new string[] { "Sales Representative", "Vice President, Sales",
"Sales Manager",
"Inside Sales Coordinator" })[new Random().Next(4)],
HireDate = DateTime.Now.AddDays(-x),
City = (new string[] { "Seattle", "Tacoma", "Redmond", "Kirkland", "London"
})[new Random().Next(5)],
Country = (new string[] { "USA", "UK" })[new Random().Next(2)],
}).ToList();
}
public class EmployeeData
{
public int? EmployeeID { get; set; }
public string FirstName { get; set; }
public string LastName { get; set; }
public string Title { get; set; }
public DateTime? HireDate { get; set; }
public string City { get; set; }
public string Country { get; set; }
}
public void BtnClick()
{
this.GridObj.DetailExpandAll();
}
}

```

* You can expand all the rows by using **ExpandAll** method.

* If you want to expand all the rows at initial DataGrid rendering, then use **ExpandAll** method in [dataBound](#) event of the DataGrid.

<!-- Row spanning

The datagrid has option to span row cells. To achieve this, You need to define the **RowSpan** attribute to span cells in the [QueryCellInfo](#) event.

In the following demo, "Davolio" cell is spanned to two rows in the **EmployeeName** column.

Also DataGrid supports the spanning of rows and columns for same cells. **Lunch Break** cell is spanned to two rows and three columns in the **1:00** column.

ASPX-CS

```
@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@columnSpanData"
GridLines="Syncfusion.Blazor.Grids.GridLine.Both">
<GridEvents TValue="SpanData" QueryCellInfo="QueryCellEvent"></GridEvents>
<GridColumns>
<GridColumn Field=@nameof(SpanData.EmployeeID) HeaderText="First Name"
Width="110"> </GridColumn>
<GridColumn Field=@nameof(SpanData.EmployeeName) HeaderText="Last Name"
Width="110"></GridColumn>
<GridColumn Field=@nameof(SpanData.Time900) HeaderText="9:00"
Width="110"></GridColumn>
<GridColumn Field=@nameof(SpanData.Time930) HeaderText="9:30"
Width="110"></GridColumn>
<GridColumn Field=@nameof(SpanData.Time1000) HeaderText="10:00"
Width="110"></GridColumn>
<GridColumn Field=@nameof(SpanData.Time1030) HeaderText="10:30"
Width="110"></GridColumn>
<GridColumn Field=@nameof(SpanData.Time1100) HeaderText="11:00"
Width="110"></GridColumn>
<GridColumn Field=@nameof(SpanData.Time1130) HeaderText="11:30"
Width="110"></GridColumn>
<GridColumn Field=@nameof(SpanData.Time1200) HeaderText="12:00"
Width="110"></GridColumn>
<GridColumn Field=@nameof(SpanData.Time1230) HeaderText="12:30"
Width="110"></GridColumn>
<GridColumn Field=@nameof(SpanData.Time100) HeaderText="1:00"
Width="110"></GridColumn>
<GridColumn Field=@nameof(SpanData.Time130) HeaderText="1:30"
Width="110"></GridColumn>
<GridColumn Field=@nameof(SpanData.Time200) HeaderText="2:00"
Width="110"></GridColumn>
<GridColumn Field=@nameof(SpanData.Time230) HeaderText="2:30"
Width="110"></GridColumn>
<GridColumn Field=@nameof(SpanData.Time300) HeaderText="3:00"
Width="110"></GridColumn>
<GridColumn Field=@nameof(SpanData.Time330) HeaderText="3:30"
Width="110"></GridColumn>
<GridColumn Field=@nameof(SpanData.Time400) HeaderText="4:00"
Width="110"></GridColumn>
<GridColumn Field=@nameof(SpanData.Time430) HeaderText="4:30"
Width="110"></GridColumn>
<GridColumn Field=@nameof(SpanData.Time500) HeaderText="5:00"
Width="110"></GridColumn>
</GridColumns>
</SfGrid>
@code {
List<SpanData> columnSpanData = new List<SpanData>
{
new SpanData() { EmployeeID = 10001 , EmployeeName = "Davolio", Time900 =
"Analysis Tasks" , Time930 = "Analysis Tasks", Time1000 = "Team Meeting",
Time1030 = "Testing", Time1100 = "Development", Time1130 = "Development",
```

```

Time1200 = "Development", Time1230 = "Support", Time100 = "Lunch Break",
Time130 = "Lunch Break", Time200 = "Lunch Break" , Time230 = "Testing",
Time300 = "Testing", Time330 = "Development", Time400 = "Conference",
Time430 = "Team Meeting", Time500 = "Team Meeting" },
new SpanData() { EmployeeID = 10002 , EmployeeName = "Buchanan", Time900 =
"Task Assign" , Time930 = "Support", Time1000 = "Support", Time1030 =
"Support", Time1100 = "Testing", Time1130 = "Testing", Time1200 = "Testing",
Time1230 = "Testing", Time100 = "Lunch Break", Time130 = "Lunch Break",
Time200 = "Lunch Break" , Time230 = "Development",
Time300 = "Development", Time330 = "Check Mail", Time400 = "'Check Mail",
Time430 = "Team Meeting", Time500 = "Team Meeting" },
new SpanData() { EmployeeID = 10003 , EmployeeName = "Fuller", Time900 =
"Check Mail" , Time930 = "Check Mail", Time1000 = "Check Mail", Time1030 =
"Analysis Tasks", Time1100 = "Analysis Tasks", Time1130 = "Support",
Time1200 = "Support", Time1230 = "Support", Time100 = "Lunch Break",
Time130 = "Lunch Break", Time200 = "Lunch Break" , Time230 = "Development",
Time300 = "Development", Time330 = "Team Meeting", Time400 = "Team Meeting",
Time430 = "Development", Time500 = "Development" },
new SpanData() { EmployeeID = 10004 , EmployeeName = "Leverling", Time900 =
"Testing" , Time930 = "Check Mail", Time1000 = "Check Mail", Time1030 =
"Support", Time1100 = "Testing", Time1130 = "Testing", Time1200 = "Testing",
Time1230 = "Testing", Time100 = "Lunch Break", Time130 = "Lunch Break",
Time200 = "Lunch Break" , Time230 = "Development",
Time300 = "Development", Time330 = "Check Mail", Time400 = "Conference",
Time430 = "Conference", Time500 = "Team Meeting" },
new SpanData() { EmployeeID = 10005 , EmployeeName = "Peacock", Time900 =
"Task Assign" , Time930 = "Task Assign", Time1000 = "Task Assign", Time1030 =
"Task Assign", Time1100 = "Check Mail", Time1130 = "Support", Time1200 =
"Support", Time1230 = "Support", Time100 = "Lunch Break", Time130 = "Lunch
Break", Time200 = "Lunch Break" , Time230 = "Development",
Time300 = "Development", Time330 = "Team Meeting", Time400 = "Team Meeting",
Time430 = "Testing", Time500 = "Testing" },
new SpanData() { EmployeeID = 10006 , EmployeeName = "Janet", Time900 =
"Testing" , Time930 = "Testing", Time1000 = "Support", Time1030 = "Support",
Time1100 = "Support", Time1130 = "Team Meeting", Time1200 = "Team Meeting",
Time1230 = "Team Meeting", Time100 = "Lunch Break", Time130 = "Lunch
Break", Time200 = "Lunch Break" , Time230 = "Development",
Time300 = "Development", Time330 = "Team Meeting", Time400 = "Team Meeting",
Time430 = "Development", Time500 = "Development" },
new SpanData() { EmployeeID = 10007 , EmployeeName = "Suyama", Time900 =
"Analysis Tasks" , Time930 = "Analysis Tasks", Time1000 = "Testing",
Time1030 = "Development", Time1100 = "Development", Time1130 = "Testing",
Time1200 = "Testing", Time1230 = "Testing", Time100 = "Lunch Break",
Time130 = "Lunch Break", Time200 = "Lunch Break" , Time230 = "Support",
Time300 = "Build", Time330 = "Build", Time400 = "Check Mail", Time430 =
"Check Mail", Time500 = "Check Mail" },
};
public class SpanData
{
public int EmployeeID { get; set; }
public string EmployeeName { get; set; }
public string Time900 { get; set; }
public string Time930 { get; set; }
public string Time1000 { get; set; }
public string Time1030 { get; set; }
public string Time1100 { get; set; }
public string Time1130 { get; set; }

```

```
public string Time1200 { get; set; }
public string Time1230 { get; set; }
public string Time100 { get; set; }
public string Time130 { get; set; }
public string Time200 { get; set; }
public string Time230 { get; set; }
public string Time300 { get; set; }
public string Time330 { get; set; }
public string Time400 { get; set; }
public string Time430 { get; set; }
public string Time500 { get; set; }
}

public void
QueryCellEvent(Syncfusion.Blazor.Grids.QueryCellInfoEventArgs<SpanData>
args)
{
var DataSpan = args.Data as SpanData;
switch (DataSpan.EmployeeID)
{
case 10001:
if (args.Column.Field == "Time900" || args.Column.Field == "Time230" ||
args.Column.Field == "Time430")
{
args.Cell.SetAttribute("ColSpan", 2);
}
else if (args.Column.Field == "Time1100")
{
args.Cell.SetAttribute("ColSpan", 2);
}
else if (args.Column.Field == "EmployeeName")
{
args.Cell.SetAttribute("RowSpan", 2);
}
else if (args.Column.Field == "Time100")
{
args.Cell.SetAttribute("ColSpan", 3);
args.Cell.SetAttribute("RowSpan", 2);
}
break;
case 10002:
if (args.Column.Field == "Time930" || args.Column.Field == "Time230" ||
args.Column.Field == "Time430")
{
args.Cell.SetAttribute("ColSpan", 3);
}
else if (args.Column.Field == "Time1100")
{
args.Cell.SetAttribute("ColSpan", 4);
}
break;
case 10003:
if (args.Column.Field == "Time900" || args.Column.Field == "Time1130")
{
args.Cell.SetAttribute("ColSpan", 3);
}
else if (args.Column.Field == "Time1030" || args.Column.Field == "Time330"
||
```

```
args.Column.Field == "Time430" || args.Column.Field == "Time230")
{
    args.Cell.SetAttribute("ColSpan", 2);
}
break;
case 10004:
if (args.Column.Field == "Time900")
{
    args.Cell.SetAttribute("ColSpan", 3);
}
else if (args.Column.Field == "Time1100")
{
    args.Cell.SetAttribute("ColSpan", 4);
}
else if (args.Column.Field == "Time400" || args.Column.Field == "Time230")
{
    args.Cell.SetAttribute("ColSpan", 2);
}
break;
case 10005:
if (args.Column.Field == "Time900")
{
    args.Cell.SetAttribute("ColSpan", 4);
}
else if (args.Column.Field == "Time1130")
{
    args.Cell.SetAttribute("ColSpan", 3);
}
else if (args.Column.Field == "Time330" || args.Column.Field == "Time430" ||
args.Column.Field == "Time230")
{
    args.Cell.SetAttribute("ColSpan", 2);
}
break;
case 10006:
if (args.Column.Field == "Time900" || args.Column.Field == "Time430" ||
args.Column.Field == "Time230" || args.Column.Field == "Time330")
{
    args.Cell.SetAttribute("ColSpan", 2);
}
else if (args.Column.Field == "Time1000" || args.Column.Field == "Time1130")
{
    args.Cell.SetAttribute("ColSpan", 2);
}
break;
case 10007:
if (args.Column.Field == "Time900" || args.Column.Field == "Time300" ||
args.Column.Field == "Time1030")
{
    args.Cell.SetAttribute("ColSpan", 2);
}
else if (args.Column.Field == "Time1130" || args.Column.Field == "Time400")
{
    args.Cell.SetAttribute("ColSpan", 3);
}
break;
}
```



```
}
}
```

When paging is enabled in datagrid, you can disable the rows and columns spanning for any particular page. To achieve this, we need to check **RequestType** as paging `<code class='language-text'>string</code>` in [QueryCellInfo](#) event of datagrid. -->

Customize rows

You can customize the appearance of a row by using the [RowDataBound](#) event. The [RowDataBound](#) event triggers for every row. In the event handler, you can get the **RowDataBoundEventArgs** that contains details of the row.

ASPX-CS

```
@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders" EnableHover=false AllowSelection=false
Height="280">
  <GridEvents TValue="Order" RowDataBound="RowBound"></GridEvents>
  <GridColumns>
    <GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID" Width="110">
    </GridColumn>
    <GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer ID"
Width="110"></GridColumn>
    <GridColumn Field=@nameof(Order.ShipCity) HeaderText="Ship City"
Width="110"></GridColumn>
    <GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
Width="110"></GridColumn>
    <GridColumn Field=@nameof(Order.OrderDate) HeaderText="Order Date"
Format="d" Width="110" Type="ColumnType.Date"></GridColumn>
  </GridColumns>
</SfGrid>
<style>
  .below-25 {
    background-color: orangered;
  }
  .below-35 {
    background-color: yellow;
  }
  .above-35 {
    background-color: greenyellow
  }
</style>
@code {
  public List<Order> Orders { get; set; }
  protected override void OnInitialized()
  {
    Orders = Enumerable.Range(1, 75).Select(x => new Order()
    {
      OrderID = 1000 + x,
      CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
    })[new Random().Next(5)],
      ShipCity = (new string[] { "Berlin", "Madrid", "Cholchester", "Marseille",
      "Tsawassen" })[new Random().Next(5)],
      Freight = 2.1 * x,
      OrderDate = DateTime.Now.AddDays(-x),
```

```

}).ToList();
}
public class Order
{
    public int? OrderID { get; set; }
    public string CustomerID { get; set; }
    public string ShipCity { get; set; }
    public DateTime? OrderDate { get; set; }
    public double? Freight { get; set; }
}
public void RowBound(RowDataBoundEventArgs<Order> args)
{
    if (args.Data.Freight < 25)
    {
        args.Row.AddClass(new string[] { "below-25" });
    }
    else if (args.Data.Freight < 35)
    {
        args.Row.AddClass(new string[] { "below-35" });
    }
    else
    {
        args.Row.AddClass(new string[] { "above-35" });
    }
}
}

```

The output will be as follows.

OrderID	CustomerID	ShipCity	Freight	OrderDate
1011	BOLID	Cholchester	\$23.10	7/29/2019
1012	ALFKI	Tsawassen	\$25.20	7/28/2019
1013	ANTON	Madrid	\$27.30	7/27/2019
1014	ANTON	Marseille	\$29.40	7/26/2019
1015	BLONP	Tsawassen	\$31.50	7/25/2019
1016	ANANTR	Berlin	\$33.60	7/24/2019
1017	ANANTR	Berlin	\$35.70	7/23/2019
1018	ALFKI	Marseille	\$37.80	7/22/2019

Styling alternate rows

You can change the datagrid's alternative rows background color by overriding the **.e-altrow** class.

CSS

```

.e-grid .e-altrow {
    background-color: #fafafa;
}

```

Please refer to the following example.

ASPX-CS

```

@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders" Height="280">
<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID" Width="110">
</GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer ID"
Width="110"></GridColumn>
<GridColumn Field=@nameof(Order.ShipCity) HeaderText="Ship City"
Width="110"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
Width="110"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText="Order Date"
Format="d" Width="110" Type="ColumnType.Date"></GridColumn>
</GridColumns>
</SfGrid>
<style type="text/css" class="cssStyles">
.e-grid .e-altrow {
background-color: #fafafa;
}
</style>
@code {
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
ShipCity = (new string[] { "Berlin", "Madrid", "Cholchester", "Marseille",
"Tsawassen" })[new Random().Next(5)],
Freight = 2.1 * x,
OrderDate = DateTime.Now.AddDays(-x),
}).ToList();
}
public class Order
{
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public string ShipCity { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
}

```

The output will be as follows.

OrderID	CustomerID	ShipCity	Freight	OrderDate
1001	ANTON	Marseille	\$2.10	8/8/2019
1002	ANANTR	Madrid	\$4.20	8/7/2019
1003	ANTON	Madrid	\$6.30	8/6/2019
1004	ALFKI	Berlin	\$8.40	8/5/2019
1005	ALFKI	Madrid	\$10.50	8/4/2019
1006	ANTON	Tsawassen	\$12.60	8/3/2019
1007	BLONP	Berlin	\$14.70	8/2/2019
1008	ANANTR	Cholchester	\$16.80	8/1/2019

Row height

You can customize the row height of datagrid rows through the [RowHeight](#) property. The [RowHeight](#) property is used to change the row height of entire datagrid rows.

In the below example, the `RowHeight` is set as '60'.

ASPX-CS

```
@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders" Height="280" RowHeight="60">
  <GridColumn>
    <GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID" Width="110">
    </GridColumn>
    <GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer ID"
    Width="110"></GridColumn>
    <GridColumn Field=@nameof(Order.ShipCity) HeaderText="Ship City"
    Width="110"></GridColumn>
    <GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
    Width="110"></GridColumn>
    <GridColumn Field=@nameof(Order.OrderDate) HeaderText="Order Date"
    Format="d" Width="110" Type="ColumnType.Date"></GridColumn>
  </GridColumn>
</SfGrid>
@code {
  public List<Order> Orders { get; set; }
  protected override void OnInitialized()
  {
    Orders = Enumerable.Range(1, 75).Select(x => new Order()
    {
      OrderID = 1000 + x,
      CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
    })[new Random().Next(5)],
      ShipCity = (new string[] { "Berlin", "Madrid", "Cholchester", "Marseille",
      "Tsawassen" })[new Random().Next(5)],
      Freight = 2.1 * x,
      OrderDate = DateTime.Now.AddDays(-x),
    }).ToList();
  }
}
public class Order
```

```
{
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public string ShipCity { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
}
```

The output will be as follows.

OrderID	CustomerID	ShipCity	Freight	OrderDate
1001	BOLID	Cholchester	\$2.10	8/8/2019
1002	BOLID	Marseille	\$4.20	8/7/2019
1003	ANANTR	Tsawassen	\$6.30	8/6/2019
1004	ALFKI	Berlin	\$8.40	8/5/2019
1005	ANANTR	Cholchester	\$10.50	8/4/2019

Customize row height for particular row

DataGrid row height for particular row can be customized using the [RowDataBound](#) event by setting the height by adding `row-height` class in required row element.

In the below example, the row height for the row with OrderID as '1003' is set as '90px' using the [RowDataBound](#) event.

ASPX-CS

```
@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders" Height="280">
  <GridEvents TValue="Order" RowDataBound="RowBound"></GridEvents>
  <GridColumns>
    <GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID" Width="110">
    </GridColumn>
    <GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer ID"
    Width="110"></GridColumn>
    <GridColumn Field=@nameof(Order.ShipCity) HeaderText="Ship City"
    Width="110"></GridColumn>
    <GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
    Width="110"></GridColumn>
    <GridColumn Field=@nameof(Order.OrderDate) HeaderText="Order Date"
    Format="d" Width="110" Type="ColumnType.Date"></GridColumn>
  </GridColumns>
</SfGrid>
<style>
```

```

.row-height {
height: 90px;
}
</style>
@code {
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
ShipCity = (new string[] { "Berlin", "Madrid", "Cholchester", "Marseille",
"Tsawassen" })[new Random().Next(5)],
Freight = 2.1 * x,
OrderDate = DateTime.Now.AddDays(-x),
}).ToList();
}
public class Order
{
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public string ShipCity { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
public void RowBound(RowDataBoundEventArgs<Order> args)
{
if (args.Data.OrderID == 1003)
{
args.Row.AddClass(new string[] { "row-height" });
}
}
}

```

The output will be as follows.

OrderID	CustomerID	ShipCity	Freight	OrderDate
1001	ALFKI	Cholchester	\$2.10	8/8/2019
1002	ANTON	Tsawassen	\$4.20	8/7/2019
1003	ANANTR	Marseille	\$6.30	8/6/2019
1004	ANANTR	Marseille	\$8.40	8/5/2019
1005	ANTON	Marseille	\$10.50	8/4/2019
1006	BLONP	Tsawassen	\$12.60	8/3/2019
1007	ALFKI	Tsawassen	\$14.70	8/2/2019

You can refer to our [Blazor DataGrid](#) feature tour page for its groundbreaking feature representations. You can also explore our [Blazor DataGrid example](#) to understand how to present and manipulate data.

Cell in Blazor DataGrid Component

Displaying the HTML content

The HTML tags can be displayed in the DataGrid header and content by enabling the [DisableHtmlEncode](#) property.

ASPX-CS

```
@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders" Height="315">
  <GridColumn>
    <GridColumn Field=@nameof(Order.OrderID) HeaderText="<span> Order ID
    </span>" DisableHtmlEncode="false" TextAlign="TextAlign.Right"
    Width="140"></GridColumn>
    <GridColumn Field=@nameof(Order.CustomerID) HeaderText="<span> Customer ID
    </span>" DisableHtmlEncode="true" Width="120"></GridColumn>
    <GridColumn Field=@nameof(Order.OrderDate) HeaderText="Order Date"
    Format="d" Type="ColumnType.Date" Width="100"></GridColumn>
    <GridColumn Field=@nameof(Order.ShipCity) HeaderText="Ship City"
    Width="100"></GridColumn>
  </GridColumn>
</SfGrid>
@code{
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
  Orders = Enumerable.Range(1, 75).Select(x => new Order()
  {
    OrderID = 1000 + x,
    CustomerID = (new string[] { "ALFKI", "<span>ANANTR</span>", "ANTON",
    "BLONP", "BOLID" })[new Random().Next(5)],
    ShipCity = (new string[] { "Seattle", "Tacoma", "Redmond", "Kirkland",
    "London" })[new Random().Next(5)],
    OrderDate = DateTime.Now.AddDays(-x),
  }).ToList();
}
public class Order
{
  public int? OrderID { get; set; }
  public string CustomerID { get; set; }
  public DateTime? OrderDate { get; set; }
  public string ShipCity { get; set; }
}
```

The following screenshot represents a DataGrid displaying the HTML content.

Order ID	 Customer ID </sp...	Order Date	Ship City
1001	BLONP	8/8/2019	Seattle
1002	ALFKI	8/7/2019	Seattle
1003	ANANTR	8/6/2019	Seattle
1004	BOLID	8/5/2019	Seattle
1005	ALFKI	8/4/2019	Kirkland
1006	ANTON	8/3/2019	Redmond
1007	ALFKI	8/2/2019	Kirkland
1008	ANANTR	8/1/2019	London
1009	ANANTR	7/31/2019	London

Customize cell styles

The appearance of cells can be customized by using the [QueryCellInfo](#) event. The [QueryCellInfo](#) event triggers for every cell. In that event handler, you can get [QueryCellInfoEventArgs](#) that contains the details of the cell.

ASPX-CS

```
@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders" AllowSelection="false" EnableHover="false"
Height="315">
<GridEvents QueryCellInfo="CustomizeCell" TValue="Order"></GridEvents>
<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
TextAlign="TextAlign.Right" Width="140"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer ID"
Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText="Order Date"
Format="d" Type="ColumnType.Date" Width="140"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
Width="100"></GridColumn>
<GridColumn Field=@nameof(Order.ShipCity) HeaderText="Ship City"
Width="100"></GridColumn>
</GridColumns>
</SfGrid>
@code{
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
```



```
ShipCity = (new string[] { "Seattle", "Tacoma", "Redmond", "Kirkland",  
"London" })[new Random().Next(5)],  
OrderDate = DateTime.Now.AddDays(-x),  
Freight = 6.2 * x,  
}).ToList();  
}  
public class Order  
{  
    public int? OrderID { get; set; }  
    public string CustomerID { get; set; }  
    public DateTime? OrderDate { get; set; }  
    public double? Freight { get; set; }  
    public string ShipCity { get; set; }  
}  
public void CustomizeCell(QueryCellInfoEventArgs<Order> args)  
{  
    if (args.Column.Field == "Freight")  
    {  
        if (args.Data.Freight < 30)  
        {  
            args.Cell.AddClass(new string[] { "below-30" });  
        }  
        else if (args.Data.Freight < 80)  
        {  
            args.Cell.AddClass(new string[] { "below-80" });  
        }  
        else  
        {  
            args.Cell.AddClass(new string[] { "above-80" });  
        }  
    }  
}  
  
<style>  
    .below-30 {  
        background-color: orangered;  
    }  
    .below-80 {  
        background-color: yellow;  
    }  
    .above-80 {  
        background-color: greenyellow  
    }  
</style>
```

The following screenshot represents a DataGrid with customized cell styles.

Order ID	Customer ID	Order Date	Ship City	Freight
1001	ALFKI	8/8/2019	London	\$6.10
1002	ANANTR	8/7/2019	Tacoma	\$12.20
1003	ANTON	8/6/2019	Kirkland	\$18.30
1004	BLONP	8/5/2019	Seattle	\$24.40
1005	ANANTR	8/4/2019	Kirkland	\$30.50
1006	ANTON	8/3/2019	Kirkland	\$36.60
1007	BLONP	8/2/2019	Tacoma	\$42.70
1008	ALFKI	8/1/2019	Seattle	\$48.80
1009	BOLID	7/31/2019	Redmond	\$54.90

Auto wrap

The auto wrap allows the cell content of the DataGrid to wrap to the next line when it exceeds the boundary of the cell width. The Cell Content wrapping works based on the position of white space between words. To enable auto wrap, set the [AllowTextWrap](#) property to `true`. You can configure the auto wrap mode by setting the [TextWrapSettings.WrapMode](#) property.

There are three types of [WrapMode](#). They are:

- **Both:** **Both** value is set by default. Auto wrap will be enabled for both the content and the header.
- **Header:** Auto wrap will be enabled only for the header.
- **Content:** Auto wrap will be enabled only for the content.

When a column width is not specified, then auto wrap of columns will be adjusted with respect to the DataGrid's width.

In the following example, the [TextWrapSettings.WrapMode](#) is set to **Content**.

ASPX-CS

```
@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders" GridLines="GridLine.Default"
AllowTextWrap="true" Height="315">
<GridTextWrapSettings WrapMode="WrapMode.Content"></GridTextWrapSettings>
<GridColumns>
```

```

<GridColumn Field=@nameof(Order.RollNo) HeaderText="Roll No"
Width="140"></GridColumn>
<GridColumn Field=@nameof(Order.Name) HeaderText="Name of the inventor"
Width="70"></GridColumn>
<GridColumn Field=@nameof(Order.PatentFamilies) HeaderText="No of
patentfamilies" Width="80"></GridColumn>
<GridColumn Field=@nameof(Order.Country) HeaderText="Country"
Width="100"></GridColumn>
<GridColumn Field=@nameof(Order.MainFields) HeaderText="Main fields of
Invention" Width="120"></GridColumn>
</GridColumn>
</SfGrid>
@code{
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
RollNo = 1000 + x,
Name = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID" })[new
Random().Next(5)],
PatentFamilies = 1000 + x * 5,
Country = (new string[] { "Australia", "Japan", "Canada", "India", "USA"
})[new Random().Next(5)],
MainFields = (new string[] { "Printing, Digital paper, Internet,
Electronics, Lab-on-a-chip, MEMS, Mechanical, VLSI", "Various", "Printing,
Digital paper, Internet, Electronics, CGI, VLSI", "Automotive, Stainless
steel products", "Gaming machines" })[new Random().Next(5)],
}).ToList();
}
public class Order
{
public int? RollNo { get; set; }
public string Name { get; set; }
public int? PatentFamilies { get; set; }
public string Country { get; set; }
public string MainFields { get; set; }
}
}

```

The following screenshot represents a DataGrid with auto wrap.

Roll No	Name of t...	No of paten...	Country	Main fields of Invent...
1001	ANTON	1005	Australia	Printing, Digital paper, Internet, Electronics, CGI, VLSI
1002	BOLID	1010	India	Printing, Digital paper, Internet, Electronics, CGI, VLSI
1003	BLONP	1015	Japan	Gaming machines
1004	BLONP	1020	USA	Printing, Digital paper, Internet, Electronics, CGI, VLSI
1005	ANTON	1025	Australia	Gaming machines
1006	ANANTR	1030	India	Various

Custom Attributes

You can customize the DataGrid cells by adding a CSS class to the [CustomAttributes](#) property of the column.

In the following example, the cells of the **OrderID** and **ShipCity** columns are customized.

ASPX-CS

```
@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders" Height="315">
  <GridColumn>
    <GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
      CustomAttributes="@ (new Dictionary<string, object>() { { "class", "e-attr"
      }}})" TextAlign="TextAlign.Right" Width="140"></GridColumn>
    <GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer ID"
      Width="120"></GridColumn>
    <GridColumn Field=@nameof(Order.ShipCity) HeaderText="Ship City"
      CustomAttributes="@ (new Dictionary<string, object>() { { "class", "e-attr"
      }}})" Width="100"></GridColumn>
    <GridColumn Field=@nameof(Order.OrderDate) HeaderText="Order Date"
      Format="d" Type="ColumnType.Date" Width="100"></GridColumn>
  </GridColumn>
</SfGrid>
@code{
  public List<Order> Orders { get; set; }
  protected override void OnInitialized()
  {
    Orders = Enumerable.Range(1, 75).Select(x => new Order()
    {
      OrderID = 1000 + x,
      CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
      })[new Random().Next(5)],
```

```

ShipCity = (new string[] { "Seattle", "Tacoma", "Redmond", "Kirkland",
"London" })[new Random().Next(5)],
OrderDate = DateTime.Now.AddDays(-x),
}).ToList();
}
public class Order
{
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public string ShipCity { get; set; }
}
}
<style>
.e-attr {
background: #d7f0f4;
}
</style>

```

DataGrid Lines

The [GridLines](#) have option to display cell border and it can be defined by the [GridLines](#) property.

The available modes of DataGrid lines are:

- | Modes | Actions |
- |-----|-----|
- | Both | Displays both the horizontal and vertical DataGrid lines. |
- | None | No DataGrid lines are displayed. |
- | Horizontal | Displays the horizontal DataGrid lines only. |
- | Vertical | Displays the vertical DataGrid lines only. |
- | Default | Displays DataGrid lines based on the theme. |

ASPX-CS

```

@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders" GridLines="GridLine.Both" Height="315">
<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
TextAlign="TextAlign.Right" Width="140"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer ID"
Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.ShipCity) HeaderText="Ship City"
Width="100"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText="Order Date"
Format="d" Type="ColumnType.Date" Width="100"></GridColumn>
</GridColumns>
</SfGrid>
@code{
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()

```

```

{
    OrderID = 1000 + x,
    CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
    })[new Random().Next(5)],
    ShipCity = (new string[] { "Seattle", "Tacoma", "Redmond", "Kirkland",
    "London" })[new Random().Next(5)],
    OrderDate = DateTime.Now.AddDays(-x),
    }).ToList();
}
public class Order
{
    public int? OrderID { get; set; }
    public string CustomerID { get; set; }
    public string ShipCity { get; set; }
    public DateTime? OrderDate { get; set; }
}
}

```

By default, the DataGrid renders with **Default** mode.

Clip Mode

The clip mode provides options to display its overflow cell content and it can be defined by the [Columns.ClipMode](#) property.

There are three types of [ClipMode](#). They are:

- **Clip**: Truncates the cell content when it overflows its area.
- **Ellipsis**: Displays ellipsis when the cell content overflows its area.
- **EllipsisWithTooltip**: Displays ellipsis when the cell content overflows its area, also it will display the tooltip while hover on ellipsis is applied.

By default, [Columns.ClipMode](#) value is **Ellipsis**.

ASPX-CS

```

@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders" AllowPaging="true" Height="315">
  <GridColumns>
    <GridColumn Field=@nameof(Order.RollNo) HeaderText="Roll No"
    Width="140"></GridColumn>
    <GridColumn Field=@nameof(Order.Name) HeaderText="Name of the inventor"
    ClipMode="ClipMode.Clip" Width="70"></GridColumn>
    <GridColumn Field=@nameof(Order.PatentFamilies) HeaderText="No of patent
    families" ClipMode="ClipMode.Ellipsis" Width="80"></GridColumn>
    <GridColumn Field=@nameof(Order.Country) HeaderText="Country"
    Width="100"></GridColumn>
    <GridColumn Field=@nameof(Order.MainFields) HeaderText="Main fields of
    Invention" ClipMode="ClipMode.EllipsisWithTooltip" Width="120"></GridColumn>
  </GridColumns>
</SfGrid>
@code{
    public List<Order> Orders { get; set; }
    protected override void OnInitialized()
    {
        Orders = Enumerable.Range(1, 75).Select(x => new Order())
    }
}

```

```

{
    RollNo = 1000 + x,
    Name = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID" })[new
    Random().Next(5)],
    PatentFamilies = 1000 + x * 5,
    Country = (new string[] { "Australia", "Japan", "Canada", "India", "USA"
    })[new Random().Next(5)],
    MainFields = (new string[] { "Printing, Digital paper, Internet,
    Electronics, Lab-on-a-chip, MEMS, Mechanical, VLSI", "Various", "Printing,
    Digital paper, Internet, Electronics, CGI, VLSI", "Automotive, Stainless
    steel products", "Gaming machines" })[new Random().Next(5)],
    }).ToList();
}
}
public class Order
{
    public int? RollNo { get; set; }
    public string Name { get; set; }
    public int? PatentFamilies { get; set; }
    public string Country { get; set; }
    public string MainFields { get; set; }
}
}

```

The following screenshot represents a clip mode in DataGrid

Roll No	Name of	No of ...	Country	Main fields of Invention
1001	BLONP	1005	Japan	Printing, Digital paper, In...
1002	ANANTR	1010	C	Printing, Digital paper, Internet, Electronics, Lab-on-a-chip, MEMS, Mechanical, VLSI
1003	ANANTR	1015	U	
1004	ANANTR	1020	USA	Printing, Digital paper, In...
1005	ANANTR	1025	Japan	Printing, Digital paper, In...
1006	BOLID	1030	India	Automotive, Stainless ste...
1007	ANANTR	1035	USA	Various
1008	ANANTR	1040	USA	Various
1009	BOLID	1045	USA	Gaming machines

K < 1 2 3 4 5 6 7 > X
 1 of 7 pages (75 items)

You can refer to the [Blazor DataGrid](#) feature tour page for its groundbreaking feature representations. You can also explore [Blazor DataGrid example](#) to understand how to present and manipulate data.

Templates in Blazor DataGrid Component

Blazor has templated components which accepts one or more UI segments as input that can be rendered as part of the component during component rendering. DataGrid is a templated blazor component, that allow you to customize various part of the UI using template parameters. It allow you to render custom components or content based on your own logic.

The available template options in datagrid are as follows,

- [Column template](#) - Used to customize cell content.
- [Header template](#) - Used to customize header cell content.
- [Row template](#) - Used to customize row content.
- [Detail template](#) - Used to customize the detail cell content.

Template context

Most of the templates used by datagrid are of type **RenderFragment** and they will be passed with parameters. You can access the parameters passed to the templates using implicit parameter named **context**. You can also change this implicit parameter name using **Context** attribute.

For example, you can access the data of the column template using **context** as follows.

CSHARP

```
@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Employees">
<GridColumns>
<GridColumn HeaderText="Employee Image" TextAlign="TextAlign.Center"
Width="120">
<Template>
@{
var employee = (context as EmployeeData);
<div class="image">

</div>
}
</Template>
</GridColumn>
<GridColumn Field=@nameof(EmployeeData.EmployeeID) HeaderText="Employee ID"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(EmployeeData.FirstName) HeaderText="First Name"
Width="130"></GridColumn>
<GridColumn Field=@nameof(EmployeeData.Title) HeaderText="Title" Format="C2"
Width="120"></GridColumn>
<GridColumn Field=@nameof(EmployeeData.HireDate) HeaderText="Hire Date"
Format="d" TextAlign="TextAlign.Right" Width="150"
Type="ColumnType.Date"></GridColumn>
</GridColumns>
</SfGrid>
<style>
.image img {
height: 55px;
width: 55px;
border-radius: 50px;
box-shadow: inset 0 0 1px #e0e0e0, inset 0 0 14px rgba(0, 0, 0, 0.2);
}
```








```

</style>
@code{
public List<EmployeeData> Employees { get; set; }
protected override void OnInitialized()
{
    Employees = Enumerable.Range(1, 9).Select(x => new EmployeeData()
    {
        EmployeeID = x,
        FirstName = (new string[] { "Nancy", "Andrew", "Janet", "Margaret", "Steven"
    }) [new Random().Next(5)],
        LastName = (new string[] { "Davolio", "Fuller", "Leverling", "Peacock",
        "Buchanan" }) [new Random().Next(5)],
        Title = (new string[] { "Sales Representative", "Vice President, Sales",
        "Sales Manager",
        "Inside Sales Coordinator" }) [new Random().Next(4)],
        HireDate = DateTime.Now.AddDays(-x),
    }).ToList();
}
public class EmployeeData
{
    public int? EmployeeID { get; set; }
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public string Title { get; set; }
    public DateTime? HireDate { get; set; }
}
}

```

The following screenshot represents the column Template.

Employee Image	Employee ID	First Name	Title	Hire Date
	1	Nancy	Sales Manager	7/24/2019
	2	Steven	Sales Representati...	7/23/2019
	3	Margaret	Sales Representati...	7/22/2019
	4	Janet	Vice President, Sales	7/21/2019
	5	Andrew	Inside Sales Coord...	7/20/2019

GridTemplates component

If a component contains any **RenderFragment** type property then it does not allow any child components other than the render fragment property, which is [by design in Blazor](#).

This prevents us from directly specifying templates such as **RowTemplate** and **DetailTemplate** as descendant of DataGrid component. Hence the templates such as **RowTemplate** and **DetailTemplate** should be wrapped around a component named **GridTemplates** as follows.

From v17.4.39, the **ModelType** property has been removed from the DataGrid Component.

CSHARP

```
@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Employees">
<GridTemplates>
<RowTemplate Context="emp">
@{
var employee = (emp as EmployeeData);
<td class="photo">

</td>
<td class="details">
<table class="CardTable" cellpadding="3" cellspacing="2">
<colgroup>
<col width="50%">
<col width="50%">
</colgroup>
<tbody>
<tr>
<td class="CardHeader">First Name </td>
<td>@employee.FirstName </td>
</tr>
<tr>
<td class="CardHeader">Last Name</td>
<td>@employee.LastName </td>
</tr>
<tr>
<td class="CardHeader">
Title
</td>
<td>
@employee.Title
</td>
</tr>
<tr>
<td class="CardHeader">
Birth Date
</td>
<td>
@employee.BirthDate.Value.ToShortDateString()
</td>
</tr>
</tbody>
</table>
</td>
}
```

```

</RowTemplate>
</GridTemplates>
<GridColumn>
  <GridColumn HeaderText="Employee Image" Width="250"
  TextAlign="TextAlign.Center"> </GridColumn>
  <GridColumn HeaderText="Employee Details" Width="300"
  TextAlign="TextAlign.Left"></GridColumn>
</GridColumn>
</SfGrid>
<style type="text/css" class="cssStyles">
.photo img {
width: 100px;
height: 100px;
border-radius: 50px;
box-shadow: inset 0 0 1px #e0e0e0, inset 0 0 14px rgba(0, 0, 0, 0.2);
}
.photo,
.details {
border-color: #e0e0e0;
border-style: solid;
}
.photo {
border-width: 1px 0px 0px 0px;
text-align: center;
}
.details {
border-width: 1px 0px 0px 0px;
padding-left: 18px;
}
.e-bigger .details {
padding-left: 25px;
}
.e-device .details {
padding-left: 8px;
}
.details > table {
width: 100%;
}
.CardHeader {
font-weight: 600;
}
td {
padding: 2px 2px 3px 4px;
}
</style>
@code{
public List<EmployeeData> Employees { get; set; }
protected override void OnInitialized()
{
Employees = Enumerable.Range(1, 9).Select(x => new EmployeeData()
{
EmployeeID = x,
FirstName = (new string[] { "Nancy", "Andrew", "Janet", "Margaret", "Steven"
})[new Random().Next(5)],
LastName = (new string[] { "Davolio", "Fuller", "Leverling", "Peacock",
"Buchanan" })[new Random().Next(5)],





```

```

Title = (new string[] { "Sales Representative", "Vice President, Sales",
"Sales Manager",
"Inside Sales Coordinator" }) [new Random().Next(4)],
BirthDate = DateTime.Now.AddDays(-x),
}).ToList();
}
public class EmployeeData
{
public int? EmployeeID { get; set; }
public string FirstName { get; set; }
public string LastName { get; set; }
public string Title { get; set; }
public DateTime? BirthDate { get; set; }
}
}

```

The following image represents the Row Template

Employee Image	Employee Details	
	First Name	Nancy
	Last Name	Buchanan
	Title	Vice President, Sales
	Country	USA
	First Name	Margaret
	Last Name	Buchanan
	Title	Inside Sales Coordinator
	Country	USA
	First Name	Steven
	Last Name	Buchanan
	Title	Sales Manager
	Country	UK
	First Name	Margaret

You can refer to our [Blazor DataGrid](#) feature tour page for its groundbreaking feature representations. You can also explore our [Blazor DataGrid example](#) to understand how to present and manipulate data.

Grouping in Blazor DataGrid Component

The DataGrid has options to group records by dragging and dropping the column header to the group drop area. When grouping is applied, datagrid records are organized into a hierarchical structure to facilitate easier expansion and collapse of records.

To enable grouping in the datagrid, set the [AllowGrouping](#) as true. Grouping options can be configured through the [GridGroupSettings](#) component.

ASPX-CS

```

@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders" AllowGrouping="true" Height="400">
<GridColumns>

```

```

<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumns>
</SfGrid>
@code{
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
OrderDate = DateTime.Now.AddDays(-x),
}).ToList();
}
public class Order {
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
}

```

The following GIF image represents a DataGrid Grouping.

Drag a column header here to group its column			
Order ID	Customer Name	Order Date	Freight
1001	BLONP	7/24/2019	\$2.10
1002	BLONP	7/23/2019	\$4.20
1003	ANTON	7/22/2019	\$6.30
1004	ANTON	7/21/2019	\$8.40
1005	BLONP	7/20/2019	\$10.50
1006	ANTON	7/19/2019	\$12.60
1007	BLONP	7/18/2019	\$14.70
1008	BLONP	7/17/2019	\$16.80
1009	ALFKI	7/16/2019	\$18.90

* You can group and ungroup columns by using the **GroupColumn** and **UngroupColumn** methods.

* To disable grouping for a particular column, set the [AllowGrouping](#) to false in **GridColumn** component.

Initial group

To apply group at initial rendering, set the column field name in the [Columns](#) property of **GridGroupSettings** component.

ASPX-CS

```
@using Syncfusion.Blazor.Grids
@{
    var Initial = (new string[] { "OrderID" });
}
<SfGrid DataSource="@Orders" AllowGrouping="true" Height="400">
    <GridGroupSettings Columns="@Initial"></GridGroupSettings>
    <GridColumn>
        <GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
            TextAlign="TextAlign.Right" Width="120"></GridColumn>
        <GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
            Width="150"></GridColumn>
        <GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
            Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
            Width="130"></GridColumn>
        <GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
            TextAlign="TextAlign.Right" Width="120"></GridColumn>
    </GridColumn>
</SfGrid>
@code{
    public List<Order> Orders { get; set; }
    protected override void OnInitialized()
    {
        Orders = Enumerable.Range(1, 75).Select(x => new Order()
        {
            OrderID = 1000 + x,
            CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
            })[new Random().Next(5)],
            Freight = 2.1 * x,
            OrderDate = DateTime.Now.AddDays(-x),
        }).ToList();
    }
    public class Order {
        public int? OrderID { get; set; }
        public string CustomerID { get; set; }
        public DateTime? OrderDate { get; set; }
        public double? Freight { get; set; }
    }
}
```

The following screenshot represents a DataGrid with initial grouping.

Order ID ↑ ×			
Customer Name	Order Date	Freight	
^ Order ID: 1001 - 1 item			
ANTON	7/23/2019	\$2.10	
^ Order ID: 1002 - 1 item			
ANTON	7/22/2019	\$4.20	
^ Order ID: 1003 - 1 item			
BOLID	7/21/2019	\$6.30	
^ Order ID: 1004 - 1 item			
BLONP	7/20/2019	\$8.40	
^ Order ID: 1005 - 1 item			
BLONP	7/19/2019	\$10.50	

Hide drop area

To avoid ungrouping or further grouping of a column after initial column grouping, define the [ShowDropArea](#) of **GridGroupSettings** as false.

ASPX-CS

```
@using Syncfusion.Blazor.Grids
@{
    var Hide = (new string[] { "Freight" });
}
<SfGrid DataSource="@Orders" AllowGrouping="true" Height="400">
    <GridGroupSettings ShowDropArea="false" Columns=@Hide></GridGroupSettings>
    <GridColumns>
        <GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
            TextAlign="TextAlign.Right" Width="120"></GridColumn>
        <GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
            Width="150"></GridColumn>
        <GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
            Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
            Width="130"></GridColumn>
        <GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
            TextAlign="TextAlign.Right" Width="120"></GridColumn>
    </GridColumns>
</SfGrid>
@code{
    public List<Order> Orders { get; set; }
    protected override void OnInitialized()
```

```

{
    Orders = Enumerable.Range(1, 75).Select(x => new Order()
    {
        OrderID = 1000 + x,
        CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
        })[new Random().Next(5)],
        Freight = 2.1 * x,
        OrderDate = DateTime.Now.AddDays(-x),
    }).ToList();
}

public class Order {
    public int? OrderID { get; set; }
    public string CustomerID { get; set; }
    public DateTime? OrderDate { get; set; }
    public double? Freight { get; set; }
}
}

```

The following screenshot represents hiding the Group drop area in DataGrid.

	Order ID	Customer Name	Order Date
^ Freight: \$2.10 - 1 item	1001	BLONP	7/24/2019
^ Freight: \$4.20 - 1 item	1002	ALFKI	7/23/2019
^ Freight: \$6.30 - 1 item	1003	BOLID	7/22/2019
^ Freight: \$8.40 - 1 item	1004	BOLID	7/21/2019
^ Freight: \$10.50 - 1 item	1005	ANANTR	7/20/2019

<!--Group with paging

On grouping columns with paging feature, the aggregated information and total items are displayed based on the current page. The grid does not consider aggregated information and total items from other pages. To get additional details (aggregated information and total items) from other pages, set the [DisablePageWiseAggregates](#) property of **GridGroupSettings** to false.

If remote data is bound to grid dataSource, two requests will be sent when performing grouping action; one for getting the grouped data and another for getting aggregate details and total items count. -->

Group by format

By default, a column will be grouped by the data or value present for the particular row. To group the numeric or datetime column based on the mentioned format, you have to enable the [EnableGroupByFormat](#) property of the corresponding datagrid columns.

ASPX-CS

```
@using Syncfusion.Blazor.Grids
@{
    var Format = (new string[] { "Freight" });
}
<SfGrid DataSource="@Orders" AllowGrouping="true" Height="400">
    <GridGroupSettings ShowDropArea="false" Columns=@Format></GridGroupSettings>
    <GridColumns>
        <GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
            TextAlign="TextAlign.Right" Width="120"></GridColumn>
        <GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
            Width="150"></GridColumn>
        <GridColumn Field=@nameof(Order.OrderDate) HeaderText="Order Date"
            Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
            Width="130"></GridColumn>
        <GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
            TextAlign="TextAlign.Right" EnableGroupByFormat="true"
            Width="120"></GridColumn>
    </GridColumns>
</SfGrid>
@code{
    public List<Order> Orders { get; set; }
    protected override void OnInitialized()
    {
        Orders = Enumerable.Range(1, 75).Select(x => new Order()
        {
            OrderID = 1000 + x,
            CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
            })[new Random().Next(5)],
            Freight = 2.1 * x,
            OrderDate = DateTime.Now.AddDays(-x),
        }).ToList();
    }
    public class Order {
        public int? OrderID { get; set; }
        public string CustomerID { get; set; }
        public DateTime? OrderDate { get; set; }
        public double? Freight { get; set; }
    }
}
```

Grouping events

During the group action, the datagrid component triggers two events. The [OnActionBegin](#) event triggers before the group action starts and the [OnActionComplete](#) event triggers after the group action is completed. Using these events you can perform any action.

ASPX-CS

```
@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders" AllowGrouping="true">
```

```

<GridEvents OnActionComplete="GroupActionHandler"
OnActionBegin="GroupActionHandler" TValue="Order"></GridEvents>
<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.ShipCountry) HeaderText="Ship Country"
Width="150"></GridColumn>
</GridColumns>
</SfGrid>
@code{
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
OrderDate = DateTime.Now.AddDays(-x),
ShipCountry = (new string[] { "USA", "UK", "JAPAN" })[new Random().Next(3)],
}).ToList();
}
public class Order {
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
public string ShipCountry { get; set; }
}
public void GroupActionHandler(ActionEventArgs<Order> args){
// You can get args.RequestType and other details.
}
}

```

The **args.RequestType** is based on the current action name. For example, when grouping, the **args.RequestType** value will be 'grouping'.

Caption template

You can customize the group caption by using the [CaptionTemplate](#) of the [GridGroupSettings](#) component.

The following sample code demonstrates the above,

ASPX-CS

```

@using Syncfusion.Blazor.Grids
@{
var Template = (new string[] { "OrderID" });

```

```

}
<SfGrid DataSource="@Orders" AllowGrouping="true" Height="400">
<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumns>
<GridGroupSettings Columns=@Template>
<CaptionTemplate>
@{
var order = (context as CaptionTemplateContext);
<div>@order.Field - @order.Key</div>
}
</CaptionTemplate>
</GridGroupSettings>
</SfGrid>
@code{
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
OrderDate = DateTime.Now.AddDays(-x),
}).ToList();
}
public class Order
{
public int OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
}

```

The following screenshot represents a DataGrid with customized group caption,

Order ID ↑ ×		
Customer Name	Order Date	Freight
BOLID	8/31/2019	\$18.90
^ OrderID - 1010		
BOLID	8/30/2019	\$21.00
^ OrderID - 1011		
BLONP	8/29/2019	\$23.10
^ OrderID - 1012		
BLONP	8/28/2019	\$25.20
^ OrderID - 1013		
BLONP	8/27/2019	\$27.30
^ OrderID - 1014		

You can refer to our [Blazor DataGrid](#) feature tour page for its groundbreaking feature representations. You can also explore our [Blazor DataGrid example](#) to understand how to present and manipulate data.

Lazy Load Grouping

The lazy load grouping allows you to load grouped records to the Grid through the on-demand concept. So, you can use this feature to load a huge amount of grouped data to the Grid without any performance degradation.

When you enable this feature, the Grid will render only the initial level caption rows in the collapsed state at grouping. The child rows of each caption will be fetched from the server and render in the Grid when you expand the caption row.

The caption row expand/collapse state will be persisted on paging and Grid pages count will be determined based on the caption and child rows.

To enable lazy load grouping in the datagrid, set the **EnableLazyLoading** as true in [GridGroupSettings](#) component.

The following sample code demonstrates the above,

ASPX-CS

```
@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders" AllowPaging="true" AllowGrouping="true">
  <GridGroupSettings EnableLazyLoading="true"
    Columns="@Initial"></GridGroupSettings>
  <GridColumns>
```

```

<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumns>
</SfGrid>
@code{
public string[] Initial = (new string[] { "CustomerID" });
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP",
"BOLID","Maria", "Ana Trujillo", "Antonio Moreno", "Thomas Hardy",
"Christina Berglund", "Hanna Moos", "Frédérique Citeaux", "Martín Sommer",
"Laurence Lebihan", "Elizabeth Lincoln",
"Victoria Ashworth", "Patricio Simpson", "Francisco Chang", "Yang Wang",
"Pedro Afonso" })[new Random().Next(20)],
Freight = 2.1 * x,
OrderDate = DateTime.Now.AddDays(-x),
}).ToList();
}
public class Order
{
public int OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
}

```

The following GIF represents the lazy load grouping functionality with paging in DataGrid

Customer Name ↑ ×			
Order ID	Order Date	Freight	
▶ Customer Name: ALFKI - 1 item			
▶ Customer Name: Ana Trujillo - 4 items			
▶ Customer Name: ANANTR - 4 items			
▶ Customer Name: ANTON - 3 items			
▶ Customer Name: Antonio Moreno - 2 items			
▶ Customer Name: BLONP - 4 items			
▶ Customer Name: BOLID - 3 items			
▶ Customer Name: Christina Berglund - 3 items			
▶ Customer Name: Elizabeth Lincoln - 6 items			
▶ Customer Name: Francisco Chang - 2 items			
▶ Customer Name: Frédérique Citeaux - 5 items			
▶ Customer Name: Hanna Moos - 3 items			
<< < 1 2 > >>			1 of 2 pages (20 items)

Lazy Load Grouping With Row Virtualization

When you enable lazy load grouping with virtualization feature, the Grid will render only the initial level caption rows in the collapsed state at grouping. The child rows of each caption will be fetched from the server and render in the Grid when you expand the caption row. The caption row expand/collapse state will be persisted while scrolling.

The following sample code demonstrates the above,

ASPX-CS

```
@using Syncfusion.Blazor.Grids
@using Syncfusion.Blazor.Data
<SfGrid TValue="Customer" DataSource="customers" ID="Grid"
AllowGrouping="true" EnableVirtualization="true" Height="400"
AllowSorting="true">
<GridGroupSettings EnableLazyLoading="true" Columns="@GroupedColumns">
</GridGroupSettings>
<GridColumns>
<GridColumn Field=@nameof(Customer.OrderID) HeaderText="Order ID"
AllowGrouping="false" TextAlign="@TextAlign.Center"
Width="180"></GridColumn>
<GridColumn Field=@nameof(Customer.ProductName) HeaderText="Product"
Width="200"></GridColumn>
<GridColumn Field=@nameof(Customer.CustomerID) HeaderText="Customer Name"
Width="170"></GridColumn>
<GridColumn Field=@nameof(Customer.UnitsInStock) HeaderText="Units In Stock"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumns>
</SfGrid>
@code{
public string[] GroupedColumns = new string[] { "ProductName" };
public List<Customer> customers { get; set; } = Customer.GetAllRecords();
```

```

public class Customer
{
    public int OrderID { get; set; }
    public string CustomerID { get; set; }
    public string CustomerName { get; set; }
    public string CustomerAddress { get; set; }
    public string ProductName { get; set; }
    public int ProductID { get; set; }
    public string Quantity { get; set; }
    public int UnitsInStock { get; set; }
    public static List<Customer> GetAllRecords()
    {
        List<Customer> customers = new List<Customer>();
        string[] CustomerId = {"VINET", "TOMSP", "HANAR", "VICTE", "SUPRD", "HANAR",
        "CHOPS", "RICSU", "WELLI", "HILAA", "ERNSH", "CENTC",
        "OTTIK", "QUEDE", "RATTC", "ERNSH", "FOLKO", "BLONP", "WARTH", "FRANK",
        "GROSR", "WHITC", "WARTH", "SPLIR", "RATTC", "QUICK", "VINET",
        "MAGAA", "TORTU", "MORGK", "BERGS", "LEHMS", "BERGS", "ROMEY", "ROMEY",
        "LILAS", "LEHMS", "QUICK", "QUICK", "RICAR", "REGGC", "BSBEV",
        "COMMI", "QUEDE", "TRADH", "TORTU", "RATTC", "VINET", "LILAS", "BLONP",
        "HUNGO", "RICAR", "MAGAA", "WANDK", "SUPRD", "GODOS", "TORTU",
        "OLDWO", "ROMEY", "LONEP", "ANATR", "HUNGO", "THEBI", "DUMON", "WANDK",
        "QUICK", "RATTC", "ISLAT", "RATTC", "LONEP", "ISLAT", "TORTU",
        "WARTH", "ISLAT", "PERIC", "KOENE", "SAVEA", "KOENE", "BOLID", "FOLKO",
        "FURIB", "SPLIR", "LILAS", "BONAP", "MEREP", "WARTH", "VICTE",
        "HUNGO", "PRINI", "FRANK", "OLDWO", "MEREP", "BONAP", "SIMOB", "FRANK",
        "LEHMS", "WHITC", "QUICK", "RATTC", "FAMIA" };
        string[] Product = { "Chai", "Chang", "Syrup", "Corn Snacks", "Gumbo Mix",
        "Seeds",
        "Dried Pears", "Sauce", "Mishi Kobe Niku", "Ikura", "Queso Cabrales", "Queso
        Manchego Pastora", "Konbu",
        "Tofu", "Genen Shouyu", "Pavlova", "Alice Mutton", "Biscuits", "Teatime
        Chocolate Biscuits", "Sir Rodney\s Marmalade", "Sir Rodney\s Scones",
        "Gustaf\s Knäckebröd", "Tunnbröd", "Guaraná Fantástica", "Nougat-Creme",
        "Gumbär Gummibärchen", "Schoggi Schokolade", "Rössle Sauerkraut",
        "Thüringer Rostbratwurst", "Nord-Ost Matjeshering", "Gorgonzola Telino",
        "Mascarpone Fabioli", "Geitost", "Sasquatch Ale", "Steeleye Stout", "Inlagd
        Sill",
        "Gravad lax", "Nuts", "Chips", "Crab Meat", "Jack\s Clam Chowder",
        "Singaporean Fried Mee", "Ipoh Coffee",
        "Gula Malacca", "Rogede sild", "Spegesild", "Zaanse koeken", "Chocolade",
        "Maxilaku", "Valkoinen suklaa", "Manjimup Dried Apples", "Filo Mix", "Perth
        Pasties",
        "Tourtière", "Pâté chinois", "Ipoh Coffee", "Ravioli Angelo", "Escargots
        Bourgogne", "Raclette Courdavault", "Cake", "Sirop d\érable",
        "Tarte au sucre", "Vegie-spread", "Lakkalikri", "Louisiana Pepper Sauce",
        "Louisiana Hot Spiced Okra", "Lumberjack Lager", "Scottish Longbreads",
        "Gudbrandsdalsost", "Outback Lager", "Flotemysost", "Mozzarella di
        Giovanni", "Röd Kaviar", "Longlife Tofu", "Rhönbräu Klosterbier",
        "Lakkalikööri", "Original Frankfurter" };
        string[] CustomerName = { "Maria", "Ana Trujillo", "Antonio Moreno", "Thomas
        Hardy", "Christina Berglund", "Hanna Moos", "Frédérique Citeaux", "Martín
        Sommer", "Laurence Lebihan", "Elizabeth Lincoln",
        "Victoria Ashworth", "Patricio Simpson", "Francisco Chang", "Yang Wang",
        "Pedro Afonso", "Elizabeth Brown", "Sven Ottlieb", "Janine Labrune", "Ann
        Devon", "Roland Mendel", "Aria Cruz", "Diego Roel",

```

```

"Martine Rancé", "Maria Larsson", "Peter Franken", "Carine Schmitt", "Paolo
Accorti", "Lino Rodriguez", "Eduardo Saavedra", "José Pedro Freyre", "André
Fonseca", "Howard Snyder", "Manuel Pereira",
"Mario Pontes", "Carlos Hernández", "Yoshi Latimer", "Patricia McKenna",
"Helen Bennett", "Philip Cramer", "Daniel Tonini", "Annette Roulet", "Yoshi
Tannamuri", "John Steel", "Renate Messner", "Jaime Yorres",
"Carlos González", "Felipe Izquierdo", "Fran Wilson", "Giovanni Rovelli",
"Catherine Dewey", "Jean Fresnière", "Alexander Feuer", "Simon Crowther",
"Yvonne Moncada", "Rene Phillips", "Henriette Pfalzheim",
"Marie Bertrand", "Guillermo Fernández", "Georg Pipp", "Isabel de Castro",
"Bernardo Batista", "Lúcia Carvalho", "Horst Kloss", "Sergio Gutiérrez",
"Paula Wilson", "Maurizio Moroni", "Janete Limeira", "Michael Holz",
"Alejandra Camino", "Jonas Bergulfen", "Jose Pavarotti", "Hari Kumar",
"Jytte Petersen", "Dominique Perrier", "Art Braunschweiger", "Pascale
Cartrain", "Liz Nixon", "Liu Wong", "Karin Josefs", "Miguel Angel Paolino",
"Anabela Domingues", "Helvetius Nagy", "Palle Ibsen", "Mary Saveley", "Paul
Henriot", "Rita Müller", "Pirkko Koskitalo", "Paula Parente", "Karl
Jablonski", "Matti Karttunen", "Zbyszek Piestrzeniewicz" };
string[] CustomerAddress = { "507 - 20th Ave. E.\r\nApt. 2A", "908 W.
Capital Way", "722 Moss Bay Blvd.", "4110 Old Redmond Rd.", "14 Garrett
Hill", "Coventry House\r\nMiner Rd.", "Edgeham Hollow\r\nWinchester Way",
"4726 - 11th Ave. N.E.", "7 Houndstooth Rd.", "59 rue de l'Abbaye",
"Luisenstr. 48", "908 W. Capital Way", "722 Moss Bay Blvd.", "4110 Old
Redmond Rd.", "14 Garrett Hill", "Coventry House\r\nMiner Rd.", "Edgeham
Hollow\r\nWinchester Way",
"7 Houndstooth Rd.", "2817 Milton Dr.", "Kirchgasse 6", "Sierras de Granada
9993", "Mehrheimerstr. 369", "Rua da Panificadora, 12", "2817 Milton Dr.",
"Mehrheimerstr. 369" };
string[] QuantityPerUnit = { "10 boxes x 20 bags", "24 - 12 oz bottles", "12
- 550 ml bottles", "48 - 6 oz jars", "36 boxes", "12 - 8 oz jars", "12 - 1
lb pkgs.", "12 - 12 oz jars", "18 - 500 g pkgs.", "12 - 200 ml jars",
"1 kg pkg.", "10 - 500 g pkgs.", "2 kg box", "40 - 100 g pkgs.", "24 - 250
ml bottles", "32 - 500 g boxes", "20 - 1 kg tins", "16 kg pkg.", "10 boxes x
12 pieces", "30 gift boxes", "24 pkgs. x 4 pieces", "24 - 500 g pkgs.", "12
- 250 g pkgs.",
"12 - 355 ml cans", "20 - 450 g glasses", "100 - 250 g bags" };
int OrderID = 1001;
int i = 0; int j = 0; int k = 0; int l = 0; int m = 0;
for (int x = 0; x < 20000; x++)
{
    i = i >= CustomerId.Length ? 0 : i; j = j >= CustomerName.Length ? 0 : j; k
    = k >= CustomerAddress.Length ? 0 : k; l = l >= Product.Length ? 0 : l; m =
    m >= QuantityPerUnit.Length ? 0 : m;
    customers.Add(new Customer() { OrderID = OrderID + x, CustomerID =
    CustomerId[i], CustomerName = CustomerName[j], CustomerAddress =
    CustomerAddress[k], ProductName = Product[l], ProductID = x, Quantity =
    QuantityPerUnit[m], UnitsInStock = new Random().Next(1, 1000) });
    i++; j++; k++; l++; m++;
}
return customers;
}
}
}

```

The following GIF represents the lazy load grouping functionality with virtualization in DataGrid

Product ↑ ×		
Order ID	Customer Name	Units In Stock
▶	Product: Alice Mutton - 260 items	
▶	Product: Biscuits - 260 items	
▶	Product: Cake - 259 items	
▶	Product: Chai - 260 items	
▶	Product: Chang - 260 items	
▶	Product: Chips - 260 items	
▶	Product: Chocolate - 260 items	
▶	Product: Corn Snacks - 260 items	
▶	Product: Crab Meat - 260 items	
▶	Product: Dried Pears - 260 items	
▶	Product: Escargots Bourgogne - 259 items	

Lazy Load Grouping With Custom Adaptor

You can use the Custom Adaptor of DataManager when binding the remote data. Along with the default server request, this feature will additionally send the below details to handle the lazy load grouping. In the server end, these details are bound with the **LazyLoad** and **LazyExpandAllGroup** parameters in the DataManagerRequest model.

| Property Name | Description |

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

| LazyLoad | To differentiate between default grouping and lazy load grouping. |

| LazyExpandAllGroup | To handle ExpandAll support for lazy load grouping. |

The following code example describes the lazy load grouping handled at the server-side with other grid actions.

CSHARP

```
// Implementing custom adaptor by extending the DataAdaptor class
public class CustomAdaptor : DataAdaptor
{
    public List<Customer> customers { get; set; } = Customer.GetAllRecords();
    // Performs data Read operation
    public override object Read(DataManagerRequest dm, string key = null)
    {
        IEnumerable<Customer> DataSource = customers;
        if (dm.Search != null && dm.Search.Count > 0)
        {
            // Searching
            DataSource = DataOperations.PerformSearching(DataSource, dm.Search);
        }
        if (dm.Sorted != null && dm.Sorted.Count > 0)
        {
            // Sorting
            DataSource = DataOperations.PerformSorting(DataSource, dm.Sorted);
        }
    }
}
```

```

}
if (dm.Where != null && dm.Where.Count > 0)
{
    // Filtering
    DataSource = DataOperations.PerformFiltering(DataSource, dm.Where,
    dm.Where[0].Operator);
}
int count = DataSource.Cast<Customer>().Count();
if (dm.Skip != 0)
{
    //Paging
    DataSource = DataOperations.PerformSkip(DataSource, dm.Skip);
}
if (dm.Take != 0)
{
    DataSource = DataOperations.PerformTake(DataSource, dm.Take);
}
DataResult DataObject = new DataResult();
if (dm.Group != null)
{
    // Grouping (Perform lazy load grouping need to send LazyLoad property in
    Group method)
    IEnumerable ResultData = DataSource.ToList();
    ResultData = DataUtil.Group<Customer>(DataSource, dm.Group[0],
    dm.Aggregates, 0, dm.GroupByFormatter, dm.LazyLoad, dm.LazyExpandAllGroup);
    DataObject.Result = ResultData;
    DataObject.Count = ResultData.Cast<object>().Count();
    return dm.RequiresCounts ? DataObject : (object)ResultData;
}
return dm.RequiresCounts ? new DataResult() { Result = DataSource, Count =
count } : (object)DataSource;
}
}

```

Filtering in Blazor DataGrid Component

Filtering allows you to view particular records based on filter criteria. To enable filtering in the DataGrid, set the [AllowFiltering](#) to true. Filtering options can be configured through [GridFilterSettings](#) component.

ASPX-CS

```

@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders" AllowFiltering="true" AllowPaging="true">
<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumns>
</SfGrid>
@code{
public List<Order> Orders { get; set; }
}

```

```
protected override void OnInitialized()
{
    Orders = Enumerable.Range(1, 75).Select(x => new Order()
    {
        OrderID = 1000 + x,
        CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
    })[new Random().Next(5)],
        Freight = 2.1 * x,
        OrderDate = (new DateTime[] { new DateTime(2010, 5, 1), new DateTime(2010,
        5, 2), new DateTime(2010, 5, 3), })[new Random().Next(3)],
    }).ToList();
}
public class Order
{
    public int? OrderID { get; set; }
    public string CustomerID { get; set; }
    public DateTime? OrderDate { get; set; }
    public double? Freight { get; set; }
}
}
```

The following screenshot shows filtering using FilterBar

Order ID	Customer Name	Order Date	Freight
<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>
1001	ANANTR	7/23/2019	\$2.10
1002	BLONP	7/22/2019	\$4.20
1003	BOLID	7/21/2019	\$6.30
1004	ANTON	7/20/2019	\$8.40
1005	ANTON	7/19/2019	\$10.50

K
<
1
2
3
4
5
6
7
8
9
10
...
>
>
1 of 15 pages (75 items)

* You can apply and clear filtering by using **FilterByColumn** and **ClearFiltering** methods.

* To disable filtering for a particular column, set [AllowFiltering](#) property of **GridColumn** as false.

Initial filter

To apply the filter at initial rendering, set the filter **Predicate** object in [Columns](#) property of **GridFilterSettings** component.

ASPX-CS

```
@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders" AllowFiltering="true">
<GridFilterSettings>
<GridFilterColumns>
<GridFilterColumn Field = "CustomerID" MatchCase =false Operator =
"Operator.StartsWith" Predicate = "and" Value = "@val" ></GridFilterColumn>
</GridFilterColumns>
```

```

</GridFilterSettings>
<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumns>
</SfGrid>
@code{
public List<Order> Orders { get; set; }
public string val="ANANTR";
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
OrderDate = (new DateTime[] { new DateTime(2010, 5, 1), new DateTime(2010,
5, 2), new DateTime(2010, 5, 3), })[new Random().Next(3)]
}).ToList();
}
public class Order
{
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
}

```

The following screenshot shows Initial filtering using FilterBar

Order ID	Customer Name	Order Date	Freight
	ANANTR		\$35.00
1020	ANANTR	7/4/2019	\$42.00
1025	ANANTR	6/29/2019	\$52.50
1033	ANANTR	6/21/2019	\$69.30
1040	ANANTR	6/14/2019	\$84.00
1043	ANANTR	6/11/2019	\$90.30

K < 1 2 3 > X
1 of 3 pages (12 items)

Customer Name: ANANTR && Freight: 35

Filter operators

The filter operator for a column can be defined in the **Operator** in [Columns](#) property of **GridFilterSettings** component.

The available operators and its supported data types are:

Operator	Description	Supported Types
= =value	StartsWith	Number
!= !=value	NotEqual	Number
>value	GreaterThan	Number
< <value	LessThan	Number
= >=value	GreaterThanOrEqual	Number
<= <=value	LessThanOrEqual	Number
/value	StartsWith	String
% %value	EndsWith	String
N/A N/A	equal	Date
N/A N/A	equal	Boolean

ASPX-CS

```
@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders" AllowFiltering="true" AllowPaging="true">
  <GridPageSettings PageSize="5"></GridPageSettings>
  <GridColumns>
    <GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
      TextAlign="TextAlign.Right" Width="120"></GridColumn>
    <GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
      Width="150"></GridColumn>
    <GridColumn Field=@nameof(Order.OrderDate) HeaderText="Order Date"
      Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
      Width="130"></GridColumn>
    <GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
      TextAlign="TextAlign.Right" Width="120"></GridColumn>
  </GridColumns>
</SfGrid>

@code{
  public List<Order> Orders { get; set; }
  protected override void OnInitialized()
  {
    Orders = Enumerable.Range(1, 75).Select(x => new Order()
    {
      OrderID = 1000 + x,
      CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
    })[new Random().Next(5)],
      Freight = 2.1 * x,
      OrderDate = (new DateTime[] { new DateTime(2010, 5, 1), new DateTime(2010,
      5, 2), new DateTime(2010, 5, 3), })[new Random().Next(3)]
    }).ToList();
  }
  public class Order {
```

```

public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
}

```

The following screenshot shows filtering using FilterBar

Order ID	Customer Name	Order Date	Freight
<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>
1001	ANANTR	7/23/2019	\$2.10
1002	BLONP	7/22/2019	\$4.20
1003	BOLID	7/21/2019	\$6.30
1004	ANTON	7/20/2019	\$8.40
1005	ANTON	7/19/2019	\$10.50

K < 1 2 3 4 5 6 7 8 9 10 ... > >
1 of 15 pages (75 items)

Filter bar template with custom component

The [FilterTemplate](#) property is used to add custom components to a particular column. The following sample shows the dropdown, used as a custom component in the Customer Name column.

To access the filtered values inside the FilterTemplate, you can use the implicit named parameter context. You can type cast the context as PredicateModel to get filter values inside template.

CSHARP

```

@using Syncfusion.Blazor.Data
@using Syncfusion.Blazor.Grids
@using Syncfusion.Blazor.DropDowns
<SfGrid @ref="@Grid" DataSource="@Orders" AllowFiltering="true"
AllowPaging="true" Height="315">
<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150">
<FilterTemplate>
<SfDropDownList Placeholder="Customer Name" ID="CustomerID"
Value="@((string)(context as PredicateModel).Value)" DataSource="@Dropdown"
TValue="string" TItem="Data">
<DropDownListEvents ValueChange="@Change" TItem="Data"
TValue="string"></DropDownListEvents>
<DropDownListFieldSettings Value="CustomerID"
Text="CustomerID"></DropDownListFieldSettings>
</SfDropDownList>
</FilterTemplate>
</GridColumn>

```

```

<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumn>
</SfGrid>
@code{
SfGrid<Order> Grid;
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
OrderDate = (new DateTime[] { new DateTime(2010, 5, 1), new DateTime(2010,
5, 2), new DateTime(2010, 5, 3), })[new Random().Next(3)],
}).ToList();
}
public class Data
{
public string CustomerID { get; set; }
}
List<Data> Dropdown = new List<Data>
{
new Data() { CustomerID= "All" },
new Data() { CustomerID= "ANTON" },
new Data() { CustomerID= "ANANTR" },
new Data() { CustomerID= "ALFKI" },
new Data() { CustomerID= "BOLID" },
new Data() { CustomerID= "BLONP" },
};
public class Order
{
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
public void Change(@Syncfusion.Blazor.DropDowns.ChangeEventArgs<string,
Data> args)
{
if (args.Value == "All")
{
Grid.ClearFiltering();
}
else
{
Grid.FilterByColumn("CustomerID", "contains", args.Value);
}
}
}

```

The following screenshot shows filtering using custom component

Order ID	Customer Name	Order Date	Freight
	Customer Name		
1001	All	8/26/2019	\$2.10
1002	ANTON	8/25/2019	\$4.20
1003	ANANTR	8/24/2019	\$6.30
1004	ALFKI	8/23/2019	\$8.40
1005	BOLID	8/22/2019	\$10.50
1006	BLONP	8/21/2019	\$12.60
1007	ALFKI	8/20/2019	\$14.70
1008	ALFKI	8/19/2019	\$16.80
1009	BOLID	8/18/2019	\$18.90

K < 1 2 3 4 5 6 7 > X 1 of 7 pages (75 items)

Change default filter operator

You can change the default filter operator by extending `FilterSettings` property in the column.

In the following sample, we have changed the default operator for CustomerID column as **contains** from **startswith**

ASPX-CS

```
@using Syncfusion.Blazor
<SfGrid @ref="@Grid" ID="Egrid" DataSource="@Orders" AllowFiltering="true"
AllowPaging="true" Height="315">
  <GridColumns>
    <GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
    TextAlign="TextAlign.Center" Width="120"></GridColumn>
    <GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
    TextAlign="TextAlign.Center" Width="150" FilterSettings="@ (new
    FilterSettings{ Operator = Operator.Contains })"></GridColumn>
    <GridColumn Field=@nameof(Order.OrderDate) HeaderText="Order Date"
    Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Center"
    Width="130"></GridColumn>
    <GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
    TextAlign="TextAlign.Center" Width="120"></GridColumn>
  </GridColumns>
</SfGrid>
@code{
SfGrid<Order> Grid;
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
```



```

OrderDate = (new DateTime[] { new DateTime(2010, 5, 1), new DateTime(2010,
5, 2), new DateTime(2010, 5, 3), })[new Random().Next(3)],
}).ToList();
}
public class Order
{
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
}

```

The following screenshot represents Filter with change in default operator as contains

Order ID	Customer Name	Order Date	Freight
	ton		
1002	ANTON	8/27/2019	\$4.20
1009	ANTON	8/20/2019	\$18.90
1018	ANTON	8/11/2019	\$37.80
1025	ANTON	8/4/2019	\$52.50
1027	ANTON	8/2/2019	\$56.70
1032	ANTON	7/28/2019	\$67.20
1037	ANTON	7/23/2019	\$77.70
1038	ANTON	7/22/2019	\$79.80
1040	ANTON	7/20/2019	\$84.00

K < 1 2 > X
 Customer Name: ton

1 of 2 pages (21 items)

Filter Modes

By default filter bar [Mode](#) will be of OnEnter type. So we need to press the Enter key after typing a value in the filter bar.

You can also perform filtering operation without pressing Enter key in the filter bar by Setting the filter bar [Mode](#) property as [Immediate](#).

The [ImmediateModeDelay](#) property of [GridFilterSettings](#) is used to define the time Grid has to wait for performing filter operation after completion of user typing word.

ASPX-CS

```

@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders" AllowFiltering="true" AllowPaging="true">
  <GridFilterSettings Mode="FilterBarMode.Immediate"
    ImmediateModeDelay="300"></GridFilterSettings>
  <GridColumns>
    <GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
      TextAlign="TextAlign.Right" Width="120"></GridColumn>
    <GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
      Width="150"></GridColumn>
  </GridColumns>
</SfGrid>

```

```

<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumns>
</SfGrid>
@code{
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
OrderDate = (new DateTime[] { new DateTime(2010, 5, 1), new DateTime(2010,
5, 2), new DateTime(2010, 5, 3), })[new Random().Next(3)],
}).ToList();
}
public class Order
{
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
}

```

Filter menu

You can enable filter menu by setting the [Type](#) of **GridFilterSettings** as **Menu**. The filter menu UI will be rendered based on its column type, which allows you to filter data.

You can filter the records with different operators.

ASPX-CS

```

@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders" AllowFiltering="true" AllowPaging="true"
Height="315">
<GridFilterSettings Type
="Syncfusion.Blazor.Grids.FilterType.Menu"></GridFilterSettings>
<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumns>
</SfGrid>
@code{

```

```

public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
    Orders = Enumerable.Range(1, 75).Select(x => new Order()
    {
        OrderID = 1000 + x,
        CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
    })[new Random().Next(5)],
        Freight = 2.1 * x,
        OrderDate = (new DateTime[] { new DateTime(2010, 5, 1), new DateTime(2010,
    5, 2), new DateTime(2010, 5, 3), })[new Random().Next(3)],
    }).ToList();
}

public class Order {
    public int? OrderID { get; set; }
    public string CustomerID { get; set; }
    public DateTime? OrderDate { get; set; }
    public double? Freight { get; set; }
}

```

The following screenshot represents Menu filter

Order ID ▾	Customer Name ▾	Order Date ▾	Freight ▾
1001	BOLID		\$2.10
1002	ALFKI		\$4.20
1003	ANANTR		\$6.30
1004	ANANTR		\$8.40
1005	BOLID		\$10.50
1006	ALFKI	7/18/2019	\$12.60
1007	ALFKI	7/17/2019	\$14.70
1008	ANTON	7/16/2019	\$16.80
1009	BOLID	7/15/2019	\$18.90

Equal ▾

Choose a Date

Filter

Clear

K < 1 2 3 4 5 6 7 > X

1 of 7 pages (75 items)

* [AllowFiltering](#) must be set as true to enable filter menu.

* Setting [AllowFiltering](#) property of **GridColumn** as false will prevent

Custom component in filter menu

You can use **Menu** type filter in the datagrid. To do so, set the [Type](#) as **Menu** in the **GridFilterSettings**.

In the following sample the [FilterTemplate](#) property is used to add custom components to a particular column. To access the filtered values inside the FilterTemplate, you can use the implicit named parameter context. You can type cast the context as `PredicateModel<T>` to get filter values inside template.

ASPX-CS

```

@using Syncfusion.Blazor.Grids
@using Syncfusion.Blazor.DropDowns
<SfGrid DataSource="@Orders" AllowFiltering="true" AllowPaging="true"
Height="315">
<GridFilterSettings
Type="Syncfusion.Blazor.Grids.FilterType.Menu"></GridFilterSettings>
<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
TextAlign="TextAlign.Center" Width="120">
<FilterTemplate>
<SfDropDownList Placeholder="OrderID" ID="OrderID" @bind-Value="@((context
as PredicateModel<int?>).Value)" TItem="Order" TValue="int?"
DataSource="@ (Orders)">
<DropDownListFieldSettings Value="OrderID"
Text="OrderID"></DropDownListFieldSettings>
</SfDropDownList>
</FilterTemplate>
</GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
TextAlign="TextAlign.Center" Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Center"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Center" Width="120"></GridColumn>
</GridColumns>
</SfGrid>
@code{
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
OrderDate = (new DateTime[] { new DateTime(2010, 5, 1), new DateTime(2010,
5, 2), new DateTime(2010, 5, 3), })[new Random().Next(3)],
}).ToList();
}
public class Order
{
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
}

```

The following screenshot shows filter menu using custom component

Order ID	Customer Name	Order Date	Freight
1001		8/26/2019	\$2.10
1002		8/25/2019	\$4.20
1003		8/24/2019	\$6.30
1004		8/23/2019	\$8.40
1005		8/22/2019	\$10.50
1006	ANTON	8/21/2019	\$12.60
1007	ALFKI	8/20/2019	\$14.70
1008	BLONP	8/19/2019	\$16.80
1009	ANANTR	8/18/2019	\$18.90

< 1 2 3 4 5 6 7 > X
 1 of 7 pages (75 items)

Override default filter operators for Menu Filtering

The default filter operators for a GridColumn can be overridden by using the [OnActionBegin](#) event of the grid. In the below code, we have overridden the filter operators for the **CustomerID** column.

ASPX-CS

```
@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders" AllowFiltering="true" AllowPaging="true"
Height="315">
<GridEvents OnActionBegin="ActionBeginHandler" TValue="Order"></GridEvents>
<GridFilterSettings
Type="Syncfusion.Blazor.Grids.FilterType.Menu"></GridFilterSettings>
<GridColumn>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumn>
</SfGrid>
@code{
public List<Order> Orders { get; set; }
public void ActionBeginHandler(ActionEventArgs<Order> Args)
{
if (Args.RequestType == Syncfusion.Blazor.Grids.Action.FilterBeforeOpen)
{
if (Args.ColumnName == "CustomerID")//Specify Field name
{
Args.FilterOperators = CustomerIDOperator;
}
}
}
public class Operators
{
public string Value { get; set; }
}
```

```

public string Text { get; set; }
}
List<object> CustomerIDOperator = new List<object> {
new Operators() { Text= "Equal", Value= "equal" },
new Operators() { Text= "Contains", Value= "contains" }
};
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
OrderDate = (new DateTime[] { new DateTime(2010, 5, 1), new DateTime(2010,
5, 2), new DateTime(2010, 5, 3), })[new Random().Next(3)],
}).ToList();
}
public class Order
{
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
}

```

Enable different filter for a column

You can use different filter types such as **Menu**, **CheckBox** and **Excel** filter in a same DataGrid. To do so, set the [Type](#) as **Menu** in **GridFilterSettings** and **CheckBox** in [Filter](#) property of **GridColumn** component.

In the following sample menu filter is enabled by default and checkbox filter is enabled for the CustomerID column using the [Filter](#) property of **GridColumn** component.

ASPX-CS

```

@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders" AllowFiltering="true" AllowPaging="true"
Height="315">
<GridFilterSettings Type
="Syncfusion.Blazor.Grids.FilterType.Menu"></GridFilterSettings>
<GridColumn>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
FilterSettings="@ (new FilterSettings{Type =
Syncfusion.Blazor.Grids.FilterType.CheckBox })" Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumn>
</SfGrid>
@code{
public List<Order> Orders { get; set; }
}

```

```
protected override void OnInitialized()
{
    Orders = Enumerable.Range(1, 75).Select(x => new Order()
    {
        OrderID = 1000 + x,
        CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
    })[new Random().Next(5)],
        Freight = 2.1 * x,
        OrderDate = (new DateTime[] { new DateTime(2010, 5, 1), new DateTime(2010,
        5, 2), new DateTime(2010, 5, 3), })[new Random().Next(3)],
    }).ToList();
}

public class Order {
    public int? OrderID { get; set; }
    public string CustomerID { get; set; }
    public DateTime? OrderDate { get; set; }
    public double? Freight { get; set; }
}
}
```

The following screenshot represents CheckBox filter

Order ID	Customer Name	Order Date	Freight
1001	BOLID		\$2.10
1002	ALFKI		\$4.20
1003	ANANTR		\$6.30
1004	ANANTR		\$8.40
1005	BOLID		\$10.50
1006	ALFKI		\$12.60
1007	ALFKI		\$14.70
1008	ANTON		\$16.80
1009	BOLID		\$18.90

1 of 7 pages (75 items)

CheckBox Filter

You can enable the CheckBox filter to show a list of possible distinct filter values for a column. Checkbox list sorted by ascending order to display the data source as readable.

To enable the checkbox list filtering in the DataGrid component by setting the [Type](#) as **CheckBox**.

CheckBox filter contains an option such as searching and Clear filter.

ASPX-CS

```
@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders" AllowFiltering="true" AllowPaging="true"
Height="375">
```

```

<GridFilterSettings
Type="Syncfusion.Blazor.Grids.FilterType.CheckBox"></GridFilterSettings>
<GridColumn>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumn>
</SfGrid>
@code{
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
OrderDate = (new DateTime[] { new DateTime(2010, 5, 1), new DateTime(2010,
5, 2), new DateTime(2010, 5, 3), })[new Random().Next(3)],
}).ToList();
}
public class Order
{
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
}

```

The following screenshot represents Checkbox filter

Order ID ▾	Customer Name ▾	Order Date ▾	Freight ▾
1001	AN	5/1/2010	\$2.10
1002	BL	5/1/2010	\$4.20
1003	AN	5/3/2010	\$6.30
1004	BC	5/3/2010	\$8.40
1005	AN	5/1/2010	\$10.50
1006	AL	5/2/2010	\$12.60
1007	BC	5/2/2010	\$14.70
1008	AL	5/2/2010	\$16.80
1009	BL	5/3/2010	\$18.90
1010	AN	5/3/2010	\$21.00
1011	BLOP	5/1/2010	\$23.10

1 of 7 pages (75 items)

Excel like filter

You can enable Excel like filter by defining [Type](#) as **Excel**. The excel menu contains an option such as Sorting, Clear filter, Sub menu for advanced filtering.

ASPX-CS

```
@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders" AllowFiltering="true">
  <GridFilterSettings Type
    ="Syncfusion.Blazor.Grids.FilterType.Excel"></GridFilterSettings>
  <GridColumn>
    <GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
      TextAlign="TextAlign.Right" Width="120"></GridColumn>
    <GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
      Width="150"></GridColumn>
    <GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
      Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
      Width="130"></GridColumn>
    <GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
      TextAlign="TextAlign.Right" Width="120"></GridColumn>
  </GridColumn>
</SfGrid>
@code{
  public List<Order> Orders { get; set; }
  protected override void OnInitialized()
  {
    Orders = Enumerable.Range(1, 75).Select(x => new Order()
    {
      OrderID = 1000 + x,
      CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
    })[new Random().Next(5)],
      Freight = 2.1 * x,
      OrderDate = (new DateTime[] { new DateTime(2010, 5, 1), new DateTime(2010,
      5, 2), new DateTime(2010, 5, 3), })[new Random().Next(3)],
    }).ToList();
  }
}
```

```

public class Order {
    public int? OrderID { get; set; }
    public string CustomerID { get; set; }
    public DateTime? OrderDate { get; set; }
    public double? Freight { get; set; }
}

```

The following screenshot represents Excel filter

Order ID	Customer Name	Order Date	Freight
1001	BOLID		\$2.10
1002	ALFKI		\$4.20
1003	ANANTR		\$6.30
1004	ANANTR		\$8.40
1005	BOLID		\$10.50
1006	ALFKI		\$12.60
1007	ALFKI		\$14.70
1008	ANTON		\$16.80
1009	BOLID		\$18.90

Clear Filter

Date Filters

Search

☒ Select All
 ☒ 5/10/2019
 ☒ 5/11/2019
 ☒ 5/12/2019
 ☒ 5/13/2019
 ☒ 5/14/2019

OK Cancel

1 of 7 pages (75 items)

The following screenshot represents Custom filter in Excel filter

Order ID	Customer Name	Order Date	Freight
		7/23/2019	\$2.10
		7/22/2019	\$4.20
		7/21/2019	\$6.30
		7/20/2019	\$8.40
		7/19/2019	\$10.50
		7/18/2019	\$12.60
		7/17/2019	\$14.70
		7/16/2019	\$16.80
		7/15/2019	\$18.90

Custom Filter

Show rows where:

Order Date

Equal

Choose a date

☒ AND
 ☐ OR

Choose a date

OK Cancel

1 of 7 pages (75 items)

Filter Item template

This **FilterItemTemplate** helps to you customize the each CheckBox list element/value for display purpose. To access the checkbox list values inside the **FilterItemTemplate**, you can use the implicit named parameter context. You can type cast the context as [FilterItemTemplateContext](#) to get list values inside template.

ASPX-CS

```
@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders" AllowFiltering="true" AllowPaging="true"
Height="375">
<GridFilterSettings
Type="Syncfusion.Blazor.Grids.FilterType.CheckBox"></GridFilterSettings>
<GridColumn>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150">
<FilterItemTemplate>
@{
var filterContext = (context as FilterItemTemplateContext);
var itemTemplateValue = "Textof(" + filterContext.Value + ")";
}
@itemTemplateValue
</FilterItemTemplate>
</GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumn>
</SfGrid>
@code{
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
OrderDate = (new DateTime[] { new DateTime(2010, 5, 1), new DateTime(2010,
5, 2), new DateTime(2010, 5, 3), })[new Random().Next(3)],
}).ToList();
}
public class Order
{
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
}
```

The following screenshot represents Filter Item template

Order ID	Customer Name	Order Date	Freight
1001	BL	5/3/2010	\$2.10
1002	BL	5/2/2010	\$4.20
1003	BL	5/3/2010	\$6.30
1004	AN	5/1/2010	\$8.40
1005	AN	5/1/2010	\$10.50
1006	AN	5/1/2010	\$12.60
1007	AN	5/1/2010	\$14.70
1008	AN	5/1/2010	\$16.80
1009	AN	5/3/2010	\$18.90
1010	AN	5/2/2010	\$21.00
1011	ANTON	5/2/2010	\$23.10

You can refer to our [Blazor DataGrid](#) feature tour page for its groundbreaking feature representations. You can also explore our [Blazor DataGrid example](#) to understand how to present and manipulate data.

<!-- Diacritics

By default, datagrid ignores diacritic characters while filtering. To include diacritic characters, set the [GridFilterSettings.IgnoreAccent](#) property to be true.

In the following sample, type aeo in [CustomerID] column to filter diacritic characters.

ASPX-CS

CSHARP

```
@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders" AllowSorting="true" Height="270">
<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumns>
</SfGrid>
@code{
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
```

```

{
    OrderID = 1000 + x,
    CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
    })[new Random().Next(5)],
    Freight = 2.1 * x,
    OrderDate = DateTime.Now.AddDays(-x),
}).ToList();
}

public class Order {
    public int? OrderID { get; set; }
    public string CustomerID { get; set; }
    public DateTime? OrderDate { get; set; }
    public double? Freight { get; set; }
}
}

```

* DataGrid columns are sorted in the **Ascending** order. If you click the already sorted column, the sort direction toggles.

* You can apply and clear sorting by invoking **SortColumn** and **ClearSorting** methods.

* To disable sorting for a particular column, set the [AllowSorting](#) property of **GridColumn** to false.

Initial sort

To sort at initial rendering, set the **Field** and **Direction** in [Columns](#) property of **GridSortSettings** component.

CSHARP

```

@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders" AllowSorting="true" Height="270">
    <GridSortSettings>
        <GridSortColumns>
            <GridSortColumn Field="OrderDate"
            Direction="SortDirection.Ascending"></GridSortColumn>
            <GridSortColumn Field="Freight"
            Direction="SortDirection.Descending"></GridSortColumn>
        </GridSortColumns>
    </GridSortSettings>
    <GridColumn>
        <GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
        TextAlign="TextAlign.Right" Width="120"></GridColumn>
        <GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
        Width="150"></GridColumn>
        <GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
        Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
        Width="130"></GridColumn>
        <GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
        TextAlign="TextAlign.Right" Width="120"></GridColumn>
    </GridColumn>
</SfGrid>
@code{
    public List<Order> Orders { get; set; }
    protected override void OnInitialized()
    {
        Orders = Enumerable.Range(1, 75).Select(x => new Order()

```

```
{
    OrderID = 1000 + x,
    CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
    })[new Random().Next(5)],
    Freight = 2.1 * x,
    OrderDate = DateTime.Now.AddDays(-x),
}).ToList();
}

public class Order {
    public int? OrderID { get; set; }
    public string CustomerID { get; set; }
    public DateTime? OrderDate { get; set; }
    public double? Freight { get; set; }
}
}
```

Multi-column sorting

You can sort more than one column in a DataGrid. To sort multiple columns, press and hold the **CTRL** key and click the column header. The sorting order will be displayed in the header while performing multi-column sorting.

To clear sorting for a particular column, press the "Shift + mouse left click".

The [AllowSorting](#) must be **true** while enabling multi-column sort.

Set [AllowMultiSorting](#) property as **false** to disable multi-column sorting.

CSHARP

```
@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders" AllowSorting="true" AllowMultiSorting="true"
Height="270">
    <GridColumn>
        <GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
        TextAlign="TextAlign.Right" Width="120"></GridColumn>
        <GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
        Width="150"></GridColumn>
        <GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
        Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
        Width="130"></GridColumn>
        <GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
        TextAlign="TextAlign.Right" Width="120"></GridColumn>
    </GridColumn>
</SfGrid>

@code{
    public List<Order> Orders { get; set; }
    protected override void OnInitialized()
    {
        Orders = Enumerable.Range(1, 75).Select(x => new Order()
        {
            OrderID = 1000 + x,
            CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
            })[new Random().Next(5)],
            Freight = 2.1 * x,
            OrderDate = DateTime.Now.AddDays(-x),
        }).ToList();
    }
}
```

```

}
public class Order {
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
}

```

Sort order

By default, the sorting order will be as **Ascending -> Descending -> None**.

When first click a column header it sorts the column in ascending. Again click the same column header, it will sort the column in descending. A repetitive third click on the same column header will clear the sorting.

Sorting events

During the sort action, the datagrid component triggers two events. The [OnActionBegin](#) event triggers before the sort action starts, and the [OnActionComplete](#) event triggers after the sort action is completed. Using these events you can perform the needed actions.

CSHARP

```

@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders" AllowSorting="true">
<GridEvents OnActionBegin="SortEvent" OnActionComplete="SortEvent"
TValue="Order"></GridEvents>
<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
IsPrimaryKey="true" TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" TextAlign="TextAlign.Right" Width="130"
Type="ColumnType.Date"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" MinWidth="10" Width="120"
MaxWidth="200"></GridColumn>
</GridColumns>
</SfGrid>
@code{
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
OrderDate = DateTime.Now.AddDays(-x),
}).ToList();
}
public class Order {
public int? OrderID { get; set; }
public string CustomerID { get; set; }
}
}

```

```

public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
public void SortEvent(ActionEventArgs<Order> args) {
    // You can get action information from the argument.
}
}

```

Custom sort comparer

You can customize the default sort action for a specific Grid column by defining the [SortComparer](#) property of GridColumn Directive. The SortComparer data type was the IComparer interface, so the custom sort comparer class should be implemented in the interface [IComparer](#).

In the following code example, custom SortComparer class was defined in the CustomerID Column.

CSHARP

```

@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders" AllowSorting="true" Width="600" Height="270">
    <GridColumn>
        <GridColumn Field=@nameof(Order.OrderID) Visible="false" HeaderText="Order
        ID" Width="80"></GridColumn>
        <GridColumn Field=@nameof(Order.CustomerID) SortComparer="new
        CustomComparer()" HeaderText="Customer Name" Width="120"></GridColumn>
        <GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
        Format="d" Type="ColumnType.Date" Width="100"></GridColumn>
        <GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
        Width="80"></GridColumn>
        <GridColumn Field=@nameof(Order.ShipCountry) HeaderText="ShipCountry"
        Format="C2" Width="100"></GridColumn>
    </GridColumn>
</SfGrid>
@code{
    public List<Order> Orders { get; set; }
    protected override void OnInitialized()
    {
        Orders = Enumerable.Range(1, 10).Select(x => new Order()
        {
            OrderID = 1000 + x,
            CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
            })[new Random().Next(5)],
            Freight = 2.1 * x,
            OrderDate = DateTime.Now.AddDays(-x),
            ShipCountry = (new string[] { "USA", "UK", "INDIA", "CHINA", "ENGLAND" })[new
            Random().Next(5)],
        }).ToList();
    }
    public class Order
    {
        public int? OrderID { get; set; }
        public string CustomerID { get; set; }
        public DateTime? OrderDate { get; set; }
        public double? Freight { get; set; }
        public string ShipCountry { get; set; }
    }
    public class CustomComparer : IComparer<Object>

```



```

{
    public int Compare(object XRowDataToCompare, object YRowDataToCompare)
    {
        Order XRowData = XRowDataToCompare as Order;
        Order YRowData = YRowDataToCompare as Order;
        int XRowDataOrderID = (int)XRowData.OrderID;
        int YRowDataOrderID = (int)YRowData.OrderID;
        if (XRowDataOrderID < YRowDataOrderID)
        {
            return -1;
        }
        else if (XRowDataOrderID > YRowDataOrderID)
        {
            return 1;
        }
        else
        {
            return 0;
        }
    }
}

```



The following GIF represents custom SortComparer for CustomerID column. When the user clicks the CustomerID column's header, the custom SortComparer will sort the data according to the OrderID field value. So that custom SortComparer class sort the column data by using another column's value.

| Customer Name | Order Date | Freight | ShipCountry |
|---------------|------------|---------|-------------|
| BLONP | 10/5/2021 | \$2.10 | ENGLAND |
| BOLID | 10/4/2021 | \$4.20 | UK |
| ALFKI | 10/3/2021 | \$6.30 | ENGLAND |
| ANTON | 10/2/2021 | \$8.40 | INDIA |
| ANANTR | 10/1/2021 | \$10.50 | USA |
| ANANTR | 9/30/2021 | \$12.60 | ENGLAND |
| BLONP | 9/29/2021 | \$14.70 | ENGLAND |
| BOLID | 9/28/2021 | \$16.80 | USA |

The SortComparer property will work only for local data.

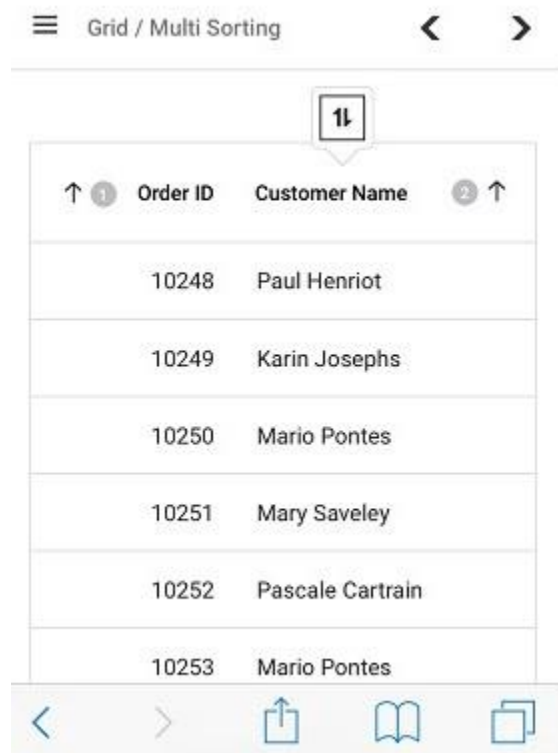
Touch interaction

When you tap the datagrid header on touchscreen devices, the selected column header is sorted. A

popup  is displayed for multi-column sorting. To sort multiple columns, tap the popup , and then tap the desired datagrid headers.

The [AllowMultiSorting](#) and [AllowSorting](#) should be **true** then only the popup will be shown.

The following screenshot shows datagrid touch sorting.



You can refer to our [Blazor DataGrid](#) feature tour page for its groundbreaking feature representations. You can also explore our [Blazor DataGrid example](#) to understand how to present and manipulate data.

Searching in Blazor DataGrid Component

You can search records in a DataGrid, by using the **Search** method with search key as a parameter. This also provides an option to integrate search text box in datagrid's toolbar by adding **Search** item to the [Toolbar](#).

ASPX-CS


```
@using Syncfusion.Blazor.Grids
@{
    var Tool = (new List<string>() { "Search" });
}
<SfGrid DataSource="@Orders" Toolbar=@Tool>
    <GridColumn>
        <GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
            TextAlign="TextAlign.Right" Width="120"></GridColumn>
```

```

<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumn>
</SfGrid>
@code{
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
OrderDate = DateTime.Now.AddDays(-x),
}).ToList();
}
public class Order {
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
}

```

The following GIF image represents a DataGrid Searching.

| <div>Search </div> | | | |
|---|---------------|------------|---------|
| Order ID | Customer Name | Order Date | Freight |
| 1001 | ANANTR | 7/24/2019 | \$2.10 |
| 1002 | BLONP | 7/23/2019 | \$4.20 |
| 1003 | BLONP | 7/22/2019 | \$6.30 |
| 1004 | ANTON | 7/21/2019 | \$8.40 |
| 1005 | ALFKI | 7/20/2019 | \$10.50 |
| 1006 | ALFKI | 7/19/2019 | \$12.60 |
| 1007 | ALFKI | 7/18/2019 | \$14.70 |
| 1008 | BOLID | 7/17/2019 | \$16.80 |

Initial search

To apply search at initial rendering, set the **Fields**, **Operator**, **Key**, and **IgnoreCase** using [GridSearchSettings](#) component.

ASPX-CS

```

@using Syncfusion.Blazor.Grids
@{
    var InitSearch = (new string[] { "CustomerID" });
    var Tool = (new List<string>() { "Search" });
}
<SfGrid DataSource="@Orders" Toolbar=@Tool>
<GridSearchSettings Fields=@InitSearch
Operator=Syncfusion.Blazor.Operator.Contains Key="anton"
IgnoreCase="true"></GridSearchSettings>
<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumns>
</SfGrid>
@code{
    public List<Order> Orders { get; set; }
    protected override void OnInitialized()
    {
        Orders = Enumerable.Range(1, 75).Select(x => new Order()
        {
            OrderID = 1000 + x,
            CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
            })[new Random().Next(5)],
            Freight = 2.1 * x,
            OrderDate = DateTime.Now.AddDays(-x),
        }).ToList();
    }
    public class Order {
        public int? OrderID { get; set; }
        public string CustomerID { get; set; }
        public DateTime? OrderDate { get; set; }
        public double? Freight { get; set; }
    }
}

```

The following screenshot represents a DataGrid with initial searching.

| <input type="text" value="anton"/> | | | |
|------------------------------------|---------------|------------|---------|
| Order ID | Customer Name | Order Date | Freight |
| 1006 | ANTON | 7/19/2019 | \$12.60 |
| 1008 | ANTON | 7/17/2019 | \$16.80 |
| 1013 | ANTON | 7/12/2019 | \$27.30 |
| 1016 | ANTON | 7/9/2019 | \$33.60 |
| 1026 | ANTON | 6/29/2019 | \$54.60 |
| 1032 | ANTON | 6/23/2019 | \$67.20 |
| 1038 | ANTON | 6/17/2019 | \$79.80 |
| 1042 | ANTON | 6/13/2019 | \$88.20 |
| 1044 | ANTON | 6/11/2019 | \$92.40 |

By default, datagrid searches all the bound column values. To customize this behavior define the [Fields](#) property of **GridSearchSettings** component.

Search operators

The search operator can be defined in the [Operator](#) property of **GridSearchSettings** to configure specific searching.

The following operators are supported in searching:

Operator | Description

StartsWith | Checks whether a value begins with the specified value.

EndsWith | Checks whether a value ends with the specified value.

Contains | Checks whether a value contains the specified value.

Equal | Checks whether a value is equal to the specified value.

NotEqual | Checks for values not equal to the specified value.

By default, the [Operator](#) value is **Contains**.

Search by external button

To search datagrid records from an external button, invoke the **Search** method.

ASPX-CS

```
@using Syncfusion.Blazor.Buttons
@using Syncfusion.Blazor.Grids
@{
    var Tool = (new List<string>() { "Search" });
}
<SfButton Content="Search" OnClick="SearchBtnHandler"></SfButton>
<SfGrid @ref="DefaultGrid" DataSource="@Orders" AllowSorting="true"
    Toolbar=@Tool>
```

```

<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" TextAlign="TextAlign.Right" Width="130"
Type="ColumnType.Date"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumns>
</SfGrid>
@code{
private SfGrid<Order> DefaultGrid;
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
OrderDate = DateTime.Now.AddDays(-x),
}).ToList();
}
public class Order {
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
public void SearchBtnHandler()
{
this.DefaultGrid.Search("1001");
}
}

```

Search specific columns

By default, datagrid searches all visible columns. You can search specific columns by defining the specific column's field names in the [Fields](#) property of **GridSearchSettings** component.

ASPX-CS

```

@using Syncfusion.Blazor.Grids
@{
var Tool = (new List<string>() { "Search" });
var SpecificCols = (new string[] { "CustomerID","ShipCountry"});
}
<SfGrid DataSource="@Orders" Toolbar=@Tool>
<GridSearchSettings Fields=@SpecificCols></GridSearchSettings>
<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>

```

```

<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.ShipCountry) HeaderText="Ship Country"
Width="120"></GridColumn>
</GridColumn>
</SfGrid>
@code{
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
OrderDate = DateTime.Now.AddDays(-x),
ShipCountry = (new string[] { "USA", "UK", "CHINA", "RUSSIA", "INDIA" })[new
Random().Next(5)]
}).ToList();
}
public class Order {
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
public string ShipCountry { get; set; }
}
}

```

Disable search for particular column

By default, DataGrid searches all visible columns. You can disable searching for a particular column by setting the [AllowSearching](#) property of **GridColumn** as false.

In the below code example, the **Order ID** column search functionality is disabled.

ASPX-CS

```

@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders" Toolbar="@ (new List<string>() { "Search" })">
<GridColumn>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
TextAlign="TextAlign.Right" AllowSearching="false" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumn>
</SfGrid>
@code{

```

```

public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
    Orders = Enumerable.Range(1, 75).Select(x => new Order()
    {
        OrderID = 1000 + x,
        CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
    })[new Random().Next(5)],
        Freight = 2.1 * x,
        OrderDate = DateTime.Now.AddDays(-x),
    }).ToList();
}
public class Order {
    public int? OrderID { get; set; }
    public string CustomerID { get; set; }
    public DateTime? OrderDate { get; set; }
    public double? Freight { get; set; }
}
}

```

You can refer to our [Blazor DataGrid](#) feature tour page for its groundbreaking feature representations. You can also explore our [Blazor DataGrid example](#) to understand how to present and manipulate data.

Immediate Searching

By default, the datagrid will initiate searching operation after the Enter key is pressed. If you want to initiate the searching operation while typing the values in the search box, then you can invoke the Search method of the datagrid in the Input event of the SfTextBox.

ASPX-CS

```

@using Syncfusion.Blazor.Data
@using Syncfusion.Blazor.Grids
@using Syncfusion.Blazor.Inputs
@using Syncfusion.Blazor.Navigations
<SfGrid @ref="DefaultGrid" DataSource="@Orders" AllowSorting="true"
AllowFiltering="true" AllowPaging="true">
    <SfToolbar>
        <ToolbarItems>
            <ToolbarItem Type="ItemType.Input"
            Align="Syncfusion.Blazor.Navigations.ItemAlign.Right">
                <Template>
                    <SfTextBox Placeholder="Enter values to search" Input="OnInput"></SfTextBox>
                    <span class="e-search-icon e-icons"></span>
                </Template>
            </ToolbarItem>
        </ToolbarItems>
    </SfToolbar>
    <GridColumn>
        <GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
        IsPrimaryKey="true" TextAlign="@TextAlign.Center" Width="140"></GridColumn>
        <GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
        Width="150"></GridColumn>
        <GridColumn Field=@nameof(Order.Freight) HeaderText="Freight"
        Width="150"></GridColumn>
        <GridColumn Field=@nameof(Order.Verified) HeaderText="Freight"
        Width="150"></GridColumn>
    </GridColumn>

```



```

</GridColumns>
</SfGrid>
@code{
private SfGrid<Order> DefaultGrid;
public List<Order> Orders { get; set; }
public void OnInput(InputEventArgs args)
{
this.DefaultGrid.Search(args.Value);
}
public class Order
{
public int OrderID { get; set; }
public string CustomerID { get; set; }
public bool Verified { get; set; }
public double? Freight { get; set; }
}
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Verified = (new bool[] { true, false })[new Random().Next(2)],
Freight = 2.1 * x,
}).ToList();
}
}

```

Editing in Blazor DataGrid Component

The DataGrid component has options to dynamically insert, delete and update records.

[Editing](#) feature requires a primary key column for CRUD operations.

To know about editing feature in Blazor DataGrid component, you can check on this video.

{% youtube

"youtube:https://www.youtube.com/watch?v=jOiZpLexDB0"%}

To define the primary key, set [IsPrimaryKey](#) to **true** in particular column whose value is unique.

You can start the edit action either by double-clicking the particular row or by selecting the required row and click on the **Edit** button in the toolbar. Similarly, you can add a new record to DataGrid either by clicking on **Add** button in the toolbar or on an external button which is bound to invoke the [AddRecord](#) method of the DataGrid, **Save** and **Cancel** while in edit mode is possible using the respective toolbar icon in DataGrid.

Deletion of the record is possible by selecting the required row and click on **Delete** button in the toolbar.

ASPX-CS

```

@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders" AllowPaging="true" Toolbar="@ (new
List<string>() { "Add", "Edit", "Delete", "Cancel", "Update" })" Height
="315">

```

```

<GridEditSettings AllowAdding="true" AllowEditing="true"
AllowDeleting="true"></GridEditSettings>
<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
IsPrimaryKey="true" TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
EditType="EditType.DatePickerEdit" Format="d" TextAlign="TextAlign.Right"
Width="130" Type="ColumnType.Date"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumns>
</SfGrid>
@code{
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
OrderDate = DateTime.Now.AddDays(-x),
}).ToList();
}
public class Order {
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
}

```

The following screenshot represents Editing with Default Mode.

| + Add ✎ Edit 🗑 Delete ✕ Cancel 💾 Update | | | | |
|--|---------------|------------|---------|--|
| Order ID | Customer Name | Order Date | Freight | |
| 1001 | ANANTR | 7/21/2019 | \$2.10 | |
| 1002 | BLONP | 7/20/2019 | \$4.20 | |
| 1003 | ANTON | 7/19/2019 | \$6.30 | |
| 1004 | ANANTR | 7/18/2019 | \$8.40 | |
| 1005 | BOLID | 7/17/2019 | \$10.50 | |
| 1006 | ALFKI | 7/16/2019 | \$12.60 | |
| 1007 | BOLID | 7/15/2019 | \$14.70 | |
| 1008 | ANANTR | 7/14/2019 | \$16.80 | |
| 1009 | BLONP | 7/13/2019 | \$18.90 | |

⏮
<
1
2
3
4
5
6
7
>
⏭

1 of 7 pages (75 items)

* Grid uses `Activator.CreateInstance<TValue>()` to generate a new record when an insert operation is invoked, so it must have a parameterless constructor defined for the model class. To provide custom logic for object creation during editing, you can refer [here](#).

* If [IsIdentity](#) is enabled, then it will be considered as a read-only column when editing and adding a record.

* You can disable editing for a particular column, by specifying

[AllowEditing](#) to **false**.

* You can disable adding for a particular column, by specifying

[AllowAdding](#) to **false**.

* You can disable editing of a record on double click, by specifying

[EditSettings.AllowEditOnDbClick](#) to **false**.

Toolbar with edit option

The datagrid toolbar has the following built-in items to execute editing actions.

- Add - Adds a new record.
- Edit - Edits the selected record.
- Update - Updates the edited record.
- Delete - Deletes the selected record.
- Cancel - Cancels the edit state.

You can define this by using the [Toolbar](#) property.

ASPX-CS

```
@using Syncfusion.Blazor.Grids
```

```

<SfGrid DataSource="@Orders" AllowPaging="true" Toolbar="@ (new
List<string>() { "Add", "Edit", "Delete", "Cancel", "Update" })">
  <GridEditSettings AllowAdding="true" AllowEditing="true"
  AllowDeleting="true"></GridEditSettings>
  <GridColumns>
    <GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
    IsPrimaryKey="true" TextAlign="TextAlign.Right" Width="120"></GridColumn>
    <GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
    Width="120" ></GridColumn>
    <GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
    EditType="EditType.DatePickerEdit" Format="d" TextAlign="TextAlign.Right"
    Width="130" Type="ColumnType.Date"></GridColumn>
    <GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
    TextAlign="TextAlign.Right" Width="120" ></GridColumn>
  </GridColumns>
</SfGrid>
@code{
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
  Orders = Enumerable.Range(1, 75).Select(x => new Order()
  {
    OrderID = 1000 + x,
    CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
  })[new Random().Next(5)],
    Freight = 2.1 * x,
    OrderDate = DateTime.Now.AddDays(-x),
  }).ToList();
}
public class Order {
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
}

```

The following screenshot represents Toolbar with Edit option.

| + Add ✎ Edit 🗑 Delete ✕ Cancel 💾 Update | | | | |
|--|---------------|------------|---------|--|
| Order ID | Customer Name | Order Date | Freight | |
| 1001 | ANANTR | 7/21/2019 | \$2.10 | |
| 1002 | BLONP | 7/20/2019 | \$4.20 | |
| 1003 | ANTON | 7/19/2019 | \$6.30 | |
| 1004 | ANANTR | 7/18/2019 | \$8.40 | |
| 1005 | BOLID | 7/17/2019 | \$10.50 | |
| 1006 | ALFKI | 7/16/2019 | \$12.60 | |
| 1007 | BOLID | 7/15/2019 | \$14.70 | |
| 1008 | ANANTR | 7/14/2019 | \$16.80 | |
| 1009 | BLONP | 7/13/2019 | \$18.90 | |

⏮
<
1
2
3
4
5
6
7
>
⏭

1 of 7 pages (75 items)

Edit Modes

DataGrid supports the following types of edit modes, they are:

- Normal
- Dialog
- Batch

Normal

In Normal edit mode, when you start editing the currently selected record is changed to edit state. You can change the cell values and save edited data to the data source. To enable the Normal edit, set the [EditSettings.Mode](#) as **Normal**.

ASPX-CS

```
@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders" AllowPaging="true" Toolbar="@ (new
List<string>() { "Add", "Edit", "Delete", "Cancel", "Update" })"
Height="315">
  <GridEditSettings AllowAdding="true" AllowEditing="true"
  AllowDeleting="true" Mode="EditMode.Normal"></GridEditSettings>
  <GridColumns>
    <GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
    IsPrimaryKey="true" ValidationRules="@ (new ValidationRules{ Required=true})"
    TextAlign="TextAlign.Right" Width="120"></GridColumn>
    <GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
    ValidationRules="@ (new ValidationRules{ Required=true})"
    Width="120"></GridColumn>
    <GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
    EditType="EditType.DatePickerEdit" Format="d" TextAlign="TextAlign.Right"
    Width="130" Type="ColumnType.Date"></GridColumn>
```

```

<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" EditType="EditType.NumericEdit"
Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.ShipCountry) HeaderText="Ship Country"
EditType="EditType.DropDownEdit" Width="150"></GridColumn>
</GridColumn>
</SfGrid>
@code{
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
OrderDate = DateTime.Now.AddDays(-x),
ShipCountry = (new string[] { "USA", "UK", "CHINA", "RUSSIA", "INDIA" })[new
Random().Next(5)]
}).ToList();
}
public class Order
{
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
public string ShipCountry { get; set; }
}
}

```

The following screenshot represents Editing in Normal Mode.

| <div> + Add ✎ Edit 🗑 Delete ✕ Cancel 💾 Update </div> | | | | |
|---|------------------------------------|--|----------------------------------|-------------------------|
| Order ID | Customer Name | Order Date | Freight | |
| 1001 | ANTON | 7/21/2019 | \$2.10 | |
| 1002 | ALFKI | 7/20/2019 | \$4.20 | |
| 1003 | BLONP | 7/19/2019 | \$6.30 | |
| 1004 | <input type="text" value="ANTON"/> | <input type="text" value="7/18/2019"/> | <input type="text" value="8.4"/> | |
| 1005 | ANTON | 7/17/2019 | \$10.50 | |
| 1006 | ALFKI | 7/16/2019 | \$12.60 | |
| 1007 | BOLID | 7/15/2019 | \$14.70 | |
| 1008 | BOLID | 7/14/2019 | \$16.80 | |
| 1009 | ANANTR | 7/13/2019 | \$18.90 | |
| <div> ⏪ < 1 2 3 4 5 6 7 > ⏩ </div> | | | | 1 of 7 pages (75 items) |

Normal edit mode is the default mode of editing.

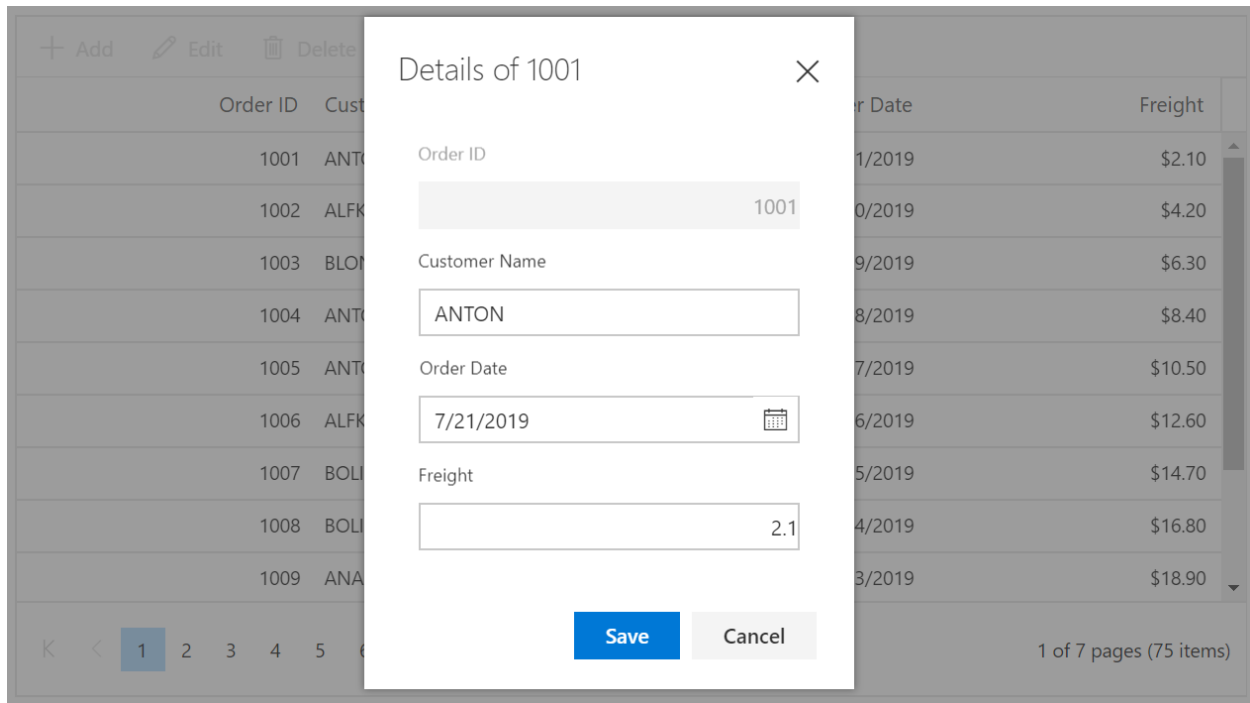
Dialog

In dialog edit mode, when you start editing the currently selected row data will be shown on a dialog. You can change the cell values and save edited data to the data source. To enable Dialog edit, set the [EditSettings.Mode](#) as **Dialog**.

ASPX-CS

```
@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders" AllowPaging="true" Toolbar="@ (new
List<string>() { "Add", "Edit", "Delete", "Update", "Cancel" })"
Height="315">
  <GridEditSettings AllowAdding="true" AllowEditing="true"
  AllowDeleting="true" Mode="EditMode.Dialog"></GridEditSettings>
  <GridColumns>
    <GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
    IsPrimaryKey="true" ValidationRules="@ (new ValidationRules{ Required=true})"
    TextAlign="TextAlign.Right" Width="120"></GridColumn>
    <GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
    ValidationRules="@ (new ValidationRules{ Required=true})"
    Width="120"></GridColumn>
    <GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
    EditType="EditType.DatePickerEdit" Format="d" TextAlign="TextAlign.Right"
    Width="130" Type="ColumnType.Date"></GridColumn>
    <GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
    TextAlign="TextAlign.Right" EditType="EditType.NumericEdit"
    Width="120"></GridColumn>
    <GridColumn Field=@nameof(Order.ShipCountry) HeaderText="Ship Country"
    EditType="EditType.DropDownEdit" Width="150"></GridColumn>
  </GridColumns>
</SfGrid>
@code{
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
  Orders = Enumerable.Range(1, 75).Select(x => new Order()
  {
    OrderID = 1000 + x,
    CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
  })[new Random().Next(5)],
    Freight = 2.1 * x,
    OrderDate = DateTime.Now.AddDays(-x),
    ShipCountry = (new string[] { "USA", "UK", "CHINA", "RUSSIA", "INDIA" })[new
    Random().Next(5)]
  }).ToList();
}
public class Order
{
  public int? OrderID { get; set; }
  public string CustomerID { get; set; }
  public DateTime? OrderDate { get; set; }
  public double? Freight { get; set; }
  public string ShipCountry { get; set; }
}
```

The following screenshot represents Editing in Dialog Mode.



Batch

In batch edit mode, when you double-click on the datagrid cell, then the target cell changed to edit state. You can bulk save (added, changed, and deleted data in the single request) to the data source by click on the toolbar's **Update** button or by externally calling the **EndEdit** method. To enable Batch edit, set the [EditSettings.Mode](#) as **Batch**.

ASPX-CS

```
@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders" AllowPaging="true" Toolbar="@ (new
List<string>() { "Add", "Delete", "Update", "Cancel" })" Height="315">
<GridEditSettings AllowAdding="true" AllowEditing="true"
AllowDeleting="true" Mode="EditMode.Batch"></GridEditSettings>
<GridColumn>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
IsPrimaryKey="true" TextAlign="TextAlign.Right" ValidationRules="@ (new
ValidationRules { Required = true })" Type="ColumnType.Number"
Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
ValidationRules="@ (new ValidationRules{ Required=true})"
Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
EditType="EditType.DatePickerEdit" Format="d" TextAlign="TextAlign.Right"
Width="130" Type="ColumnType.Date"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" EditType="EditType.NumericEdit"
Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.ShipCountry) HeaderText="Ship Country"
EditType="EditType.DropDownEdit" Width="150"></GridColumn>
</GridColumn>
</SfGrid>
@code{
public List<Order> Orders { get; set; }
```



```
protected override void OnInitialized()
{
    Orders = Enumerable.Range(1, 75).Select(x => new Order()
    {
        OrderID = 1000 + x,
        CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
    })[new Random().Next(5)],
        Freight = 2.1 * x,
        OrderDate = DateTime.Now.AddDays(-x),
        ShipCountry = (new string[] { "USA", "UK", "CHINA", "RUSSIA", "INDIA" })[new
        Random().Next(5)]
    }).ToList();
}

public class Order
{
    public int? OrderID { get; set; }
    public string CustomerID { get; set; }
    public DateTime? OrderDate { get; set; }
    public double? Freight { get; set; }
    public string ShipCountry { get; set; }
}
}
```

The following screenshot represents Editing in Batch mode.

+

Add

Edit

Delete

Cancel

Update

| Order ID | Customer Name | Order Date | Freight |
|----------|---------------|------------|---------|
| 1001 | ANTON | 7/21/2019 | \$2.10 |
| 1002 | VINET | 7/20/2019 | \$4.20 |
| 1003 | BLONP | 7/19/2019 | \$6.30 |
| 1004 | ANTON | 7/18/2019 | \$8.40 |
| 1005 | ANTON | 7/17/2019 | \$10.50 |
| 1006 | ALFKI | 7/16/2019 | \$12.60 |
| 1007 | BOLID | 7/15/2019 | \$14.70 |
| 1008 | BOLID | 7/14/2019 | \$16.80 |
| 1009 | ANANTR | 7/13/2019 | \$18.90 |

K

<

1

2

3

4

5

6

7

>

X

1 of 7 pages (75 items)

Edit next row or previous row from the current row

You can continue editing the next row or previous row from the current record in batch mode by enabling [EditSettings.AllowNextRowEdit](#) to **true**.

Pressing **TAB** from the last cell of the current row allows editing the next row and Pressing **SHIFT + TAB** from the first cell of the current row allows editing the previous row.





ASPX-CS





```

@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders" AllowPaging="true" Toolbar="@ (new
List<string>() { "Add", "Delete", "Update", "Cancel" })" Height="315">
<GridEditSettings AllowAdding="true" AllowEditing="true"
AllowDeleting="true" Mode="EditMode.Batch"></GridEditSettings>
<GridColumn>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
IsPrimaryKey="true" TextAlign="TextAlign.Right" ValidationRules="@ (new
ValidationRules { Required = true })" Type="ColumnType.Number"
Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
ValidationRules="@ (new ValidationRules{ Required=true})"
Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
EditType="EditType.DatePickerEdit" Format="d" TextAlign="TextAlign.Right"
Width="130" Type="ColumnType.Date"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" EditType="EditType.NumericEdit"
Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.ShipCountry) HeaderText="Ship Country"
EditType="EditType.DropDownEdit" Width="150"></GridColumn>
</GridColumn>
</SfGrid>
@code{
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
OrderDate = DateTime.Now.AddDays(-x),
ShipCountry = (new string[] { "USA", "UK", "CHINA", "RUSSIA", "INDIA" })[new
Random().Next(5)]
}).ToList();
}
public class Order
{
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
public string ShipCountry { get; set; }
}
}

```

The following GIF represents Editing in Batch mode.

|  Add |  Delete |  Update |  Cancel | | |
|---|--|--|--|--------------|--|
| Order ID | Customer Name | Order Date | Freight | Ship Country | |
| 1001 | ALFKI | 14-12-2020 | ₹ 2.10 | RUSSIA | |
| 1002 | <input type="text" value="BLONP"/> | 13-12-2020 | ₹ 4.20 | CHINA | |
| 1003 | ANANTR | 12-12-2020 | ₹ 6.30 | CHINA | |
| 1004 | BLONP | 11-12-2020 | ₹ 8.40 | INDIA | |
| 1005 | ALFKI | 10-12-2020 | ₹ 10.50 | RUSSIA | |
| 1006 | BLONP | 09-12-2020 | ₹ 12.60 | CHINA | |
| 1007 | ANANTR | 08-12-2020 | ₹ 14.70 | CHINA | |
| 1008 | BOLID | 07-12-2020 | ₹ 16.80 | RUSSIA | |
| 1009 | ANANTR | 06-12-2020 | ₹ 18.90 | RUSSIA | |



1
2
3
4
5
6
7



1 of 7 pages (75 items)

Cell edit type

The [EditType](#) property of the [GridColumn](#) component is used for defining the editor component for any particular column. You can set the [EditType](#) based on the data type of the column.

The available default edit types are,

- [NumericEdit](#) component for integers, double, and decimal data types.
- [DefaultEdit](#) component for string data type.
- [DropDownEdit](#) component to show all unique values related to that field.
- [BooleanEdit](#) component for boolean data type.
- [DatePickerEdit](#) component for date data type.
- [DateTimePickerEdit](#) component for date time data type.

Customizing the default editor controls

You can customize the behavior of the editor component through the [EditorSettings](#) property of the [GridColumn](#) component.

We have limited the properties of editor components that can be customized using [EditorSettings](#) in Grid default editor components. Kindly find the list of properties that can be customized the below topics.

If you want to customize other properties, refer to our [EditTemplate](#) documentation to render the custom components in EditForm along with your customization.

DefaultEdit

[StringEditCellParams](#) class helps us to customize the default [TextBox](#) component in Grid EditForm. The following table describes properties of [TextBox](#) control than can be customized using [EditorSettings](#) of [GridColumn](#) editor component.

| Component | Description |
|-----------|-------------|
|-----------|-------------|

- |-----|-----|
- | CssClass | Specifies the CSS class value that is appended to wrapper of Textbox. |
 - | EnableRtl | Enable or disable rendering component in right to left direction. |
 - | ReadOnly | Specifies the boolean value whether the TextBox allows user to change the text. |
 - | ShowClearButton | Specifies a Boolean value that indicates whether the clear button is displayed in Textbox. |
 - | Multiline | Specifies a boolean value that enable or disable the multiline on the TextBox. The TextBox changes from single line to multiline when enable this multiline mode. |

The following sample code demonstrates the customization applied to TextBox component set for the DataGrid columns,

ASPX-CS

```
@using Syncfusion.Blazor.Inputs
@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@OrderData" Toolbar=@ToolbarItems>
<GridEditSettings AllowEditing="true" AllowAdding="true"
AllowDeleting="true"></GridEditSettings>
<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
IsPrimaryKey="true" TextAlign="TextAlign.Center" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer ID"
EditType="EditType.DefaultEdit" EditorSettings="@CustomerEditParams"
TextAlign="TextAlign.Center" Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
EditType="EditType.NumericEdit" TextAlign="TextAlign.Center"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.ShipName) HeaderText="Ship Name"
TextAlign="TextAlign.Center" EditType="EditType.DropDownEdit"
Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.Verified) HeaderText="Verified"
EditType="EditType.BooleanEdit" TextAlign="TextAlign.Center"
DisplayAsCheckBox="true" Width="120"></GridColumn>
</GridColumns>
</SfGrid>
@code{
public string[] ToolbarItems = new string[] { "Add", "Edit", "Delete",
"Update", "Cancel" };
public IEditorSettings CustomerEditParams = new StringEditCellParams
{
Params = new TextBoxModel() { EnableRtl = true, ShowClearButton = false,
Multiline = true }
};
List<Order> OrderData = new List<Order>
{
new Order() { OrderID = 10248, CustomerID = "VINET", Freight = 32.38,
ShipName = "Vins et alcools Chevalier", Verified = true },
new Order() { OrderID = 10249, CustomerID = "TOMSP", Freight = 11.61,
ShipName = "Toms Spezialitäten", Verified = false },
new Order() { OrderID = 10250, CustomerID = "HANAR", Freight = 65.83,
ShipName = "Hanari Carnes", Verified = true },
```

```

new Order() { OrderID = 10251, CustomerID = "VICTE", Freight = 41.34,
ShipName = "Victuailles en stock", Verified = false },
new Order() { OrderID = 10252, CustomerID = "SUPRD", Freight = 51.3,
ShipName = "Suprêmes délices", Verified = false },
new Order() { OrderID = 10253, CustomerID = "HANAR", Freight = 58.17,
ShipName = "Hanari Carnes", Verified = false },
new Order() { OrderID = 10254, CustomerID = "CHOPS", Freight = 22.98,
ShipName = "Chop-suey Chinese", Verified = true },
new Order() { OrderID = 10255, CustomerID = "RICSU", Freight = 148.33,
ShipName = "Richter Supermarket", Verified = true },
new Order() { OrderID = 10256, CustomerID = "WELLI", Freight = 13.97,
ShipName = "Wellington Importadora", Verified = false },
new Order() { OrderID = 10257, CustomerID = "HILAA", Freight = 81.91,
ShipName = "HILARION-Abastos", Verified = true }
};
public class Order
{
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public double Freight { get; set; }
public string ShipName { get; set; }
public bool Verified { get; set; }
}
}

```

NumericEdit

NumericEditCellParams class helps us to customize the default **NumericTextBox** component in **Grid EditForm**. The following table describes properties of **NumericTextBox** control than can be customized using **EditorSettings** of **GridColumn** editor component.

| Component | Description |
|-----------------------|--|
| ----- ----- | |
| CssClass | Gets or Sets the CSS classes to root element of the NumericTextBox which helps to customize the complete UI styles for the NumericTextBox component. |
| Decimals | Specifies the number precision applied to the textbox value when the NumericTextBox is focused. |
| EnableRtl | Enable or disable rendering component in right to left direction. |
| Format | Specifies the number format that indicates the display format for the value of the NumericTextBox . |
| Placeholder | Gets or sets the string shown as a hint/placeholder when the NumericTextBox is empty. It acts as a label and floats above the NumericTextBox . |
| ShowClearButton | Specifies whether to show or hide the clear icon. |
| ShowSpinButton | Specifies whether the up and down spin buttons should be displayed in NumericTextBox . |
| ValidateDecimalOnType | Specifies whether the decimals length should be restricted during typing. |

The following sample code demonstrates the customization applied to **NumericTextBox** component set for the **DataGrid** columns,

ASPX-CS

```

@using Syncfusion.Blazor.Grids
@using Syncfusion.Blazor.Inputs
<SfGrid DataSource="@OrderData" Toolbar=@ToolbarItems>
<GridEditSettings AllowEditing="true" AllowAdding="true"
AllowDeleting="true"></GridEditSettings>
<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
IsPrimaryKey="true" TextAlign="TextAlign.Center" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer ID"
TextAlign="TextAlign.Center" Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
EditType="EditType.NumericEdit" EditorSettings="@FreightEditParams"
TextAlign="TextAlign.Center" Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.ShipName) HeaderText="Ship Name"
TextAlign="TextAlign.Center" EditType="EditType.DropDownEdit"
Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.Verified) HeaderText="Verified"
EditType="EditType.BooleanEdit" TextAlign="TextAlign.Center"
DisplayAsCheckBox="true" Width="120"></GridColumn>
</GridColumns>
</SfGrid>
@code{
public string[] ToolbarItems = new string[] { "Add", "Edit", "Delete",
"Update", "Cancel" };
public IEditorSettings FreightEditParams = new NumericEditCellParams
{
Params = new NumericTextBoxModel<object>() { ShowClearButton= true,
ShowSpinButton = false }
};
List<Order> OrderData = new List<Order>
{
new Order() { OrderID = 10248, CustomerID = "VINET", Freight = 32.38,
ShipName = "Vins et alcools Chevalier", Verified = true },
new Order() { OrderID = 10249, CustomerID = "TOMSP", Freight = 11.61,
ShipName = "Toms Spezialitäten", Verified = false },
new Order() { OrderID = 10250, CustomerID = "HANAR", Freight = 65.83,
ShipName = "Hanari Carnes", Verified = true },
new Order() { OrderID = 10251, CustomerID = "VICTE", Freight = 41.34,
ShipName = "Victuailles en stock", Verified = false },
new Order() { OrderID = 10252, CustomerID = "SUPRD", Freight = 51.3,
ShipName = "Suprêmes délices", Verified = false },
new Order() { OrderID = 10253, CustomerID = "HANAR", Freight = 58.17,
ShipName = "Hanari Carnes", Verified = false },
new Order() { OrderID = 10254, CustomerID = "CHOPS", Freight = 22.98,
ShipName = "Chop-suey Chinese", Verified = true },
new Order() { OrderID = 10255, CustomerID = "RICSU", Freight = 148.33,
ShipName = "Richter Supermarket", Verified = true },
new Order() { OrderID = 10256, CustomerID = "WELLI", Freight = 13.97,
ShipName = "Wellington Importadora", Verified = false },
new Order() { OrderID = 10257, CustomerID = "HILAA", Freight = 81.91,
ShipName = "HILARION-Abastos", Verified = true }
};
public class Order
{
public int? OrderID { get; set; }

```

```
public string CustomerID { get; set; }
public double Freight { get; set; }
public string ShipName { get; set; }
public bool Verified { get; set; }
}
```

DropDownEdit

DropDownEditCellParams class helps us to customize the default DropDownList component in Grid EditForm. The following table describes properties of DropDownList control that can be customized using **EditorSettings** of GridColumn editor component.

| Component | Example |
|-----------------------|--|
| ----- ----- | |
| Enabled | Specifies a value that indicates whether the component is enabled or not |
| CssClass | Sets CSS classes to the root element of the component that allows customization of appearance. |
| FilterBarPlaceholder | Accepts the value to be displayed as a watermark text on the filter bar. |
| AllowFiltering | When allowFiltering is set to true, show the filter bar (search box) of the component. |
| FooterTemplate | Accepts the template design and assigns it to the footer container of the popup list. |
| HeaderTemplate | Accepts the template design and assigns it to the header container of the popup list. |
| PopupHeight | Specifies the height of the popup list |
| PopupWidth | Specifies the width of the popup list. By default, the popup width sets based on the width of the component |
| ReadOnly | When set to true, the user interactions on the component are disabled. |
| ShowClearButton | Specifies whether to show or hide the clear button. When the clear button is clicked, Value , text , and index properties are reset to null. |
| ValueTemplate | Accepts the template design and assigns it to the selected list item in the input element of the component. |
| ActionFailureTemplate | Accepts the template and assigns it to the popup list content of the component when the data fetch request from the remote server fails |
| DataSource | Accepts the list items either through local or remote service and binds it to the component. It can be an array of JSON Objects or an instance of DataManager . |

IEnumerable<TItem> is the type of **DataSource** property in **DropDownListModel**, so you should not bind **string[]** or **List<string>** type to the **DataSource** property.

The following sample code demonstrates the customization applied to DropDownList component set for the DataGrid columns,

ASPX-CS

```

@using Syncfusion.Blazor.DropDowns
@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@OrderData" Toolbar=@ToolbarItems>
<GridEditSettings AllowEditing="true" AllowAdding="true"
AllowDeleting="true"></GridEditSettings>
<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
IsPrimaryKey="true" TextAlign="TextAlign.Center" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer ID"
TextAlign="TextAlign.Center" Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
EditType="EditType.NumericEdit" TextAlign="TextAlign.Center"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.ShipName) HeaderText="Ship Name"
TextAlign="TextAlign.Center" EditType="EditType.DropDownEdit"
EditorSettings="@ShipNameEditParams" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.Verified) HeaderText="Verified"
EditType="EditType.BooleanEdit" TextAlign="TextAlign.Center"
DisplayAsCheckBox="true" Width="120"></GridColumn>
</GridColumns>
</SfGrid>
@code{
public string[] ToolbarItems = new string[] { "Add", "Edit", "Delete",
"Update", "Cancel" };
public IEditorSettings ShipNameEditParams = new DropDownEditCellParams
{
Params = new DropDownListModel<object, object>() { AllowFiltering = true,
ShowClearButton = true }
};
List<Order> OrderData = new List<Order>
{
new Order() { OrderID = 10248, CustomerID = "VINET", Freight = 32.38,
ShipName = "Vins et alcools Chevalier", Verified = true },
new Order() { OrderID = 10249, CustomerID = "TOMSP", Freight = 11.61,
ShipName = "Toms Spezialitäten", Verified = false },
new Order() { OrderID = 10250, CustomerID = "HANAR", Freight = 65.83,
ShipName = "Hanari Carnes", Verified = true },
new Order() { OrderID = 10251, CustomerID = "VICTE", Freight = 41.34,
ShipName = "Victuailles en stock", Verified = false },
new Order() { OrderID = 10252, CustomerID = "SUPRD", Freight = 51.3,
ShipName = "Suprêmes délices", Verified = false },
new Order() { OrderID = 10253, CustomerID = "HANAR", Freight = 58.17,
ShipName = "Hanari Carnes", Verified = false },
new Order() { OrderID = 10254, CustomerID = "CHOPS", Freight = 22.98,
ShipName = "Chop-suey Chinese", Verified = true },
new Order() { OrderID = 10255, CustomerID = "RICSU", Freight = 148.33,
ShipName = "Richter Supermarket", Verified = true },
new Order() { OrderID = 10256, CustomerID = "WELLI", Freight = 13.97,
ShipName = "Wellington Importadora", Verified = false },
new Order() { OrderID = 10257, CustomerID = "HILAA", Freight = 81.91,
ShipName = "HILARION-Abastos", Verified = true }
};
public class Order
{
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public double Freight { get; set; }
}

```



```
public string ShipName { get; set; }
public bool Verified { get; set; }
}
```

BooleanEdit

BooleanEditCellParams class helps us to customize the default Checkbox component in Grid EditForm. The following table describes properties of CheckBox control than can be customized using **EditorSettings** of GridColumn editor component.

| Component | Description |
|---------------|--|
| CssClass | Defines class/multiple classes separated by a space in the CheckBox element. You can add custom styles to the CheckBox by using this property. |
| Label | Defines the caption for the CheckBox, that describes the purpose of the CheckBox. |
| EnableRtl | Enable or disable rendering component in right to left direction. |
| LabelPosition | Positions label Before/after the CheckBox. The possible values are: Before - The label is positioned to left of the CheckBox. After - The label is positioned to right of the CheckBox. |
| Indeterminate | Specifies a value that indicates whether the CheckBox is in Indeterminate state or not. When set to true , the CheckBox will be in indeterminate state. |
| Disabled | Specifies a value that indicates whether the CheckBox is Disabled or not. When set to true , the CheckBox will be in disabled state. |

The following sample code demonstrates the customization applied to Checkbox component set for the DataGrid columns,

ASPX-CS

```
@using Syncfusion.Blazor.Buttons
@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@OrderData" Toolbar=@ToolbarItems>
  <GridEditSettings AllowEditing="true" AllowAdding="true"
  AllowDeleting="true"></GridEditSettings>
  <GridColumn>
    <GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
    IsPrimaryKey="true" TextAlign="TextAlign.Center" Width="120"></GridColumn>
    <GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer ID"
    TextAlign="TextAlign.Center" Width="130"></GridColumn>
    <GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
    EditType="EditType.NumericEdit" TextAlign="TextAlign.Center"
    Width="130"></GridColumn>
    <GridColumn Field=@nameof(Order.ShipName) HeaderText="Ship Name"
    TextAlign="TextAlign.Center" EditType="EditType.DropDownEdit"
    Width="120"></GridColumn>
    <GridColumn Field=@nameof(Order.Verified) HeaderText="Verified"
    EditType="EditType.BooleanEdit" TextAlign="TextAlign.Center"
    DisplayAsCheckBox="true" Width="120"
    EditorSettings="@VerifiedEditParams"></GridColumn>
  </GridColumn>
</SfGrid>
```

```

@code{
public string[] ToolbarItems = new string[] { "Add", "Edit", "Delete",
"Update", "Cancel" };
public IEditorSettings VerifiedEditParams = new BooleanEditCellParams
{
Params = new CheckBoxModel<bool>() { Label = "Checked", Disabled = true,
LabelPosition = LabelPosition.Before }
};
List<Order> OrderData = new List<Order>
{
new Order() { OrderID = 10248, CustomerID = "VINET", Freight = 32.38,
ShipName = "Vins et alcools Chevalier", Verified = true },
new Order() { OrderID = 10249, CustomerID = "TOMSP", Freight = 11.61,
ShipName = "Toms Spezialitäten", Verified = false },
new Order() { OrderID = 10250, CustomerID = "HANAR", Freight = 65.83,
ShipName = "Hanari Carnes", Verified = true },
new Order() { OrderID = 10251, CustomerID = "VICTE", Freight = 41.34,
ShipName = "Victuailles en stock", Verified = false },
new Order() { OrderID = 10252, CustomerID = "SUPRD", Freight = 51.3,
ShipName = "Suprêmes délices", Verified = false },
new Order() { OrderID = 10253, CustomerID = "HANAR", Freight = 58.17,
ShipName = "Hanari Carnes", Verified = false },
new Order() { OrderID = 10254, CustomerID = "CHOPS", Freight = 22.98,
ShipName = "Chop-suey Chinese", Verified = true },
new Order() { OrderID = 10255, CustomerID = "RICSU", Freight = 148.33,
ShipName = "Richter Supermarket", Verified = true },
new Order() { OrderID = 10256, CustomerID = "WELLI", Freight = 13.97,
ShipName = "Wellington Importadora", Verified = false },
new Order() { OrderID = 10257, CustomerID = "HILAA", Freight = 81.91,
ShipName = "HILARION-Abastos", Verified = true }
};
public class Order
{
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public double Freight { get; set; }
public string ShipName { get; set; }
public bool Verified { get; set; }
}
}

```

[DatePickerEdit](#)

DateEditCellParams class helps us to customize the default DatePicker and DateTimePicker component in Grid EditForm. The following table describes properties of DatePicker control than can be customized using **EditorSettings** of GridColumn editor component.

| Component | Example |

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

| CssClass | Specifies the root CSS class of the DatePicker that allows to customize the appearance by overriding the styles. |

| EnableRtl | Enable or disable rendering component in right to left direction. |

| ReadOnly | Specifies the component in readonly state. When the Component is readonly it does not allow user input. |

| ShowClearButton | Specifies whether to show or hide the clear icon in textbox. |

The following sample code demonstrates the customization applied to DatePicker component set for the DataGrid columns,

ASPX-CS

```
@using Syncfusion.Blazor.Calendars
@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@OrderData" Toolbar=@ToolbarItems>
<GridEditSettings AllowEditing="true" AllowAdding="true"
AllowDeleting="true"></GridEditSettings>
<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
IsPrimaryKey="true" TextAlign="TextAlign.Center" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer ID"
TextAlign="TextAlign.Center" Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
EditType="EditType.NumericEdit" TextAlign="TextAlign.Center"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText="OrderDate" Format="d"
EditType="EditType.DatePickerEdit" EditorSettings="@DateEditParams"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.ShipName) HeaderText="Ship Name"
TextAlign="TextAlign.Center" EditType="EditType.DropDownEdit"
Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.Verified) HeaderText="Verified"
EditType="EditType.BooleanEdit" TextAlign="TextAlign.Center"
DisplayAsCheckBox="true" Width="120"></GridColumn>
</GridColumns>
</SfGrid>
@code{
public string[] ToolbarItems = new string[] { "Add", "Edit", "Delete",
"Update", "Cancel" };
public IEditorSettings DateEditParams = new DateEditCellParams
{
Params = new DatePickerModel() { EnableRtl = true, ShowClearButton = false }
};
List<Order> OrderData = new List<Order>
{
new Order() { OrderID = 10248, CustomerID = "VINET", Freight = 32.38,
ShipName = "Vins et alcools Chevalier", Verified = true },
new Order() { OrderID = 10249, CustomerID = "TOMSP", Freight = 11.61,
ShipName = "Toms Spezialitäten", Verified = false },
new Order() { OrderID = 10250, CustomerID = "HANAR", Freight = 65.83,
ShipName = "Hanari Carnes", Verified = true },
new Order() { OrderID = 10251, CustomerID = "VICTE", Freight = 41.34,
ShipName = "Victuailles en stock", Verified = false },
new Order() { OrderID = 10252, CustomerID = "SUPRD", Freight = 51.3,
ShipName = "Suprêmes délices", Verified = false },
new Order() { OrderID = 10253, CustomerID = "HANAR", Freight = 58.17,
ShipName = "Hanari Carnes", Verified = false },
new Order() { OrderID = 10254, CustomerID = "CHOPS", Freight = 22.98,
ShipName = "Chop-suey Chinese", Verified = true },
```

```

new Order() { OrderID = 10255, CustomerID = "RICSU", Freight = 148.33,
ShipName = "Richter Supermarket", Verified = true },
new Order() { OrderID = 10256, CustomerID = "WELLI", Freight = 13.97,
ShipName = "Wellington Importadora", Verified = false },
new Order() { OrderID = 10257, CustomerID = "HILAA", Freight = 81.91,
ShipName = "HILARION-Abastos", Verified = true }
};
public class Order
{
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public double Freight { get; set; }
public DateTime? OrderDate { get; set; } = DateTime.Now;
public string ShipName { get; set; }
public bool Verified { get; set; }
}
}

```

Similar way customization can be applied to default DateTimePicker Component using same **DateEditCellParams**

Cell Edit Template

Before adding edit template to the datagrid, we strongly recommend you to go through the [template](#) section topic to configure the template.

The cell edit template is used to add a custom component for a particular column. You can use the **EditTemplate** of the [GridColumn](#) component to add the custom component. You can access the parameters passed to the templates using implicit parameter named **context**.

Custom components inside the EditTemplate must be specified with two-way (**@bind-Value**) binding to reflect the changes in DataGrid.

Using AutoComplete in EditTemplate

You can able to render SfAutoComplete component in EditTemplate. In the below sample we have rendered **SfAutoComplete** component in **EditTemplate** for Customer ID column.

ASPX-CS

```

@using Syncfusion.Blazor.DropDowns
@using Syncfusion.Blazor.Grids
<SfGrid AllowPaging="true" DataSource="@Orders" Toolbar="@ (new
List<string>() { "Add", "Edit", "Delete", "Cancel", "Update" }) ">
<GridEditSettings AllowEditing="true" AllowDeleting="true"
AllowAdding="true" Mode="@EditMode.Normal"></GridEditSettings>
<GridColumn>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
IsPrimaryKey="true" TextAlign="@TextAlign.Center" Width="140"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150">
<EditTemplate>
<SfAutoComplete ID="CustomerID" TItem="Order" TValue="string" @bind-
Value="@((context as Order).CustomerID)" DataSource="@Orders">
<AutoCompleteFieldSettings Value="CustomerID"></AutoCompleteFieldSettings>
</SfAutoComplete>
</EditTemplate>
</GridColumn>

```

```

<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight"
EditType="EditType.NumericEdit" Format="C2" Width="140"
TextAlign="@TextAlign.Right"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText="Order Date"
EditType="EditType.DatePickerEdit" Format="d" Type="ColumnType.Date"
Width="160"></GridColumn>
</GridColumn>
</SfGrid>
@code{
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
OrderDate = DateTime.Now.AddDays(-x),
}).ToList();
}
public class Order
{
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
}

```

In the following image, **Autocomplete** component is rendered with **EditTemplate** in Customer ID column

| + Add ✎ Edit 🗑 Delete ✕ Cancel 💾 Update | | | | |
|--|--------------------------------|---------|------------|--|
| Order ID | Customer Name | Freight | Order Date | |
| 1001 | BLONP | \$2.10 | 7/29/2019 | |
| 1002 | <input type="text" value="a"/> | 4.20 | 7/28/2019 | |
| 1003 | ALFKI | \$6.30 | 7/27/2019 | |
| 1004 | ANTON | \$8.40 | 7/26/2019 | |
| 1005 | ANTON | \$10.50 | 7/25/2019 | |
| 1006 | ANANTR | \$12.60 | 7/24/2019 | |
| 1007 | ANTON | \$14.70 | 7/23/2019 | |
| 1008 | ANANTR | \$16.80 | 7/22/2019 | |
| 1009 | ANTON | \$18.90 | 7/21/2019 | |
| 1010 | BOLID | \$21.00 | 7/20/2019 | |
| 1011 | ANANTR | \$23.10 | 7/19/2019 | |

Using DropDownList in EditTemplate

You can able to render SfDropDownList component in EditTemplate. In the below sample we have rendered **SfDropDownList** component in **EditTemplate** for ShipCountry column.

ASPX-CS

```
@using Syncfusion.Blazor.Grids
@using Syncfusion.Blazor.DropDowns
<SfGrid DataSource="@Orders" AllowPaging="true" Toolbar="@ (new
List<string>() { "Add", "Edit", "Delete", "Cancel", "Update" })"
Height="315">
  <GridEditSettings AllowAdding="true" AllowEditing="true"
  AllowDeleting="true" Mode="EditMode.Normal"></GridEditSettings>
  <GridColumns>
    <GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
    IsPrimaryKey="true" ValidationRules="@ (new ValidationRules{ Required=true})"
    TextAlign="TextAlign.Right" Width="120"></GridColumn>
    <GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
    ValidationRules="@ (new ValidationRules{ Required=true})"
    Width="120"></GridColumn>
    <GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
    EditType="EditType.DatePickerEdit" Format="d" TextAlign="TextAlign.Right"
    Width="130" Type="ColumnType.Date"></GridColumn>
    <GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
    TextAlign="TextAlign.Right" EditType="EditType.NumericEdit"
    Width="120"></GridColumn>
    <GridColumn Field=@nameof(Order.ShipCountry) HeaderText="Ship Country"
    EditType="EditType.DropDownEdit" Width="150">
      <EditTemplate>
        <SfDropDownList ID="ShipCountry" TItem="Country" TValue="string" @bind-
        Value="@ ((context as Order).ShipCountry)" DataSource="@Countries">
          <DropDownListFieldSettings Value="CountryName"
          Text="CountryName"></DropDownListFieldSettings>
        </SfDropDownList>
      </EditTemplate>
    </GridColumn>
  </GridColumns>
</SfGrid>

@code{
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
  Orders = Enumerable.Range(1, 75).Select(x => new Order()
  {
    OrderID = 1000 + x,
    CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
  })[new Random().Next(5)],
    Freight = 2.1 * x,
    OrderDate = DateTime.Now.AddDays(-x),
    ShipCountry = (new string[] { "USA", "UK", "CHINA", "RUSSIA", "INDIA" })[new
    Random().Next(5)]
  }).ToList();
}
public class Order
{
  public int? OrderID { get; set; }
  public string CustomerID { get; set; }
```

```

public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
public string ShipCountry { get; set; }
}
public List<Country> Countries { get; set; } = new List<Country>()
{
    new Country(){ CountryName="Brazil", ID=1},
    new Country(){ CountryName="Argentina", ID=2},
    new Country(){ CountryName="Canada", ID=3}
};
public class Country
{
    public string CountryName { get; set; }
    public int ID { get; set; }
}
}

```

In the following image, **SfDropDownList** component is rendered with **EditTemplate** in ShipCountry column

| Order ID | Customer Name | Order Date | Freight | Ship Country |
|----------|---------------|------------|---------|------------------|
| 1001 | BOLID | 5/23/2021 | \$2.10 | RUSSIA |
| 1002 | BOLID | 5/22/2021 | \$4.20 | RUSSIA |
| 1003 | ALFKI | 5/21/2021 | \$6.30 | INDIA |
| 1004 | BLONP | 5/20/2021 | \$8.40 | RUSSIA |
| 1,005 | BOLID | 5/19/2021 | 10.50 | Select a country |
| 1006 | BLONP | 5/18/2021 | \$12.60 | Brazil |
| 1007 | BOLID | 5/17/2021 | \$14.70 | Argentina |
| 1008 | ALFKI | 5/16/2021 | \$16.80 | Canada |
| 1009 | BOLID | 5/15/2021 | \$18.90 | USA |

Using TimePicker in EditTemplate

You can able to render SfTimePicker component in EditTemplate. In the below sample we have rendered **SfTimePicker** component in **EditTemplate** for OrderDate column.

ASPX-CS

```

@using Syncfusion.Blazor.Grids
@using Syncfusion.Blazor.Calendars
<SfGrid DataSource="@Orders" AllowPaging="true" Toolbar="@ (new
List<string>() { "Add", "Edit", "Delete", "Cancel", "Update" })"
Height="315">
    <GridEditSettings AllowAdding="true" AllowEditing="true"
    AllowDeleting="true" Mode="EditMode.Normal"></GridEditSettings>
    <GridColumns>
        <GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
        IsPrimaryKey="true" ValidationRules="@ (new ValidationRules{ Required=true})"
        TextAlign="TextAlign.Right" Width="120"></GridColumn>
        <GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
        ValidationRules="@ (new ValidationRules{ Required=true})"
        Width="120"></GridColumn>
    </GridColumns>
</SfGrid>

```

```

<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
TextAlign="TextAlign.Right" Width="130" Format="hh:mm tt"
DefaultValue="DateTime.Now" Type="ColumnType.DateTime">
<EditTemplate>
<SfTimePicker TValue="DateTime?" @bind-Value="@((context as
Order).OrderDate)"
AllowEdit="true" Format="hh:mm:tt" CssClass="CustomDateCSS"
ShowClearButton="true"></SfTimePicker>
</EditTemplate>
</GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" EditType="EditType.NumericEdit"
Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.ShipCountry) HeaderText="Ship Country"
EditType="EditType.DropDownEdit" Width="150">
</GridColumn>
</GridColumn>
</SfGrid>
@code{
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
OrderDate = DateTime.Now.AddDays(-x),
ShipCountry = (new string[] { "USA", "UK", "CHINA", "RUSSIA", "INDIA" })[new
Random().Next(5)]
}).ToList();
}
public class Order
{
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
public string ShipCountry { get; set; }
}
}

```

In the following image, **SfTimePicker** component is rendered with **EditTemplate** in OrderDate column

| + Add Edit Delete X Cancel Update | | | | | | |
|-----------------------------------|---------------|------------|--|---------|--------------|--|
| Order ID | Customer Name | Order Date | | Freight | Ship Country | |
| 1001 | ANTON | 08:37 PM | | \$2.10 | USA | |
| 1,002 | ANANTR | 08:37 PM | | 4.20 | CHINA | |
| 1003 | BLONP | 12:00 AM | | \$6.30 | RUSSIA | |
| 1004 | ALFKI | 12:30 AM | | \$8.40 | USA | |
| 1005 | BLONP | 01:00 AM | | \$10.50 | USA | |
| 1006 | BLONP | 01:30 AM | | \$12.60 | CHINA | |
| 1007 | ALFKI | 02:00 AM | | \$14.70 | CHINA | |
| 1008 | ANTON | 02:30 AM | | \$16.80 | CHINA | |
| 1009 | ANANTR | 03:00 AM | | \$18.90 | UK | |
| | | 03:30 AM | | | | |
| | | 04:00 AM | | | | |

« < 1 2 3 4 5 6 7 > » 1 of 7 pages (75 items)

Using MultiSelect Dropdown in EditTemplate

You can able to render SfMultiSelect component in EditTemplate. In the below sample we have rendered SfMultiSelect component in EditTemplate for ChosenItems column.

ASPX-CS

```
@using Syncfusion.Blazor.DropDowns
@using Syncfusion.Blazor.Grids
<SfGrid AllowPaging="true" DataSource="@Orders" Toolbar="@ (new
List<string>() { "Add", "Edit", "Delete", "Cancel", "Update" })">
<GridEditSettings AllowEditing="true" AllowDeleting="true"
AllowAdding="true" Mode="@EditMode.Normal"></GridEditSettings>
<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
IsPrimaryKey="true" TextAlign="@TextAlign.Center" Width="80"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.ChosenItems) HeaderText="Chosen Items"
Width="150">
<EditTemplate>
<SfMultiSelect ID="ChosenItems" @bind-Value="@((context as
Order).ChosenItems)" DataSource="@AvailableChoices" TValue="string[]"
TItem="MyChoiceItem">
<MultiSelectFieldSettings Value="ChosenItems"
Text="ChosenItems"></MultiSelectFieldSettings>
</SfMultiSelect>
</EditTemplate>
</Template>
@{
var d = (context as Order).ChosenItems;
<span>@String.Join(",", d)</span>
}
</Template>
</GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight"
EditType="EditType.NumericEdit" Format="C2" Width="90"
TextAlign="@TextAlign.Right"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText="Order Date"
EditType="EditType.DatePickerEdit" Format="d" Type="ColumnType.Date"
Width="100"></GridColumn>
</GridColumns>
```

```
</SfGrid>
@code{
public List<Order> Orders { get; set; }
public List<MyChoiceItem> AvailableChoices { get; set; }
protected override void OnInitialized()
{
    Orders = Enumerable.Range(1, 75).Select(x => new Order()
    {
        OrderID = 1000 + x,
        CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
    })[new Random().Next(5)],
        ChosenItems = new string[] { x + "ItemA" },
        Freight = 2.1 * x,
        OrderDate = DateTime.Now.AddDays(-x),
    }).ToList();
    AvailableChoices = Enumerable.Range(1, 75).Select(x => new MyChoiceItem()
    {
        Id = x,
        ChosenItems = x + "ItemA"
    }).ToList();
}
public class Order
{
    public int? OrderID { get; set; }
    public string CustomerID { get; set; }
    public string[] ChosenItems { get; set; }
    public DateTime? OrderDate { get; set; }
    public double? Freight { get; set; }
}
public class MyChoiceItem
{
    public int Id { get; set; }
    public string ChosenItems { get; set; }
}
}
```

In the following image, **SfMultiSelect** component is rendered with **EditTemplate** in ChosenItems column

| + Add Edit Delete Cancel Update | | | | | |
|---|---------------|-------------------|---------|-----------------------------------|-----------|
| Order ID | Customer Name | Chosen Items | Freight | Order Date | |
| 1001 | ALFKI | 1ItemA | \$2.10 | 6/24/2021 | |
| 1002 | ANTON | 2ItemA | \$4.20 | 6/23/2021 | |
| 1,003 | ANANTR | 3ItemA x 5ItemA x | 6.30 | <div><div></div><div></div></div> | 6/22/2021 |
| 1004 | ANANTR | 6ItemA | \$8.40 | 6/21/2021 | |
| 1005 | BOLID | 7ItemA | \$10.50 | 6/20/2021 | |
| 1006 | ANANTR | 8ItemA | \$12.60 | 6/19/2021 | |
| 1007 | ANTON | 9ItemA | \$14.70 | 6/18/2021 | |
| 1008 | BOLID | 10ItemA | \$16.80 | 6/17/2021 | |
| 1009 | ALFKI | 11ItemA | \$18.90 | 6/16/2021 | |
| 1010 | ANANTR | 12ItemA | \$21.00 | 6/15/2021 | |
| 1011 | ANTON | 13ItemA | \$23.10 | 6/14/2021 | |
| 1012 | ANTON | 14ItemA | \$25.20 | 6/13/2021 | |

<<

<

1

2

3

4

5

6

7

>

>>

1 of 7 pages (75 items)

Command column

The command column provides an option to add CRUD action buttons in a column. This can be defined by using the `GridCommandColumns` component which needs to be wrapped inside the `GridColumn` component.

The available built-in command buttons are:

- | Command Button | Actions |
- |-----|-----|
- | Edit | Edit the current row. |
- | Delete | Delete the current row. |
- | Save | Update the edited row. |
- | Cancel | Cancel the edited state. |

ASPX-CS


















```
@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders" AllowPaging="true" Height="315">
  <GridEditSettings AllowAdding="true" AllowEditing="true"
    AllowDeleting="true"></GridEditSettings>
  <GridColumns>
    <GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
      IsPrimaryKey="true" TextAlign="TextAlign.Right" Width="120"></GridColumn>
    <GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
      Width="120"></GridColumn>
    <GridColumn Field=@nameof(Order.OrderDate) HeaderText="Order Date"
      EditType="EditType.DatePickerEdit" Format="d" TextAlign="TextAlign.Right"
      Width="130"></GridColumn>
    <GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
      TextAlign="TextAlign.Right" Width="120"></GridColumn>
    <GridColumn HeaderText="Manage Records" Width="150">
      <GridCommandColumns>
```

```

<GridCommandColumn Type="CommandButtonType.Edit" ButtonOption="@ (new
CommandButtonOptions() { IconCss = "e-icons e-edit", CssClass = "e-flat"
}) "></GridCommandColumn>
<GridCommandColumn Type="CommandButtonType.Delete" ButtonOption="@ (new
CommandButtonOptions() { IconCss = "e-icons e-delete", CssClass = "e-flat"
}) "></GridCommandColumn>
<GridCommandColumn Type="CommandButtonType.Save" ButtonOption="@ (new
CommandButtonOptions() { IconCss = "e-icons e-update", CssClass = "e-flat"
}) "></GridCommandColumn>
<GridCommandColumn Type="CommandButtonType.Cancel" ButtonOption="@ (new
CommandButtonOptions() { IconCss = "e-icons e-cancel-icon", CssClass = "e-
flat" }) "></GridCommandColumn>
</GridCommandColumns>
</GridColumn>
</GridColumns>
</SfGrid>
@code{
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
OrderDate = DateTime.Now.AddDays(-x),
ShipCountry = (new string[] { "USA", "UK", "CHINA", "RUSSIA", "INDIA" })[new
Random().Next(5)]
}).ToList();
}
public class Order
{
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
public string ShipCountry { get; set; }
}
}

```

The following screenshot represents the command column.

| Order ID | Customer Name | Order Date | Freight | Manage Records |
|----------|---------------|---|---------|---|
| 1001 | BLONP | 7/21/2019 | \$2.10 |   |
| 1002 | ANANT | 7/20/2019  | 4.2 |   |
| 1003 | BLONP | 7/19/2019 | \$6.30 |   |
| 1004 | BLONP | 7/18/2019 | \$8.40 |   |
| 1005 | ANANTR | 7/17/2019 | \$10.50 |   |
| 1006 | ANTON | 7/16/2019 | \$12.60 |   |
| 1007 | BLONP | 7/15/2019 | \$14.70 |   |
| 1008 | BOLID | 7/14/2019 | \$16.80 |   |

K < 1 2 3 4 5 6 7 > X

1 of 7 pages (75 items)

Custom command

The custom command buttons can be added in a column by using the [Commands](#) property of the [GridColumn](#) component and the action for the custom buttons can be defined in the [CommandClicked](#) event.

The following sample code demonstrates adding custom command in the **Manage Records** column and the **CommandClicked** event which triggers when the command is clicked,

ASPX-CS

```
@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders" AllowPaging="true" Height="315">
  <GridEvents CommandClicked="OnCommandClicked" TValue="Order"></GridEvents>
  <GridEditSettings AllowAdding="true" AllowEditing="true"
    AllowDeleting="true"></GridEditSettings>
  <GridColumn>
    <GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
      IsPrimaryKey="true" TextAlign="TextAlign.Right" Width="120"></GridColumn>
    <GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
      Width="120"></GridColumn>
    <GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
      EditType="EditType.DatePickerEdit" Format="d" TextAlign="TextAlign.Right"
      Width="130"></GridColumn>
    <GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
      TextAlign="TextAlign.Right" Width="120"></GridColumn>
    <GridColumn HeaderText="Manage Records" Width="150">
      <GridCommandColumns>
        <GridCommandColumn ButtonOption="@ (new CommandButtonOptions() { Content =
          "Details", CssClass = "e-flat" })"></GridCommandColumn>
      </GridCommandColumns>
    </GridColumn>
  </GridColumn>
</SfGrid>
@code{
  public List<Order> Orders { get; set; }
```

```
protected override void OnInitialized()
{
    Orders = Enumerable.Range(1, 75).Select(x => new Order()
    {
        OrderID = 1000 + x,
        CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
    })[new Random().Next(5)],
        Freight = 2.1 * x,
        OrderDate = DateTime.Now.AddDays(-x),
        ShipCountry = (new string[] { "USA", "UK", "CHINA", "RUSSIA", "INDIA" })[new
        Random().Next(5)]
    }).ToList();
}

public class Order
{
    public int? OrderID { get; set; }
    public string CustomerID { get; set; }
    public DateTime? OrderDate { get; set; }
    public double? Freight { get; set; }
    public string ShipCountry { get; set; }
}

public void OnCommandClicked(CommandClickEventArgs<Order> args)
{
    // Perform required operations here
}
```

The following image represents the custom command added in the **Manage Records** column of the DataGrid component,

| Order ID | Customer Name | Order Date | Freight | Manage Records |
|----------|---------------|------------|---------|----------------|
| 1001 | BLONP | 8/28/2019 | \$2.10 | Details |
| 1002 | ALFKI | 8/27/2019 | \$4.20 | Details |
| 1003 | BLONP | 8/26/2019 | \$6.30 | Details |
| 1004 | ALFKI | 8/25/2019 | \$8.40 | Details |
| 1005 | BLONP | 8/24/2019 | \$10.50 | Details |
| 1006 | ANTON | 8/23/2019 | \$12.60 | Details |
| 1007 | ANANTR | 8/22/2019 | \$14.70 | Details |
| 1008 | BOLID | 8/21/2019 | \$16.80 | Details |

⏪
<
1
2
3
4
5
6
7
>
⏩

1 of 7 pages (75 items)

Column validation

Column validation allows you to validate the edited or added row data and it display errors for invalid fields before saving data. DataGrid uses **Form Validator** library for column validation. You can set validation rules by defining the [ValidationRules](#).

Validation in datagrid works based on the Microsoft Blazor EditForm behavior. So once the validation message is shown then it will be again validated only during the form submit or when you focus out from that particular field. Please refer the [Microsoft Validation](#) for further reference.

ASPX-CS

```
@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders" Height="315" AllowPaging="true" Toolbar="@ (new
List<string>() { "Add", "Edit", "Delete", "Update", "Cancel" })">
<GridEditSettings AllowAdding="true" AllowEditing="true"
AllowDeleting="true"></GridEditSettings>
<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
IsPrimaryKey="true" ValidationRules="@ (new ValidationRules{ Required= true
})" TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="120" ValidationRules="@ (new ValidationRules{ Required= true,
MinLength = 3 })"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
EditType="EditType.DatePickerEdit" Format="d" TextAlign="TextAlign.Right"
Width="130" Type="ColumnType.Date"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumns>
</SfGrid>
@code{
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
OrderDate = DateTime.Now.AddDays(-x),
}).ToList();
}
public class Order
{
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
}
```

The following screenshot represents the Column Validation in Normal Editing.

| + Add ✎ Edit 🗑 Delete 📄 Update ✕ Cancel | | | | |
|--|----------------------|----------------------|----------------------|--|
| Order ID | Customer Name | Order Date | Freight | |
| <input type="text"/> | <input type="text"/> | <input type="text"/> | <input type="text"/> | |
| 1001 | BOLID | 7/21/2019 | \$2.10 | |
| 1002 | BOLID | 7/20/2019 | \$4.20 | |
| 1003 | BOLID | 7/19/2019 | \$6.30 | |
| 1004 | ALFKI | 7/18/2019 | \$8.40 | |
| 1005 | ANANTR | 7/17/2019 | \$10.50 | |
| 1006 | BOLID | 7/16/2019 | \$12.60 | |
| 1007 | ALFKI | 7/15/2019 | \$14.70 | |
| 1008 | BLONP | 7/14/2019 | \$16.80 | |

K
<
1
2
3
4
5
6
7
>
X

1 of 7 pages (75 items)

Data Annotation

Data Annotation validation attributes are used to validate the fields in the DataGrid. The validation attributes that are supported in the DataGrid are listed below.

| Attribute Name | Functionality |

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

| Validations are,
 1. RequiredAttribute
 2. StringLengthAttribute
 3. RangeAttribute
 4. RegularExpressionAttribute
 5. MinLengthAttribute
 6. MaxLengthAttribute
 7. EmailAddressAttribute
 8. CompareAttribute
 9. DataTypeAttribute
 10. DataType.Custom
 11. DataType.Date
 12. DataType.DateTime
 13. DataType.EmailAddress
 14. DataType.ImageUrl
 15. DataType.Url | The data annotation validation attributes are used as **validation rules** in the DataGrid CRUD operations |

More information on the data annotation can be found in this [documentation](#) section.

Custom Validation

Custom Validation allows the users to customize the validations manually according to the user's criteria.

Custom Validation can be used by overriding the IsValid method inside the class inherits the Validation Attribute. All the validations are done inside the IsValid method.

The following sample code demonstrates custom validations implemented in the fields EmployeeID and Freight.

ASPX-CS

```
@using Syncfusion.Blazor.Grids;
@using System.ComponentModel.DataAnnotations;
@using System.Text.RegularExpressions;
<SfGrid DataSource="EmployeeList" AllowPaging="true" Toolbar="toolbar">
```



```

<GridEditSettings AllowAdding="true" AllowDeleting="true"
AllowEditing="true"></GridEditSettings>
<GridColumns>
<GridColumn Field="@nameof(EmployeeDetails.OrderID)" HeaderText="Order ID"
TextAlign="TextAlign.Right" IsPrimaryKey="true" > </GridColumn>
<GridColumn Field="@nameof(EmployeeDetails.CustomerName)"
HeaderText="Customer Name" TextAlign="TextAlign.Left"> </GridColumn>
<GridColumn Field="@nameof(EmployeeDetails.EmployeeID)" HeaderText="Employee
ID" TextAlign="TextAlign.Right"> </GridColumn>
<GridColumn Field="@nameof(EmployeeDetails.Freight)" HeaderText="Freight"
TextAlign="TextAlign.Right" Format="C2"> </GridColumn>
<GridColumn Field="@nameof(EmployeeDetails.ShipCity)" HeaderText="Ship City"
TextAlign="TextAlign.Left"> </GridColumn>
<GridColumn Field="@nameof(EmployeeDetails.ShipName)" HeaderText="Ship Name"
TextAlign="TextAlign.Left"> </GridColumn>
</GridColumns>
</SfGrid>
@code
{
List<EmployeeDetails> EmployeeList;
string[] toolbar = new string[] { "Add", "Edit", "Delete", "Update",
"Cancel" };
protected override void OnInitialized()
{
base.OnInitialized();
EmployeeList = Enumerable.Range(1, 20).Select(x => new EmployeeDetails()
{
OrderID = 10240 + x,
CustomerName = new string[] { "VINET", "TOSMP", "HANAR", "VICTE" }[new
Random().Next(4)],
EmployeeID = x,
Freight = new float[] { 32.28f, 22.90f, 30.99f, 50.52f }[new
Random().Next(4)],
ShipCity = new string[] { "Reims", "Munster", "Rio de Janeir", "Lyon" }[new
Random().Next(4)],
ShipName = new string[] { "Vins et alocools chevalie", "Toms Spezialitaten",
"Hanari Carnes", "Supremes delices" }[new Random().Next(4)]
}).ToList();
}
public class EmployeeDetails
{
[Required]
public int? OrderID { get; set; }
public string CustomerName { get; set; }
[CustomValidationEmployeeID]
public int EmployeeID { get; set; }
[CustomValidationFreight]
public float Freight { get; set; }
public string ShipCity { get; set; }
public string ShipName { get; set; }
}
public class CustomValidationEmployeeID : ValidationAttribute
{
protected override ValidationResult IsValid(object value, ValidationContext
validationContext)
{
if (value != null)

```

```
{
int employeeID = Convert.ToInt16(value);
if (employeeID >= 1)
{
return ValidationResult.Success;
}
else
{
return new ValidationResult("Employee ID value should be greater than
zero");
}
}
else
{
return new ValidationResult("Employee ID value is required");
}
}
}
public class CustomValidationFreight : ValidationAttribute
{
protected override ValidationResult IsValid(object value, ValidationContext
validationContext)
{
if (value != null)
{
float freight = (float)value;
if (freight >= 1 && freight <= 10000)
{
return ValidationResult.Success;
}
else
{
return new ValidationResult("Freight value should between 1 and 10,000");
}
}
else
{
return new ValidationResult("Freight value is required");
}
}
}
}
```

Custom validator component

Apart from using default validation and custom validation, there are cases where you might want to use your validator component to validate the grid edit form. Such cases can be achieved using the **Validator** property of the **GridEditSettings** component which accepts a validation component and inject it inside the **EditForm** of the grid. Inside the **Validator**, you can access the data using the implicit named parameter context which is of type [ValidatorTemplateContext](#).

For creating a form validator component you can refer [here](#).

In the below code example, the following things have been done.

- Created a form validator component named **MyCustomValidator** which accepts [ValidatorTemplateContext](#) value as parameter.
- Used the **MyCustomValidator** component inside the **Validator** property.
- This validator component will check whether Freight value is in between 0 to 100.
- Displayed the validation error messages using **ValidationMessage** component.

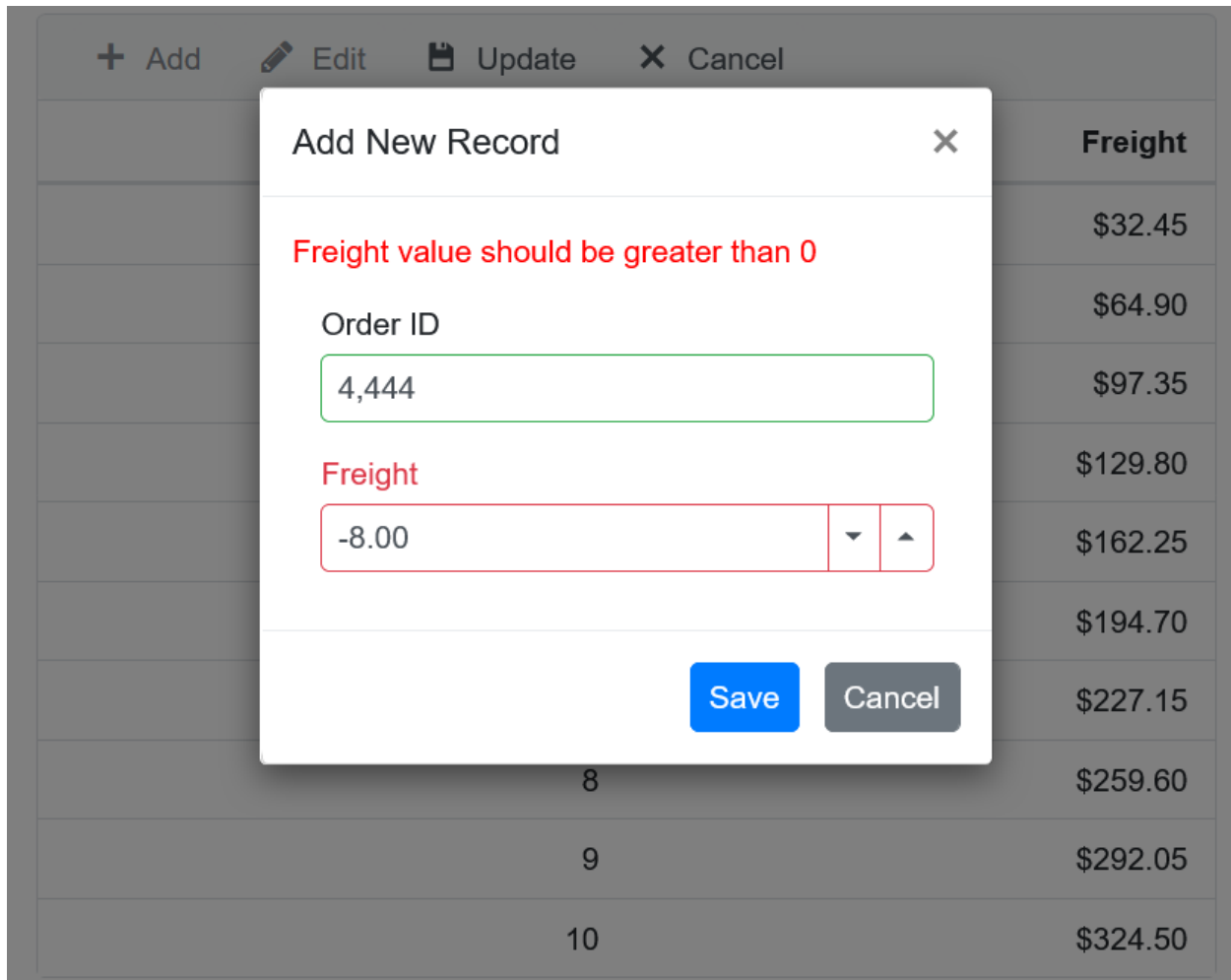
CSHARP

```
[MyCustomValidator.cs]
using Microsoft.AspNetCore.Components;
using Microsoft.AspNetCore.Components.Forms;
using Syncfusion.Blazor.Grids;
public class MyCustomValidator : ComponentBase
{
    [Parameter]
    public ValidatorTemplateContext context { get; set; }
    private ValidationMessageStore messageStore;
    [CascadingParameter]
    private EditContext CurrentEditContext { get; set; }
    protected override void OnInitialized()
    {
        messageStore = new ValidationMessageStore(CurrentEditContext);
        CurrentEditContext.OnValidationRequested += ValidateRequested;
        CurrentEditContext.OnFieldChanged += ValidateField;
    }
    protected void HandleValidation(FieldIdentifier identifier)
    {
        if (identifier.FieldName.Equals("Freight"))
        {
            messageStore.Clear(identifier);
            if ((context.Data as OrdersDetails).Freight < 0)
            {
                messageStore.Add(identifier, "Freight value should be greater than 0");
            }
            else if ((context.Data as OrdersDetails).Freight > 100)
            {
                messageStore.Add(identifier, "Freight value should be lesser than 100");
            }
            else
            {
                messageStore.Clear(identifier);
            }
        }
    }
    protected void ValidateField(object editContext, FieldChangedEventArgs fieldChangedEventArgs)
    {
        HandleValidation(fieldChangedEventArgs.FieldIdentifier);
    }
    private void ValidateRequested(object editContext, ValidationRequestedEventArgs validationEventArgs)
    {
        HandleValidation(CurrentEditContext.Field("Freight"));
    }
}
```

CSHARP

```
[Index.razor]
<SfGrid TValue="OrdersDetails" DataSource="GridData"
Toolbar="@ (new List<string>() { "Add", "Edit", "Update", "Cancel" })">
<GridEditSettings AllowAdding="true" AllowEditing="true"
Mode="EditMode.Dialog">
<Validator>
@{
ValidatorTemplateContext txt = context as ValidatorTemplateContext;
}
<MyCustomValidator context="@txt"></MyCustomValidator>
<ValidationMessage For="@(() => (txt.Data as
OrdersDetails).Freight)"></ValidationMessage>
</Validator>
</GridEditSettings>
<GridColumns>
<GridColumn Field=@nameof(OrdersDetails.OrderID) HeaderText="Order ID"
TextAlign="TextAlign.Right" Width="120" IsPrimaryKey="true"></GridColumn>
<GridColumn Field=@nameof(OrdersDetails.Freight) HeaderText="Freight"
Format="C2" TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumns>
</SfGrid>
@code{
private List<OrdersDetails> GridData;
protected override void OnInitialized()
{
Random r = new Random();
GridData = Enumerable.Range(1, 10).Select(x => new OrdersDetails()
{
OrderID = x,
Freight = 32.45 * x
}).ToList();
}
}
```

The output will be as follows.



Display validation message using in-built tooltip

In the above code example, you can see that **ValidationMessage** component is used, this might be not suitable when using Inline editing or batch editing. In such cases, you can use the in-built validation tooltip to show those error messages by using

`ValidatorTemplateContext.ShowValidationMessage(fieldName, IsValid, Message)` method.

Now the `HandleValidation` method of the `MyCustomValidator` component would be changed like below.

CSHARP

```
protected void HandleValidation(FieldIdentifier identifier)
{
    if (identifier.FieldName.Equals("Freight"))
    {
        messageStore.Clear(identifier);
        if ((context.Data as OrdersDetails).Freight < 0)
        {
            messageStore.Add(identifier, "Freight value should be greater than 0");
            context.ShowValidationMessage("Freight", false, "Freight value should be greater than 0");
        }
        else if ((context.Data as OrdersDetails).Freight > 100)
        {

```

```
messageStore.Add(identifier, "Freight value should be lesser than 100");
context.ShowValidationMessage("Freight", false, "Freight value should be
lesser than 100");
}
else
{
messageStore.Clear(identifier);
context.ShowValidationMessage("Freight", true, null);
}
}
}
```

The output will be as follows.

| Order ID | Freight |
|----------|----------|
| 1 | \$32.45 |
| 2 | \$64.90 |
| 3 | \$97.35 |
| 4 | \$129.80 |
| 5 | \$162.25 |
| 6 | \$194.70 |
| 7 | \$227.15 |
| 8 | \$259.60 |
| 9 | \$292.05 |
| 10 | \$324.50 |

[Disable in-built validator component](#)

Validator property can also be used to disable the in-built validator component used by the grid. For instance, by default, the grid uses two validator components, **DataAnnotationValidator** and an internal [ValidationRules](#) property handling validator, for handling edit form validation. If you are willing to use only the **DataAnnotationValidator** component, then it could be simply achieved by using the below code.

ASPX-CS

```

<SfGrid TValue="OrdersDetails" DataSource="GridData"
Toolbar="@ (new List<string>() { "Add", "Edit", "Update", "Cancel" })">
<GridEditSettings AllowAdding="true" AllowEditing="true"
Mode="EditMode.Dialog">
<Validator>
<DataAnnotationsValidator></DataAnnotationsValidator>
</Validator>
</GridEditSettings>
<GridColumns>
<GridColumn Field=@nameof(OrdersDetails.OrderID) HeaderText="Order ID"
TextAlign="TextAlign.Right" Width="120" IsPrimaryKey="true"></GridColumn>
<GridColumn Field=@nameof(OrdersDetails.Freight) HeaderText="Freight"
Format="C2" TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumns>
</SfGrid>
@code{
private List<OrdersDetails> GridData;
protected override void OnInitialized()
{
Random r = new Random();
GridData = Enumerable.Range(1, 10).Select(x => new OrdersDetails()
{
OrderID = x,
Freight = 32.45 * x
}).ToList();
}
}

```

Provide new item or edited item using events

Grid uses `Activator.CreateInstance<TValue>()` to create or clone new record instance during add and edit operations, so it is must to have parameterless constructor defined for the model class.

There are cases where custom logic is required for creating new object or new object instance cannot be created using `Activator.CreateInstance<TValue>()`. In such cases you can provide model object instance manually using events.

Normal or Dialog editing

You can use `OnActionBegin` event to provide new object instance during editing operation. The new object should be assigned to the `OnActionBegin<TValue>.Data` property.

In the following example:

- A model class with no parameter-less constructor is bound with the grid.
- Enabled inline editing feature in grid.
- `OnActionBegin` event callback is assigned in which `Data` property is assigned with custom object for both add and edit operation.

CSHARP

```

<SfGrid DataSource="@Orders" Toolbar="@ (new List<string>() { "Add", "Edit",
"Delete", "Update", "Cancel" })">
<GridEditSettings AllowEditing="true" AllowAdding="true"
Mode="EditMode.Normal"></GridEditSettings>
<GridColumns>

```

```

<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
IsPrimaryKey="true" TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumns>
<GridEvents TValue="Order" OnActionBegin="ActionBegin"></GridEvents>
</SfGrid>
@code {
List<Order> Orders { get; set; }
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 10).Select(x => new Order(1000 + x)
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI",
"ANANTR", "ANTON", "BLONP", "BOLID" })[new Random().Next(5)],
Freight = (new double[] { 2, 1, 4, 5, 3 })[new Random().Next(5)] * x,
OrderDate = (new DateTime[] { new DateTime(2019, 01, 01), new DateTime(2019,
01, 02) })[new Random().Next(2)]
}).ToList();
}
public void ActionBegin(ActionEventArgs<Order> arg)
{
//Handles add operation
if (arg.RequestType.Equals(Syncfusion.Blazor.Grids.Action.Add))
{
arg.Data = new Order(0) { CustomerID = "Customer ID" };
}
//Handles edit operation. During edit operation, original object will be
cloned.
if (arg.RequestType.Equals(Syncfusion.Blazor.Grids.Action.BeginEdit))
{
arg.Data = new Order(arg.RowData.OrderID)
{
CustomerID = arg.RowData.CustomerID,
Freight = arg.RowData.Freight,
OrderDate = arg.RowData.OrderDate
};
}
}
// This class does not contain any parameter-less constructor, hence this
cannot be instantiated using Activator.CreateInstance.
public class Order
{
public Order(int? orderid) => OrderID = orderid;
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
}

```


Batch editing

You can use `OnBatchAdd` and `OnCellEdit` event to provide new object instance during add and cell edit operation respectively.

For add operation assign new object to the `OnBatchAdd.DefaultData` property. For cell edit, assign cloned object in the `OnCellEdit.Data` property.

In the following example:

- A model class with no parameter-less constructor is bound with grid.
- Enabled batch editing feature in grid.
- `OnBatchAdd` event callback is assigned in which `DefaultData` property is assigned with custom object for add operation.
- `OnCellEdit` event callback is assigned in which the `Data` property is assigned with a custom object for handling edit operation.

CSHARP

```
<SfGrid DataSource="@Orders" Toolbar="@ (new List<string>() { "Add",
"Update", "Cancel" }) ">
<GridEditSettings AllowEditing="true" AllowAdding="true"
Mode="EditMode.Batch"></GridEditSettings>
<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
IsPrimaryKey="true" TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumns>
<GridEvents TValue="Order" OnBatchAdd="BeforeAdd"
OnCellEdit="CellEdit"></GridEvents>
</SfGrid>
@code {
List<Order> Orders { get; set; }
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 10).Select(x => new Order(1000 + x)
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI",
"ANANTR", "ANTON", "BLONP", "BOLID" })[new Random().Next(5)],
Freight = (new double[] { 2, 1, 4, 5, 3 })[new Random().Next(5)] * x,
OrderDate = (new DateTime[] { new DateTime(2019, 01, 01), new DateTime(2019,
01, 02) })[new Random().Next(2)]
}).ToList();
}
public void BeforeAdd(BeforeBatchAddArgs<Order> arg)
{
arg.DefaultData = new Order(0) { CustomerID = "Customer ID" };
}
```

```

}
public void CellEdit(CellEditArgs<Order> arg)
{
    //Return args.Data if its not null so previously edited data will not be
    lost.
    arg.Data = arg.Data ?? new Order(arg.RowData.OrderID)
    {
        CustomerID = arg.RowData.CustomerID,
        Freight = arg.RowData.Freight,
        OrderDate = arg.RowData.OrderDate
    };
}
// This class does not contain any parameter-less constructor, hence this
cannot be instantiated using Activator.CreateInstance.
public class Order
{
    public Order(int orderid) => OrderID = orderid;
    public int? OrderID { get; set; }
    public string CustomerID { get; set; }
    public DateTime? OrderDate { get; set; }
    public double? Freight { get; set; }
}
}

```

Entity Framework

This section uses and follows the code explained in the [Entity Framework data binding](#) section hence we recommend you to refer Entity framework data binding section before continuing this section.

Handle CRUD in data access layer class

Now add methods **AddOrder**, **UpdateOrder**, **DeleteOrder** in the “**OrderDataAccessLayer.cs**” to handle the insert, update and remove operations respectively. **CRUD** record details are bound to the **Order** parameter. Please refer the following code.

CSHARP

```

using Microsoft.EntityFrameworkCore;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using EFGrid.Shared.Models;
namespace EFGrid.Shared.DataAccess
{
    public class OrderDataAccessLayer
    {
        OrderContext db = new OrderContext();
        public DbSet<Order> GetAllOrders()
        {
            try
            {
                return db.Orders;
            }
            catch
            {
                throw;
            }
        }
    }
}

```

```
}  
public void AddOrder(Order Order)  
{  
    try  
    {  
        db.Orders.Add(Order);  
        db.SaveChanges();  
    }  
    catch  
    {  
        throw;  
    }  
}  
public void UpdateOrder(Order Order)  
{  
    try  
    {  
        db.Entry(Order).State = EntityState.Modified;  
        db.SaveChanges();  
    }  
    catch  
    {  
        throw;  
    }  
}  
public void DeleteOrder(int id)  
{  
    try  
    {  
        Order ord = db.Orders.Find(id);  
        db.Orders.Remove(ord);  
        db.SaveChanges();  
    }  
    catch  
    {  
        throw;  
    }  
}  
}
```

Enable CRUD in Web API

Now you have to create a new **Post**, **Put**, **Delete** method in the Web API controller which will perform the CRUD operations and returns the appropriate resultant data. The '**SfDataManager**' will make requests to this action based on route name.

C#

```
using System;  
using System.Collections;  
using System.Collections.Generic;  
using System.Linq;  
using System.Threading.Tasks;  
using Microsoft.AspNetCore.Http;  
using Microsoft.AspNetCore.Mvc;  
using Microsoft.Extensions.Primitives;
```

```
using EFGGrid.Shared.DataAccess;
using EFGGrid.Shared.Models;
namespace WebApplication1.Server.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class DefaultController : ControllerBase
    {
        OrderDataAccessLayer db = new OrderDataAccessLayer();
        [HttpGet]
        public object Get()
        {
            IQueryable<Order> data = db.GetAllOrders().AsQueryable();
            var count = data.Count();
            var queryString = Request.Query;
            if (queryString.Keys.Contains("$inlinecount"))
            {
                StringValues Skip;
                StringValues Take;
                int skip = (queryString.TryGetValue("$skip", out Skip)) ?
                    Convert.ToInt32(Skip[0]) : 0;
                int top = (queryString.TryGetValue("$top", out Take)) ?
                    Convert.ToInt32(Take[0]) : data.Count();
                return new { Items = data.Skip(skip).Take(top), Count = count };
            }
            else
            {
                return data;
            }
        }
        [HttpGet("{id}", Name = "Get")]
        public string Get(int id)
        {
            return "value";
        }
        [HttpPost]
        public void Post([FromBody]Order Order)
        {
            Random rand = new Random();
            db.AddOrder(Order);
        }
        [HttpPut]
        public object Put([FromBody]Order Order)
        {
            db.UpdateOrder(Order);
            return Order;
        }
        [HttpDelete("{id}")]
        public void Delete(int id)
        {
            db.DeleteOrder(id);
        }
    }
}
```

Configure the datagrid to perform CRUD operations

ASPX-CS

```
@using Syncfusion.Blazor
@using Syncfusion.Blazor.Grids
<SfGrid TValue="Order" AllowPaging="true" Toolbar="@ (new List<string>() {
    "Add", "Edit", "Delete", "Cancel", "Update" }) ">
<SfDataManager Url="api/Default"
    Adaptor="Adaptors.WebApiAdaptor"></SfDataManager>
<GridEditSettings AllowAdding="true" AllowDeleting="true"
    AllowEditing="true" Mode="EditMode.Normal"></GridEditSettings>
<GridColumns>
<GridColumn Field="OrderID" HeaderText="Order ID" IsPrimaryKey="true"
    IsIdentity="true" TextAlign="@Syncfusion.Blazor.Grids.TextAlign.Right"
    Width="90"></GridColumn>
<GridColumn Field="CustomerID" HeaderText="Customer ID"
    Width="90"></GridColumn>
<GridColumn Field="EmployeeID" HeaderText="Employee ID"
    Width="90"></GridColumn>
</GridColumns>
</SfGrid>
@code{
public class Order {
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
}
```

You can find the fully working sample [here](#).

Perform CRUD operation using Grid Events

IQueryable data can be bound directly to Grid component from database without using Data Adaptors. IQueryable data bound to Grid component using DataSource property of SfGrid. While binding the Data to Grid component using **DataSource** property, CRUD actions needs to be handled using Grid Action Events (i.e.) using **OnActionBegin** and **OnActionComplete** events of Grid.

Create an interface layer to the Database

Create an interface with CRUD methods like below based on your model class.

CSHARP

```
using System.Collections.Generic;
using System.Linq;
namespace LibraryManagement.Models
{
interface ILibraryService
{
IQueryable<Book> GetBooks();
void InsertBook(Book employee);
void UpdateBook(long id, Book employee);
Book SingleBook(long id);
void DeleteBook(long id);
}
}
```

Create an intermediate service using the interface

By inheriting the interface, create a new service to retrieve the data from database and perform CRUD operation. Refer the below demonstration.

CSHARP

```
using Microsoft.EntityFrameworkCore;
using System.Linq;
namespace LibraryManagement.Models
{
    public class LibraryService : ILibraryService
    {
        private LibraryContext _context;
        public LibraryService(LibraryContext context)
        {
            _context = context;
        }
        public void DeleteBook(long id)
        {
            try
            {
                Book ord = _context.Books.Find(id);
                _context.Books.Remove(ord);
                _context.SaveChanges();
            }
            catch
            {
                throw;
            }
        }
        public IQueryable<Book> GetBooks()
        {
            try
            {
                return _context.Books.AsQueryable();
            }
            catch
            {
                throw;
            }
        }
        public void InsertBook(Book book)
        {
            try
            {
                _context.Books.Add(book);
                _context.SaveChanges();
            }
            catch
            {
                throw;
            }
        }
        public void UpdateBook(long id, Book book)
        {

```

```

try
{
    var local = _context.Set<Book>().Local.FirstOrDefault(entry =>
        entry.Id.Equals(book.Id));
    // check if local is not null
    if (local != null)
    {
        // detach
        _context.Entry(local).State = EntityState.Detached;
    }
    _context.Entry(book).State = EntityState.Modified;
    _context.SaveChanges();
}
catch
{
    throw;
}
}
}
}

```

Configure the DataGrid component to perform CRUD actions using Grid events

Since data is bound to Grid using DataSource property, perform CRUD actions will be reflected at Grid component level only. To reflect the changes in database, we need to handle the changes in Grid action events.

OnActionBegin – This event will be triggered when the action gets initiated. So, while inserting/updating a record, **RequestType Save** will be sent in the event arguments to save the changes in the database. Similarly, while deleting a record, RequestType as Delete will be initiated to perform actions externally. Since for both Update and Insert action, RequestType will be Save, we can differentiate them by using the **Args.Action** property, which will indicate the current action.

OnActionComplete – It will be triggered when certain actions are completed. Here, we can refresh the Grid component with an updated datasource to reflect the changes.

ASPX-CS

```

@using LibraryManagement.Models
@inject ILibraryService LibraryService
<SfGrid DataSource="@LibraryBooks" Toolbar="@new List<string>() { "Add",
    "Edit", "Delete", "Cancel", "Update" })" TValue="Book">
    <GridEditSettings AllowAdding="true" AllowDeleting="true"
        AllowEditing="true" Mode="EditMode.Normal"></GridEditSettings>
    <GridEvents OnActionBegin="ActionBeginHandler"
        OnActionComplete="ActionCompleteHandler" TValue="Book"></GridEvents>
    <GridColumns>
        <GridColumn Field="@nameof(Book.Id)" IsPrimaryKey="true" IsIdentity="true"
            Visible="false"></GridColumn>
        <GridColumn Field="@nameof(Book.Name)" Width="150"></GridColumn>
        <GridColumn Field="@nameof(Book.Author)" Width="150"></GridColumn>
        <GridColumn Field="@nameof(Book.Quantity)" Width="90"
            TextAlign="TextAlign.Right"></GridColumn>
        <GridColumn Field="@nameof(Book.Price)" Width="90" Format="C2"
            TextAlign="TextAlign.Right"></GridColumn>
    </GridColumns>
</SfGrid>

```

```

<GridColumn Field="@nameof(Book.Available)" DisplayAsCheckBox="true"
Width="70"></GridColumn>
</GridColumn>
</SfGrid>
@code
{
public IQueryable<Book> LibraryBooks { get; set; }
protected override void OnInitialized()
{
LibraryBooks = LibraryService.GetBooks();
}
public void ActionBeginHandler(ActionEventArgs<Book> Args)
{
if (Args.RequestType.Equals(Syncfusion.Blazor.Grids.Action.Save))
{
if (Args.Action == "Add")
{
LibraryService.InsertBook(Args.Data);
}
else
{
LibraryService.UpdateBook(Args.Data.Id, Args.Data);
}
}
if (Args.RequestType.Equals(Syncfusion.Blazor.Grids.Action.Delete))
{
LibraryService.DeleteBook(Args.Data.Id);
}
}
public void ActionCompleteHandler(ActionEventArgs<Book> Args)
{
if (Args.RequestType.Equals(Syncfusion.Blazor.Grids.Action.Save))
{
LibraryBooks = LibraryService.GetBooks(); //to fetch the updated data from
db to Grid
}
}
}

```

Please find the sample from this [Github](#) location.

Performing CRUD operations programmatically

You can perform CRUD operations like **Add** , **Update** , **Delete** by using the [AddRecord](#) , [UpdateRow](#) , [DeleteRow](#) methods.

- **AddRecord** - Add a new record into the datagrid
- **UpdateRow** - Update a existing record in a datagrid.
- **DeleteRow** - Delete a selected row in the datagrid

ASPX-CS

```

@using Syncfusion.Blazor.Grids
@using Syncfusion.Blazor.Buttons
<SfButton @onclick="Add"> Add </SfButton>
<SfButton @onclick="Update"> Update - 1001 </SfButton>

```



```

<SfButton @onclick="Delete"> Delete the selected row </SfButton>
<SfGrid DataSource="@Orders" AllowPaging="true" @ref="Grid" Height="315">
<GridEditSettings AllowAdding="true" AllowEditing="true"
AllowDeleting="true"></GridEditSettings>
<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
IsPrimaryKey="true" ValidationRules="@ (new ValidationRules{ Required = true
})" TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
ValidationRules="@ (new ValidationRules{ Required = true })"
Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" EditType="EditType.NumericEdit"
Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.ShipCountry) HeaderText="Ship Country"
EditType="EditType.DropDownEdit" Width="150"></GridColumn>
</GridColumns>
</SfGrid>
@code{
public List<Order> Orders { get; set; }
SfGrid<Order> Grid;
public async Task Add()
{
Order adddata = new Order()
{
OrderID = 1000,
CustomerID = "MJDGX",
ShipCountry = "LONDON",
Freight = 3.01
};
await this.Grid.AddRecord(adddata);
}
public async Task Update()
{
Order data = new Order() { OrderID = 1001, CustomerID = "ABCDE", ShipCountry
= "LONDON", Freight = 2.91 };
await this.Grid.UpdateRow(1, data);
}
public async Task Delete()
{
await this.Grid.DeleteRecord();
}
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
OrderDate = DateTime.Now.AddDays(-x),
ShipCountry = (new string[] { "USA", "UK", "CHINA", "RUSSIA", "INDIA" })[new
Random().Next(5)]
}).ToList();
}
}
public class Order
{

```

```

public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
public string ShipCountry { get; set; }
}

```

The following GIF represents the datagrid with Add, Update, Delete items,

| Add | Update - 1001 | Delete the selected row |
|----------|---------------|-------------------------|
| Order ID | Customer Name | Freight Ship Country |
| 1001 | BLONP | \$2.10 UK |
| 1002 | ANANTR | \$4.20 CHINA |
| 1003 | ALFKI | \$6.30 USA |
| 1004 | ANANTR | \$8.40 UK |
| 1005 | ANANTR | \$10.50 CHINA |
| 1006 | ANTON | \$12.60 INDIA |
| 1007 | ANANTR | \$14.70 RUSSIA |
| 1008 | BLONP | \$16.80 RUSSIA |
| 1009 | ANTON | \$18.90 USA |

K < 1 2 3 4 5 6 7 > X
 1 of 7 pages (75 items)

Custom external form editing

You can perform the edit operation of Datagrid in a Custom external form. The edit operation can be done by [RowSelected](#) property.

CSHARP

```

@using Syncfusion.Blazor.Grids
@using Syncfusion.Blazor.Buttons
@using Syncfusion.Blazor.DropDowns
@using Syncfusion.Blazor.Inputs
<div class="row">
<div class="col-md-6">
<div>
<div class="form-row">
<div class="form-group col-md-12">
<label for="orderedit">OrderID</label>
<input class="form-control" @bind="@ (SelectedProduct.OrderID)" type="number"
disabled />
</div>
</div>
<div class="form-row">
<div class="form-group col-md-12">
<label for="customeredit">CustomerID</label>

```

```

<SfTextBox ID="CustomerID" @bind-
Value="@ (SelectedProduct.CustomerID) "></SfTextBox>
</div>
</div>
<div class="form-row">
<div class="form-group col-md-12">
<label for="freightedit">Freight</label>
<SfNumericTextBox ID="Freight" TValue="double?" @bind-
Value="@SelectedProduct.Freight"></SfNumericTextBox>
</div>
</div>
<div class="form-row">
<div class="form-group col-md-12">
<label for="countryedit">ShipCountry</label>
<SfDropDownList ID="ShipCountry" @bind-Value="@SelectedProduct.ShipCountry"
TItem="Country" TValue="string" DataSource="@Dropdown">
<DropDownListFieldSettings Value="ShipCountry"
Text="ShipCountry"></DropDownListFieldSettings>
</SfDropDownList>
</div>
</div>
</div>
<SfButton @onclick="Save">Save</SfButton>
</div>
<div class="col-md-6">
<SfGrid DataSource="@Orders" AllowPaging="true" @ref="Grid" Height=315>
<GridEditSettings AllowEditing="true"></GridEditSettings>
<GridEvents RowSelected="RowSelectHandler" TValue="Order"></GridEvents>
<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="OrderID"
IsPrimaryKey="true" TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="CustomerID"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight"
TextAlign="TextAlign.Right" Format="C2" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.ShipCountry) HeaderText="ShipCountry"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumns>
</SfGrid>
</div>
</div>
@code{
public List<Order> Orders { get; set; }
SfGrid<Order> Grid;
public Order SelectedProduct = new Order();
public class Country
{
public string ShipCountry { get; set; }
}
List<Country> Dropdown = new List<Country>
{
new Country() { ShipCountry= "USA" },
new Country() { ShipCountry= "UK" },
new Country() { ShipCountry= "RUSSIA" },
new Country() { ShipCountry= "INDIA" },
new Country() { ShipCountry= "CHINA" },
};

```

```

async Task Save()
{
    await this.Grid.UpdateRow(1, SelectedProduct);
}
protected override void OnInitialized()
{
    Orders = Enumerable.Range(1, 75).Select(x => new Order()
    {
        OrderID = 1000 + x,
        CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
    }) [new Random().Next(5)],
        Freight = 2.1 * x,
        ShipCountry = (new string[] { "USA", "UK", "CHINA", "RUSSIA", "INDIA" }) [new
        Random().Next(5)]
    }).ToList();
}
public class Order
{
    public int? OrderID { get; set; }
    public string CustomerID { get; set; }
    public double? Freight { get; set; }
    public string ShipCountry { get; set; }
}
public void RowSelectHandler(RowSelectEventArgs<Order> args)
{
    SelectedProduct = args.Data;
}
}

```

The following GIF represent the datagrid with Custom External form editing,

| OrderID | CustomerID | Freight | ShipCountry |
|---------|------------|---------|-------------|
| 1001 | BOLID | \$2.10 | CHINA |
| 1002 | BOLID | \$4.20 | UK |
| 1003 | ALFKI | \$6.30 | USA |
| 1004 | BOLID | \$8.40 | RUSSIA |
| 1005 | ALFKI | \$10.50 | USA |
| 1006 | BLONP | \$12.60 | UK |
| 1007 | ANTON | \$14.70 | CHINA |
| 1008 | ALFKI | \$16.80 | INDIA |
| 1009 | ALFKI | \$18.90 | USA |

1 of 7 pages (75 items)

Dialog template

Before adding dialog template to the datagrid, we strongly recommend you to go through the [Template](#) section topic to configure the template.

To know about customizing the **Dialog Template** in Blazor DataGrid Component, you can check this video.

{% youtube

"youtube:https://www.youtube.com/watch?v=Cfj476xT2ao"%}

The dialog template editing provides an option to customize the default behavior of dialog editing. Using the dialog template, you can render your editors by defining the [GridEditSettings](#) component's [Mode](#) property as **Dialog** and wrapping the HTML elements inside the [Template](#) property of [GridEditSettings](#).

Custom components inside the Dialog Template must be specified with two-way (**@bind-Value**) binding to reflect the changes in DataGrid.

In some cases, you would need to add new field editors in the dialog which are not present in the column model. In that case, the dialog template will help you to customize the default edit dialog.

The following sample code demonstrates DataGrid enabled with dialog template editing,

ASPX-CS

```
@using Syncfusion.Blazor.Grids
@using Syncfusion.Blazor.Calendars
@using Syncfusion.Blazor.DropDowns
@using Syncfusion.Blazor.Inputs
<SfGrid DataSource="@GridData" Toolbar="@ (new string[] { "Add", "Edit",
    "Delete", "Update", "Cancel" }) ">
<GridEditSettings AllowAdding="true" AllowEditing="true"
    AllowDeleting="true" Mode="@EditMode.Dialog">
<Template>
@{
    var Order = (context as OrdersDetails);
<div>
<div class="form-row">
<div class="form-group col-md-6">
<label class="e-float-text e-label-top">Order ID</label>
<SfNumericTextBox ID="OrderID" @bind-Value="@ (Order.OrderID) "
    Enabled="@ ((Order.OrderID == null) ? true : false) "></SfNumericTextBox>
</div>
<div class="form-group col-md-6">
<label class="e-float-text e-label-top">Customer Name</label>
<SfAutoComplete ID="customerID" TItem="OrdersDetails" @bind-
    Value="@ (Order.CustomerID) " TValue="string" DataSource="@GridData">
<AutoCompleteFieldSettings Value="CustomerID"></AutoCompleteFieldSettings>
</SfAutoComplete>
</div>
</div>
<div class="form-row">
<div class="form-group col-md-6">
<label class="e-float-text e-label-top">Freight</label>
<SfNumericTextBox ID="Freight" @bind-Value="@ (Order.Freight) "
    TValue="double?"></SfNumericTextBox>
</div>
<div class="form-group col-md-6">
<label class="e-float-text e-label-top">Order Date</label>
<SfDatePicker ID="OrderDate" @bind-
    Value="@ (Order.OrderDate) "></SfDatePicker>
</div>
</div>
```

```

</div>
<div class="form-row">
<div class="form-group col-md-6">
<label class="e-float-text e-label-top">Ship Country</label>
<SfDropDownList ID="ShipCountry" @bind-Value="@ (Order.ShipCountry) "
TItem="OrdersDetails" TValue="string" DataSource="@GridData">
<DropDownListFieldSettings Value="ShipCountry"
Text="ShipCountry"></DropDownListFieldSettings>
</SfDropDownList>
</div>
<div class="form-group col-md-6">
<label class="e-float-text e-label-top">Ship City</label>
<SfDropDownList ID="ShipCity" @bind-Value="@ (Order.ShipCity) "
TItem="OrdersDetails" TValue="string" DataSource="@GridData">
<DropDownListFieldSettings Value="ShipCity"
Text="ShipCity"></DropDownListFieldSettings>
</SfDropDownList>
</div>
</div>
<div class="form-row">
<div class="form-group col-md-12">
<label class="e-float-text e-label-top">Ship Address</label>
<SfTextBox ID="ShipAddress" @bind-Value="@ (Order.ShipAddress) "></SfTextBox>
</div>
</div>
</div>
}
</Template>
</GridEditSettings>
<GridColumns>
<GridColumn Field=@nameof(OrdersDetails.OrderID) HeaderText="Order ID"
IsPrimaryKey="true" TextAlign="@TextAlign.Center"
HeaderTextAlign="@TextAlign.Center" Width="140"></GridColumn>
<GridColumn Field=@nameof(OrdersDetails.CustomerID) HeaderText="Customer
Name" Width="150"></GridColumn>
<GridColumn Field=@nameof(OrdersDetails.Freight) HeaderText="Freight"
Format="C2" Width="140" TextAlign="@TextAlign.Right"
HeaderTextAlign="@TextAlign.Right"></GridColumn>
<GridColumn Field=@nameof(OrdersDetails.OrderDate) HeaderText="Order Date"
Format="d" Type="ColumnType.Date" Width="160"></GridColumn>
<GridColumn Field=@nameof(OrdersDetails.ShipCountry) HeaderText="Ship
Country" Width="150"></GridColumn>
</GridColumns>
</SfGrid>
@code{
public List<OrdersDetails> GridData = new List<OrdersDetails>
{
new OrdersDetails() { OrderID = 10248, CustomerID = "VINET", Freight =
32.38, ShipCity = "Berlin", OrderDate = DateTime.Now.AddDays(-2), ShipName =
"Vins et alcools Chevalier", ShipCountry = "Denmark", ShipAddress =
"Kirchgasse 6" },
new OrdersDetails() { OrderID = 10249, CustomerID = "TOMSP", Freight =
11.61, ShipCity = "Madrid", OrderDate = DateTime.Now.AddDays(-5), ShipName =
"Toms Spezialitäten", ShipCountry = "Brazil", ShipAddress = "Avda. Azteca
123" },
new OrdersDetails() { OrderID = 10250, CustomerID = "HANAR", Freight =
65.83, ShipCity = "Cholchester", OrderDate = DateTime.Now.AddDays(-12),

```

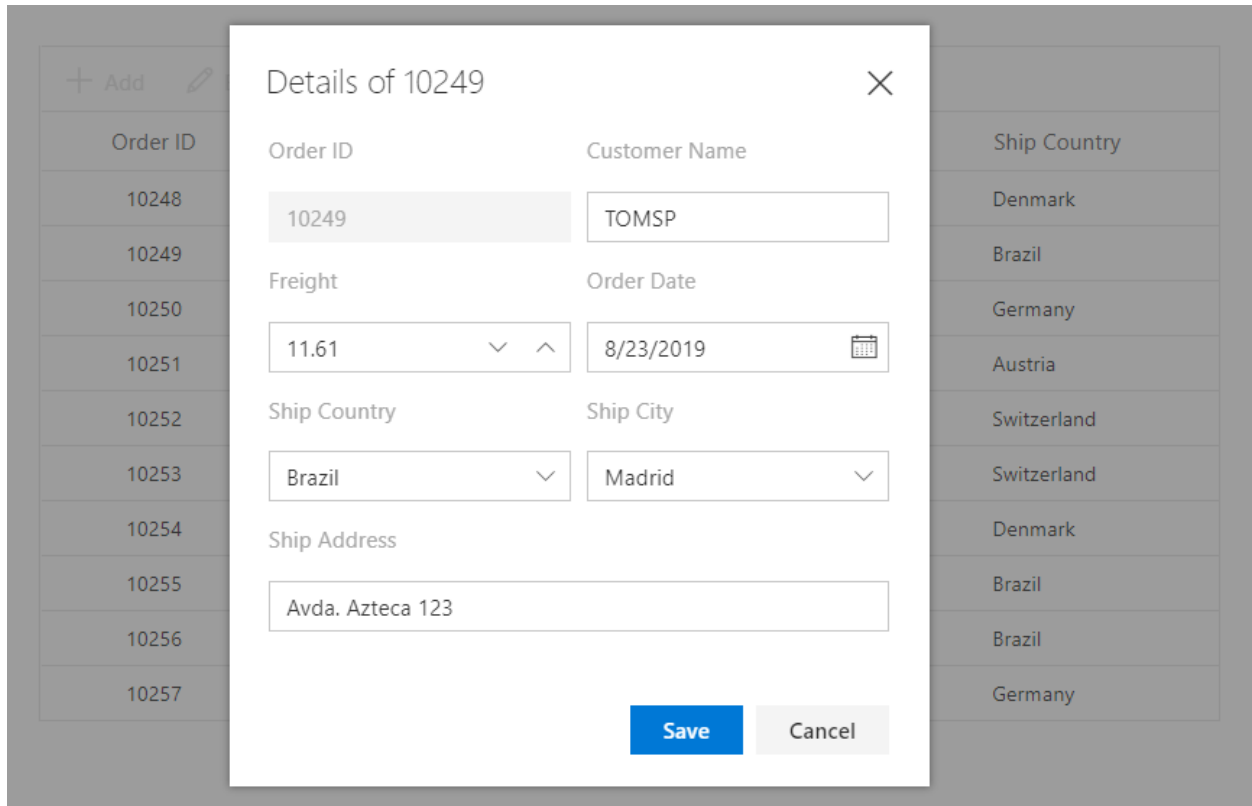
```

ShipName = "Hanari Carnes", ShipCountry = "Germany", ShipAddress = "Carrera
52 con Ave. Bolívar #65-98 Llano Largo" },
new OrdersDetails() { OrderID = 10251, CustomerID = "VICTE", Freight =
41.34, ShipCity = "Marseille", OrderDate = DateTime.Now.AddDays(-18),
ShipName = "Victuailles en stock", ShipCountry = "Austria", ShipAddress =
"Magazinweg 7" },
new OrdersDetails() { OrderID = 10252, CustomerID = "SUPRD", Freight = 51.3,
ShipCity = "Tsawassen", OrderDate = DateTime.Now.AddDays(-22), ShipName =
"Suprêmes délices", ShipCountry = "Switzerland", ShipAddress = "1029 - 12th
Ave. S." },
new OrdersDetails() { OrderID = 10253, CustomerID = "HANAR", Freight =
58.17, ShipCity = "Tsawassen", OrderDate = DateTime.Now.AddDays(-26),
ShipName = "Hanari Carnes", ShipCountry = "Switzerland", ShipAddress = "1029
- 12th Ave. S." },
new OrdersDetails() { OrderID = 10254, CustomerID = "CHOPS", Freight =
22.98, ShipCity = "Berlin", OrderDate = DateTime.Now.AddDays(-34), ShipName
= "Chop-suey Chinese", ShipCountry = "Denmark", ShipAddress = "Kirchgasse 6"
},
new OrdersDetails() { OrderID = 10255, CustomerID = "RICSU", Freight =
148.33, ShipCity = "Madrid", OrderDate = DateTime.Now.AddDays(-39), ShipName
= "Richter Supermarket", ShipCountry = "Brazil", ShipAddress = "Avda. Azteca
123" },
new OrdersDetails() { OrderID = 10256, CustomerID = "WELLI", Freight =
13.97, ShipCity = "Madrid", OrderDate = DateTime.Now.AddDays(-43), ShipName
= "Wellington Importadora", ShipCountry = "Brazil", ShipAddress = "Avda.
Azteca 123" },
new OrdersDetails() { OrderID = 10257, CustomerID = "HILAA", Freight =
81.91, ShipCity = "Cholchester", OrderDate = DateTime.Now.AddDays(-48),
ShipName = "HILARION-Abastos", ShipCountry = "Germany", ShipAddress =
"Carrera 52 con Ave. Bolívar #65-98 Llano Largo" }
};
public class OrdersDetails
{
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public double? Freight { get; set; }
public string ShipCity { get; set; }
public DateTime? OrderDate { get; set; }
public string ShipName { get; set; }
public string ShipCountry { get; set; }
public string ShipAddress { get; set; }
}
}
<style>
.form-group.col-md-6 {
width: 200px;
}
label.e-float-text {
position: relative;
padding-left: 0;
top: 10%;
}
</style>

```

In the above sample code, the textbox rendered for **OrderID** column inside the dialog template is disabled using its `Enabled` property to prevent editing of the primary key column.

The following image represents the dialog template that is displayed on double-clicking a DataGrid cell,



Disable components in dialog template

It is possible to disable particular components rendered inside the dialog template using the data source value. This can be achieved by utilizing the `Enabled` property of the components which specifies whether the component is enabled or disabled.

This is demonstrated in the below sample code where if the `RequestType` argument value of the `OnActionBegin` event is `BeginEdit` then the `Enabled` property of the **OrderID** Textbox is set to false.

ASPX-CS

```
@using Syncfusion.Blazor.Grids
@using Syncfusion.Blazor.Calendars
@using Syncfusion.Blazor.DropDowns
@using Syncfusion.Blazor.Inputs
<SfGrid DataSource="@GridData" Toolbar="@ (new string[] { "Add", "Edit"
, "Delete", "Update", "Cancel" }) ">
<GridEvents OnActionBegin="ActionBeginHandler"
TValue="OrdersDetails"></GridEvents>
<GridEditSettings AllowAdding="true" AllowEditing="true"
AllowDeleting="true" Mode="@EditMode.Dialog">
<Template>
@{
var Order = (context as OrdersDetails);
<div>
```



```

<div class="form-row">
<div class="form-group col-md-6">
<label class="e-float-text e-label-top">Order ID</label>
<SfNumericTextBox ID="OrderID" @bind-Value="@ (Order.OrderID) "
Enabled="@Enabled"></SfNumericTextBox>
</div>
<div class="form-group col-md-6">
<label class="e-float-text e-label-top">Customer Name</label>
<SfAutoComplete ID="CustomerID" TItem="OrdersDetails" @bind-
Value="@ (Order.CustomerID) " TValue="string" DataSource="@GridData">
<AutoCompleteFieldSettings Value="CustomerID"></AutoCompleteFieldSettings>
</SfAutoComplete>
</div>
</div>
<div class="form-row">
<div class="form-group col-md-6">
<label class="e-float-text e-label-top">Freight</label>
<SfNumericTextBox ID="Freight" @bind-Value="@ (Order.Freight) "
TValue="double"></SfNumericTextBox>
</div>
<div class="form-group col-md-6">
<label class="e-float-text e-label-top">Order Date</label>
<SfDatePicker ID="OrderDate" @bind-
Value="@ (Order.OrderDate) "></SfDatePicker>
</div>
</div>
<div class="form-row">
<div class="form-group col-md-6">
<label class="e-float-text e-label-top">Ship Country</label>
<SfDropDownList ID="ShipCountry" @bind-Value="@ (Order.ShipCountry) "
TItem="OrdersDetails" TValue="string" DataSource="@GridData">
<DropDownListFieldSettings Value="ShipCountry"
Text="ShipCountry"></DropDownListFieldSettings>
</SfDropDownList>
</div>
<div class="form-group col-md-6">
<label class="e-float-text e-label-top">Ship City</label>
<SfDropDownList ID="ShipCity" @bind-Value="@ (Order.ShipCity) "
TItem="OrdersDetails" TValue="string" DataSource="@GridData">
<DropDownListFieldSettings Value="ShipCity"
Text="ShipCity"></DropDownListFieldSettings>
</SfDropDownList>
</div>
</div>
<div class="form-row">
<div class="form-group col-md-12">
<label class="e-float-text e-label-top">Ship Address</label>
<SfTextBox ID="ShipAddress" @bind-Value="@ (Order.ShipAddress) "></SfTextBox>
</div>
</div>
</div>
}
</Template>
</GridEditSettings>
</GridColumn>

```

```

<GridColumn Field=@nameof(OrdersDetails.OrderID) HeaderText="Order ID"
IsPrimaryKey="true" TextAlign="@TextAlign.Center"
HeaderTextAlign="@TextAlign.Center" Width="140"></GridColumn>
<GridColumn Field=@nameof(OrdersDetails.CustomerID) HeaderText="Customer
Name" Width="150"></GridColumn>
<GridColumn Field=@nameof(OrdersDetails.Freight) HeaderText="Freight"
Format="C2" Width="140" TextAlign="@TextAlign.Right"
HeaderTextAlign="@TextAlign.Right"></GridColumn>
<GridColumn Field=@nameof(OrdersDetails.OrderDate) HeaderText="Order Date"
Format="d" Type="ColumnType.Date" Width="160"></GridColumn>
<GridColumn Field=@nameof(OrdersDetails.ShipCountry) HeaderText="Ship
Country" Width="150"></GridColumn>
</GridColumns>
</SfGrid>
@code{
public bool Enabled = true;
public List<OrdersDetails> GridData = new List<OrdersDetails>
{
new OrdersDetails() { OrderID = 10248, CustomerID = "VINET", Freight =
32.38, ShipCity = "Berlin", OrderDate = DateTime.Now.AddDays(-2), ShipName =
"Vins et alcools Chevalier", ShipCountry = "Denmark", ShipAddress =
"Kirchgasse 6" },
new OrdersDetails() { OrderID = 10249, CustomerID = "TOMSP", Freight =
11.61, ShipCity = "Madrid", OrderDate = DateTime.Now.AddDays(-5), ShipName =
"Toms Spezialitäten", ShipCountry = "Brazil", ShipAddress = "Avda. Azteca
123" },
new OrdersDetails() { OrderID = 10250, CustomerID = "HANAR", Freight =
65.83, ShipCity = "Cholchester", OrderDate = DateTime.Now.AddDays(-12),
ShipName = "Hanari Carnes", ShipCountry = "Germany", ShipAddress = "Carrera
52 con Ave. Bolívar #65-98 Llano Largo" },
new OrdersDetails() { OrderID = 10251, CustomerID = "VICTE", Freight =
41.34, ShipCity = "Marseille", OrderDate = DateTime.Now.AddDays(-18),
ShipName = "Victuailles en stock", ShipCountry = "Austria", ShipAddress =
"Magazinweg 7" },
new OrdersDetails() { OrderID = 10252, CustomerID = "SUPRD", Freight = 51.3,
ShipCity = "Tsawassen", OrderDate = DateTime.Now.AddDays(-22), ShipName =
"Suprêmes délices", ShipCountry = "Switzerland", ShipAddress = "1029 - 12th
Ave. S." },
new OrdersDetails() { OrderID = 10253, CustomerID = "HANAR", Freight =
58.17, ShipCity = "Tsawassen", OrderDate = DateTime.Now.AddDays(-26),
ShipName = "Hanari Carnes", ShipCountry = "Switzerland", ShipAddress = "1029
- 12th Ave. S." },
new OrdersDetails() { OrderID = 10254, CustomerID = "CHOPS", Freight =
22.98, ShipCity = "Berlin", OrderDate = DateTime.Now.AddDays(-34), ShipName =
"Chop-suey Chinese", ShipCountry = "Denmark", ShipAddress = "Kirchgasse 6"
},
new OrdersDetails() { OrderID = 10255, CustomerID = "RICSU", Freight =
148.33, ShipCity = "Madrid", OrderDate = DateTime.Now.AddDays(-39), ShipName =
"Richter Supermarket", ShipCountry = "Brazil", ShipAddress = "Avda. Azteca
123" },
new OrdersDetails() { OrderID = 10256, CustomerID = "WELLI", Freight =
13.97, ShipCity = "Madrid", OrderDate = DateTime.Now.AddDays(-43), ShipName =
"Wellington Importadora", ShipCountry = "Brazil", ShipAddress = "Avda.
Azteca 123" },
new OrdersDetails() { OrderID = 10257, CustomerID = "HILAA", Freight =
81.91, ShipCity = "Cholchester", OrderDate = DateTime.Now.AddDays(-48),

```

```
ShipName = "HILARION-Abastos", ShipCountry = "Germany", ShipAddress =  
"Carrera 52 con Ave. Bolívar #65-98 Llano Largo" }  
};  
public class OrdersDetails  
{  
    public int OrderID { get; set; }  
    public string CustomerID { get; set; }  
    public double Freight { get; set; }  
    public string ShipCity { get; set; }  
    public DateTime? OrderDate { get; set; }  
    public string ShipName { get; set; }  
    public string ShipCountry { get; set; }  
    public string ShipAddress { get; set; }  
}  
public void ActionBeginHandler(ActionEventArgs<OrdersDetails> args)  
{  
    if (args.RequestType == Syncfusion.Blazor.Grids.Action.BeginEdit)  
    {  
        // The Textbox component is disabled using its Enabled property  
        this.Enabled = false;  
    }  
    else  
    {  
        this.Enabled = true;  
    }  
}  
<style>  
.form-group.col-md-6 {  
    width: 200px;  
}  
label.e-float-text {  
    position: relative;  
    padding-left: 0;  
    top: 10%;  
}  
</style>
```

The following image represents the dialog template of the DataGrid component with disabled components,

| Order ID | Customer Name | Ship Country | Ship City |
|----------|---------------|--------------|-----------|
| 10248 | VINET | Denmark | Berlin |
| 10249 | | Brazil | |
| 10250 | | Germany | |
| 10251 | | Austria | |
| 10252 | | Switzerland | |
| 10253 | | Switzerland | |
| 10254 | | Denmark | |
| 10255 | | Brazil | |
| 10256 | | Brazil | |
| 10257 | | Germany | |

Details of 10248

Order ID: 10248 Customer Name: VINET

Freight: 32.38 Order Date: 8/26/2019

Ship Country: Denmark Ship City: Berlin

Ship Address: Kirchgasse 6

Save Cancel

Set focus to editor

By default, the first input element in the dialog will be focused while opening it. If the first input element is in a disabled or hidden state, you can set focus to the required input element in the corresponding components **Created** or **DataBound** event.

This is demonstrated in the below sample code where the first input element is in disabled state. So the **CustomerID** Autocomplete component is focused by invoking its [FocusIn](#) method in the AutoComplete's [DataBound](#) event.

ASPX-CS

```
@using Syncfusion.Blazor.Grids
@using Syncfusion.Blazor.Calendars
@using Syncfusion.Blazor.DropDowns
@using Syncfusion.Blazor.Inputs
<SfGrid DataSource="@GridData" Toolbar="@ (new string[] { "Add", "Edit",
    "Delete", "Update", "Cancel" }) ">
<GridEditSettings AllowAdding="true" AllowEditing="true"
    AllowDeleting="true" Mode="@EditMode.Dialog">
<Template>
@{
    var Order = (context as OrdersDetails);
<div>
<div class="form-row">
<div class="form-group col-md-6">
<label class="e-float-text e-label-top">Order ID</label>
<SfNumericTextBox ID="OrderID" @bind-Value="@ (Order.OrderID) "
    Enabled="@ ((Order.OrderID == null) ? true : false)"></SfNumericTextBox>
</div>
```

```

<div class="form-group col-md-6">
<label class="e-float-text e-label-top">Customer Name</label>
<SfAutoComplete ID="customerID" TItem="OrdersDetails" @bind-
Value="@ (Order.CustomerID)" TValue="string" DataSource="@GridData">
<AutoCompleteFieldSettings Value="CustomerID"></AutoCompleteFieldSettings>
<AutoCompleteEvents TValue="string" TItem="OrdersDetails"
DataBound="FocusAutoComplete"></AutoCompleteEvents>
</SfAutoComplete>
</div>
</div>
<div class="form-row">
<div class="form-group col-md-6">
<label class="e-float-text e-label-top">Freight</label>
<SfNumericTextBox ID="Freight" @bind-Value="@ (Order.Freight)"
TValue="double?"></SfNumericTextBox>
</div>
<div class="form-group col-md-6">
<label class="e-float-text e-label-top">Order Date</label>
<SfDatePicker ID="OrderDate" @bind-
Value="@ (Order.OrderDate)"></SfDatePicker>
</div>
</div>
<div class="form-row">
<div class="form-group col-md-6">
<label class="e-float-text e-label-top">Ship Country</label>
<SfDropDownList ID="ShipCountry" @bind-Value="@ (Order.ShipCountry)"
TItem="OrdersDetails" TValue="string" DataSource="@GridData">
<DropDownListFieldSettings Value="ShipCountry"
Text="ShipCountry"></DropDownListFieldSettings>
</SfDropDownList>
</div>
<div class="form-group col-md-6">
<label class="e-float-text e-label-top">Ship City</label>
<SfDropDownList ID="ShipCity" @bind-Value="@ (Order.ShipCity)"
TItem="OrdersDetails" TValue="string" DataSource="@GridData">
<DropDownListFieldSettings Value="ShipCity"
Text="ShipCity"></DropDownListFieldSettings>
</SfDropDownList>
</div>
</div>
<div class="form-row">
<div class="form-group col-md-12">
<label class="e-float-text e-label-top">Ship Address</label>
<SfTextBox ID="ShipAddress" @bind-Value="@ (Order.ShipAddress)"></SfTextBox>
</div>
</div>
</div>
}
</Template>
</GridEditSettings>
<GridColumns>
<GridColumn Field=@nameof(OrdersDetails.OrderID) HeaderText="Order ID"
IsPrimaryKey="true" TextAlign="@TextAlign.Center"
HeaderTextAlign="@TextAlign.Center" Width="140"></GridColumn>
<GridColumn Field=@nameof(OrdersDetails.CustomerID) HeaderText="Customer
Name" Width="150"></GridColumn>

```

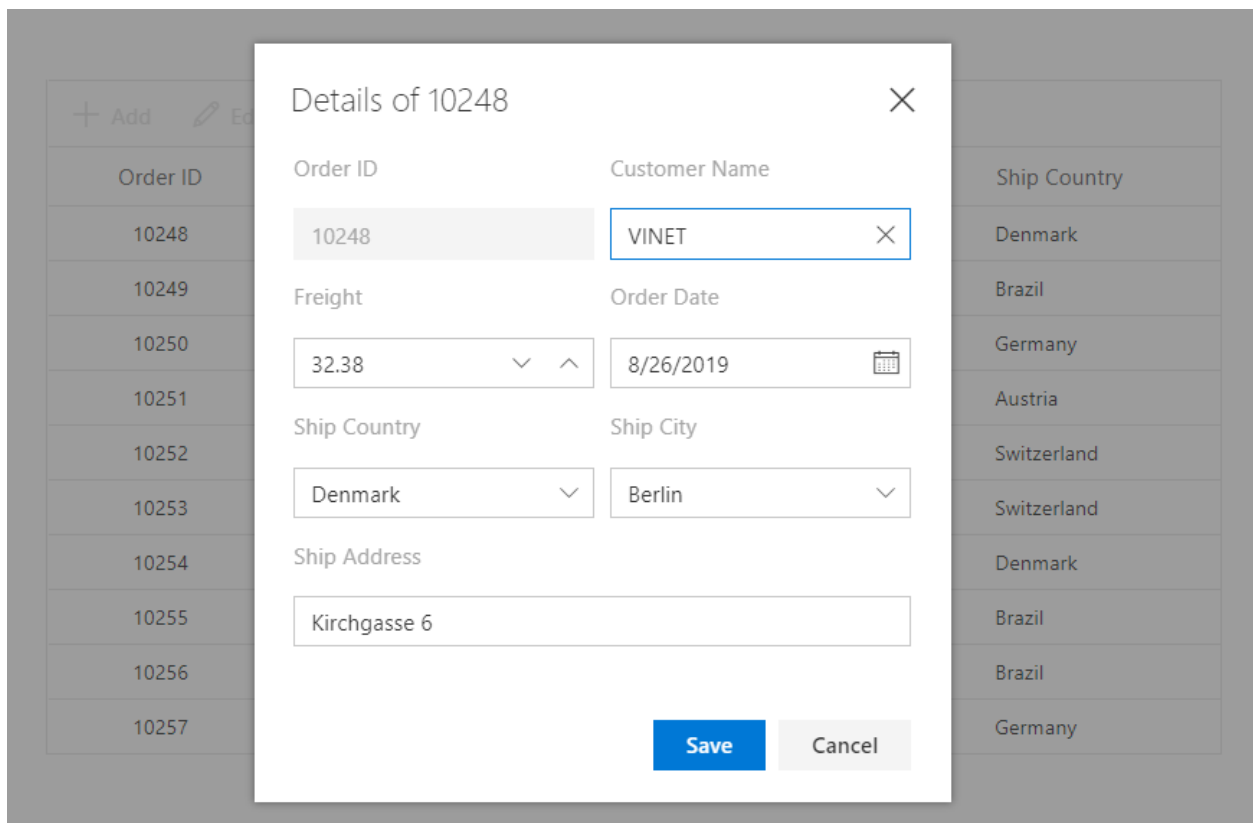
```

<GridColumn Field=@nameof(OrdersDetails.Freight) HeaderText="Freight"
Format="C2" Width="140" TextAlign="@TextAlign.Right"
HeaderTextAlign="@TextAlign.Right"></GridColumn>
<GridColumn Field=@nameof(OrdersDetails.OrderDate) HeaderText="Order Date"
Format="d" Type="ColumnType.Date" Width="160"></GridColumn>
<GridColumn Field=@nameof(OrdersDetails.ShipCountry) HeaderText="Ship
Country" Width="150"></GridColumn>
</GridColumn>
</SfGrid>
@code{
SfAutoComplete<string, OrdersDetails> AutoComplete { get; set; }
public List<OrdersDetails> GridData = new List<OrdersDetails>
{
new OrdersDetails() { OrderID = 10248, CustomerID = "VINET", Freight =
32.38, ShipCity = "Berlin", OrderDate = DateTime.Now.AddDays(-2), ShipName =
"Vins et alcools Chevalier", ShipCountry = "Denmark", ShipAddress =
"Kirchgasse 6" },
new OrdersDetails() { OrderID = 10249, CustomerID = "TOMSP", Freight =
11.61, ShipCity = "Madrid", OrderDate = DateTime.Now.AddDays(-5), ShipName =
"Toms Spezialitäten", ShipCountry = "Brazil", ShipAddress = "Avda. Azteca
123" },
new OrdersDetails() { OrderID = 10250, CustomerID = "HANAR", Freight =
65.83, ShipCity = "Cholchester", OrderDate = DateTime.Now.AddDays(-12),
ShipName = "Hanari Carnes", ShipCountry = "Germany", ShipAddress = "Carrera
52 con Ave. Bolívar #65-98 Llano Largo" },
new OrdersDetails() { OrderID = 10251, CustomerID = "VICTE", Freight =
41.34, ShipCity = "Marseille", OrderDate = DateTime.Now.AddDays(-18),
ShipName = "Victuailles en stock", ShipCountry = "Austria", ShipAddress =
"Magazinweg 7" },
new OrdersDetails() { OrderID = 10252, CustomerID = "SUPRD", Freight = 51.3,
ShipCity = "Tsawassen", OrderDate = DateTime.Now.AddDays(-22), ShipName =
"Suprêmes délices", ShipCountry = "Switzerland", ShipAddress = "1029 - 12th
Ave. S." },
new OrdersDetails() { OrderID = 10253, CustomerID = "HANAR", Freight =
58.17, ShipCity = "Tsawassen", OrderDate = DateTime.Now.AddDays(-26),
ShipName = "Hanari Carnes", ShipCountry = "Switzerland", ShipAddress = "1029
- 12th Ave. S." },
new OrdersDetails() { OrderID = 10254, CustomerID = "CHOPS", Freight =
22.98, ShipCity = "Berlin", OrderDate = DateTime.Now.AddDays(-34), ShipName
= "Chop-suey Chinese", ShipCountry = "Denmark", ShipAddress = "Kirchgasse 6"
},
new OrdersDetails() { OrderID = 10255, CustomerID = "RICSU", Freight =
148.33, ShipCity = "Madrid", OrderDate = DateTime.Now.AddDays(-39), ShipName
= "Richter Supermarket", ShipCountry = "Brazil", ShipAddress = "Avda. Azteca
123" },
new OrdersDetails() { OrderID = 10256, CustomerID = "WELLI", Freight =
13.97, ShipCity = "Madrid", OrderDate = DateTime.Now.AddDays(-43), ShipName
= "Wellington Importadora", ShipCountry = "Brazil", ShipAddress = "Avda.
Azteca 123" },
new OrdersDetails() { OrderID = 10257, CustomerID = "HILAA", Freight =
81.91, ShipCity = "Cholchester", OrderDate = DateTime.Now.AddDays(-48),
ShipName = "HILARION-Abastos", ShipCountry = "Germany", ShipAddress =
"Carrera 52 con Ave. Bolívar #65-98 Llano Largo" }
};
public class OrdersDetails
{
public int? OrderID { get; set; }

```

```
public string CustomerID { get; set; }
public double? Freight { get; set; }
public string ShipCity { get; set; }
public DateTime? OrderDate { get; set; }
public string ShipName { get; set; }
public string ShipCountry { get; set; }
public string ShipAddress { get; set; }
}
private async Task FocusAutoComplete() {
    await this.AutoComplete.FocusIn();
}
}
<style>
.form-group.col-md-6 {
width: 200px;
}
label.e-float-text {
position: relative;
padding-left: 0;
top: 10%;
}
</style>
```

The following image represents the AutoComplete component in focused state inside the dialog template of the DataGrid component,



Inline Template

Before adding an Inline template to the DataGrid, we strongly recommend you to go through the [Template](#) section topic to configure the template.

The Inline template editing provides an option to customize the default behavior of Inline editing. Using the Inline template, you can render your editors by defining the [GridEditSettings](#) component's [Mode](#) property as **Normal** and wrapping the HTML elements inside the [Template](#) property of [GridEditSettings](#).

Custom components inside the Inline Template must be specified with two-way (**@bind-Value**) binding to reflect the changes in DataGrid.

In some cases, you would need to add new field editors in the Inline editing which are not present in the column model. In that case, the Inline template editing will help you to customize the default editing.

The following sample code demonstrates DataGrid enabled with Inline template editing,

ASPX-CS

```
@using Syncfusion.Blazor.Grids
@using Syncfusion.Blazor.Calendars
@using Syncfusion.Blazor.DropDowns
@using Syncfusion.Blazor.Inputs
<SfGrid DataSource="@GridData" Toolbar="@ (new string[] { "Add", "Edit"
, "Delete", "Update", "Cancel" }) ">
<GridEditSettings AllowAdding="true" AllowEditing="true"
AllowDeleting="true" Mode="@EditMode.Normal">
<Template>
@{
var Order = (context as OrdersDetails);
<div>
<div class="form-row">
<div class="form-group col-md-6">
<label class="e-float-text e-label-top">Order ID</label>
<SfNumericTextBox ID="OrderID" @bind-Value="@ (Order.OrderID) "
ShowSpinButton="false" Enabled="@ ((Order.OrderID == null)? true:
false) "></SfNumericTextBox>
</div>
<div class="form-group col-md-6">
<label class="e-float-text e-label-top">Customer Name</label>
<SfAutoComplete ID="CustomerID" TItem="OrdersDetails"
FloatLabelType="FloatLabelType.Auto" @bind-Value="@ (Order.CustomerID) "
TValue="string" DataSource="@GridData">
<AutoCompleteFieldSettings Value="CustomerID"></AutoCompleteFieldSettings>
</SfAutoComplete>
</div>
</div>
<div class="form-row">
<div class="form-group col-md-6">
<label class="e-float-text e-label-top">Freight</label>
<SfNumericTextBox ID="Freight" @bind-Value="@ (Order.Freight) "
TValue="double?"></SfNumericTextBox>
</div>
<div class="form-group col-md-6">
<label class="e-float-text e-label-top">Order Date</label>
<SfDatePicker ID="OrderDate" @bind-
Value="@ (Order.OrderDate) "></SfDatePicker>
</div>
</div>
```



```

</div>
<div class="form-row">
<div class="form-group col-md-6">
<label class="e-float-text e-label-top">Ship Country</label>
<SfDropDownList ID="ShipCountry" TItem="OrdersDetails" @bind-
Value="@ (Order.ShipCountry)" TValue="string" DataSource="@GridData">
<DropDownListFieldSettings Value="ShipCountry"
Text="ShipCountry"></DropDownListFieldSettings>
</SfDropDownList>
</div>
<div class="form-group col-md-6">
<label class="e-float-text e-label-top">Ship City</label>
<SfDropDownList ID="ShipCity" TItem="OrdersDetails" @bind-
Value="@ (Order.ShipCity)" TValue="string" DataSource="@GridData">
<DropDownListFieldSettings Value="ShipCity"
Text="ShipCity"></DropDownListFieldSettings>
</SfDropDownList>
</div>
</div>
<div class="form-row">
<div class="form-group col-md-12">
<label class="e-float-text e-label-top">Ship Address</label>
<SfTextBox ID="ShipAddress" @bind-Value="@ (Order.ShipAddress)"></SfTextBox>
</div>
</div>
</div>
}
</Template>
</GridEditSettings>
<GridColumns>
<GridColumn Field=@nameof(OrdersDetails.OrderID) HeaderText="Order ID"
IsPrimaryKey="true" TextAlign="@TextAlign.Center"
HeaderTextAlign="@TextAlign.Center" Width="140"></GridColumn>
<GridColumn Field=@nameof(OrdersDetails.CustomerID) HeaderText="Customer
Name" Width="150"></GridColumn>
<GridColumn Field=@nameof(OrdersDetails.Freight) HeaderText="Freight"
Format="C2" Width="140" TextAlign="@TextAlign.Right"
HeaderTextAlign="@TextAlign.Right"></GridColumn>
<GridColumn Field=@nameof(OrdersDetails.OrderDate) HeaderText="Order Date"
Format="d" Type="ColumnType.Date" Width="160"></GridColumn>
<GridColumn Field=@nameof(OrdersDetails.ShipCountry) HeaderText="Ship
Country" Width="150"></GridColumn>
</GridColumns>
</SfGrid>
@code{
public List<OrdersDetails> GridData = new List<OrdersDetails>
{
new OrdersDetails() { OrderID = 10248, CustomerID = "VINET", Freight =
32.38, ShipCity = "Berlin", OrderDate = DateTime.Now.AddDays(-2), ShipName =
"Vins et alcools Chevalier", ShipCountry = "Denmark", ShipAddress =
"Kirchgasse 6" },
new OrdersDetails() { OrderID = 10249, CustomerID = "TOMSP", Freight =
11.61, ShipCity = "Madrid", OrderDate = DateTime.Now.AddDays(-5), ShipName =
"Toms Spezialitäten", ShipCountry = "Brazil", ShipAddress = "Avda. Azteca
123" },
new OrdersDetails() { OrderID = 10250, CustomerID = "HANAR", Freight =
65.83, ShipCity = "Cholchester", OrderDate = DateTime.Now.AddDays(-12),

```

```

ShipName = "Hanari Carnes", ShipCountry = "Germany", ShipAddress = "Carrera
52 con Ave. Bolívar #65-98 Llano Largo" },
new OrdersDetails() { OrderID = 10251, CustomerID = "VICTE", Freight =
41.34, ShipCity = "Marseille", OrderDate = DateTime.Now.AddDays(-18),
ShipName = "Victuailles en stock", ShipCountry = "Austria", ShipAddress =
"Magazinweg 7" },
new OrdersDetails() { OrderID = 10252, CustomerID = "SUPRD", Freight = 51.3,
ShipCity = "Tsawassen", OrderDate = DateTime.Now.AddDays(-22), ShipName =
"Suprêmes délices", ShipCountry = "Switzerland", ShipAddress = "1029 - 12th
Ave. S." },
new OrdersDetails() { OrderID = 10253, CustomerID = "HANAR", Freight =
58.17, ShipCity = "Tsawassen", OrderDate = DateTime.Now.AddDays(-26),
ShipName = "Hanari Carnes", ShipCountry = "Switzerland", ShipAddress = "1029
- 12th Ave. S." },
new OrdersDetails() { OrderID = 10254, CustomerID = "CHOPS", Freight =
22.98, ShipCity = "Berlin", OrderDate = DateTime.Now.AddDays(-34), ShipName
= "Chop-suey Chinese", ShipCountry = "Denmark", ShipAddress = "Kirchgasse 6"
},
new OrdersDetails() { OrderID = 10255, CustomerID = "RICSU", Freight =
148.33, ShipCity = "Madrid", OrderDate = DateTime.Now.AddDays(-39), ShipName
= "Richter Supermarket", ShipCountry = "Brazil", ShipAddress = "Avda. Azteca
123" },
new OrdersDetails() { OrderID = 10256, CustomerID = "WELLI", Freight =
13.97, ShipCity = "Madrid", OrderDate = DateTime.Now.AddDays(-43), ShipName
= "Wellington Importadora", ShipCountry = "Brazil", ShipAddress = "Avda.
Azteca 123" },
new OrdersDetails() { OrderID = 10257, CustomerID = "HILAA", Freight =
81.91, ShipCity = "Cholchester", OrderDate = DateTime.Now.AddDays(-48),
ShipName = "HILARION-Abastos", ShipCountry = "Germany", ShipAddress =
"Carrera 52 con Ave. Bolívar #65-98 Llano Largo" }
};
public class OrdersDetails
{
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public double? Freight { get; set; }
public string ShipCity { get; set; }
public DateTime? OrderDate { get; set; }
public string ShipName { get; set; }
public string ShipCountry { get; set; }
public string ShipAddress { get; set; }
}
}
<style>
.form-group.col-md-6 {
width: 200px;
}
label.e-float-text {
position: relative;
padding-left: 0;
top: 10%;
}
</style>

```

In the above sample code, the textbox rendered for **OrderID** column inside the Inline editing template is disabled using its `Enabled` property to prevent editing of the primary key column.

Adding a new row at the bottom of the datagrid

By default, a new row will be added at the top of the datagrid. You can change it by setting `NewRowPosition` property of the `GridEditSettings` component as **Bottom**.

The following sample code demonstrates changing the position of the new row that gets added in the DataGrid component,

ASPX-CS

```
@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders" Height="315" AllowPaging="true" Toolbar="@ (new
List<string>() { "Add", "Edit", "Delete", "Update", "Cancel" })">
<GridEditSettings AllowAdding="true" AllowEditing="true"
AllowDeleting="true"
NewRowPosition="NewRowPosition.Bottom"></GridEditSettings>
<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
IsPrimaryKey="true" TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
EditType="EditType.DatePickerEdit" Format="d" TextAlign="TextAlign.Right"
Width="130" Type="ColumnType.Date"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumns>
</SfGrid>
@code{
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
OrderDate = DateTime.Now.AddDays(-x),
}).ToList();
}
public class Order {
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
}
```

The following image represents the new row added at the bottom of the DataGrid,

| Add Edit Delete Update Cancel | | | | |
|---|----------------------|----------------------|----------------------|-------------------------|
| Order ID | Customer Name | Order Date | Freight | |
| 1005 | ALFKI | 8/24/2019 | \$10.50 | |
| 1006 | BOLID | 8/23/2019 | \$12.60 | |
| 1007 | ANTON | 8/22/2019 | \$14.70 | |
| 1008 | ALFKI | 8/21/2019 | \$16.80 | |
| 1009 | BLONP | 8/20/2019 | \$18.90 | |
| 1010 | ANANTR | 8/19/2019 | \$21.00 | |
| 1011 | ANANTR | 8/18/2019 | \$23.10 | |
| 1012 | BOLID | 8/17/2019 | \$25.20 | |
| <input type="text"/> | <input type="text"/> | <input type="text"/> | <input type="text"/> | |
| K < 1 2 3 4 5 6 7 > X | | | | 1 of 7 pages (75 items) |

* In Batch mode while in edit mode, you can add a new row at bottom using the TAB key when you are on the last cell of the last row.

Confirmation messages

It is possible to display confirmation dialog's on performing deletion or batch operations on the DataGrid records.

Delete confirmation

The delete confirms dialog can be shown on deleting a record by setting the [ShowDeleteConfirmDialog](#) property of the [GridEditSettings](#) component as **true**.

The following sample code demonstrates enabling delete confirmation dialog in the DataGrid component,

ASPX-CS

```
@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders" Height="315" AllowPaging="true" Toolbar="@ (new
List<string>() { "Add", "Edit", "Delete", "Update", "Cancel" })">
  <GridEditSettings AllowAdding="true" AllowEditing="true"
  AllowDeleting="true" ShowDeleteConfirmDialog="true"></GridEditSettings>
  <GridColumns>
    <GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
    IsPrimaryKey="true" TextAlign="TextAlign.Right" Width="120"></GridColumn>
    <GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
    Width="120"></GridColumn>
    <GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
    EditType="EditType.DatePickerEdit" Format="d" TextAlign="TextAlign.Right"
    Width="130" Type="ColumnType.Date"></GridColumn>
```

```

<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumn>
</SfGrid>
@code{
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
OrderDate = DateTime.Now.AddDays(-x),
}).ToList();
}
public class Order
{
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
}

```

The following GIF represents the delete confirmation dialog displayed while deleting a record in DataGrid,

| + Add ✎ Edit 🗑 Delete 📄 Update ✕ Cancel | | | | |
|---|---------------|------------|---------|--|
| Order ID | Customer Name | Order Date | Freight | |
| 1001 | ANANTR | 8/28/2019 | \$2.10 | |
| 1002 | ANTON | 8/27/2019 | \$4.20 | |
| 1003 | BLONP | 8/26/2019 | \$6.30 | |
| 1004 | ANTON | 8/25/2019 | \$8.40 | |
| 1005 | ANTON | 8/24/2019 | \$10.50 | |
| 1006 | ANTON | 8/23/2019 | \$12.60 | |
| 1007 | ALFKI | 8/22/2019 | \$14.70 | |
| 1008 | ANANTR | 8/21/2019 | \$16.80 | |
| 1009 | BLONP | 8/20/2019 | \$18.90 | |
| ⏪ < 1 2 3 4 5 6 7 > ⏩ 1 of 7 pages (75 items) | | | | |

The [ShowDeleteConfirmDialog](#) supports all type of edit modes.

Batch confirmation

The confirmation dialog can be enabled for all the batch operations by setting the [ShowConfirmDialog](#) property of the [GridEditSettings](#) component as **true**.

The following sample code demonstrates enabling confirmation dialog for batch operations in the DataGrid component,

ASPX-CS

```
@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders" Height="315" AllowPaging="true"
AllowSorting="true" Toolbar="@ (new List<string>() { "Add", "Edit", "Delete",
"Update", "Cancel" })">
  <GridEditSettings AllowAdding="true" AllowEditing="true"
AllowDeleting="true" ShowConfirmDialog="true" ShowDeleteConfirmDialog="true"
Mode="EditMode.Batch"></GridEditSettings>
  <GridColumns>
    <GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
IsPrimaryKey="true" TextAlign="TextAlign.Right" Width="120"></GridColumn>
    <GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="120"></GridColumn>
    <GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
EditType="EditType.DatePickerEdit" Format="d" TextAlign="TextAlign.Right"
Width="130" Type="ColumnType.Date"></GridColumn>
    <GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
  </GridColumns>
</SfGrid>
@code{
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
OrderDate = DateTime.Now.AddDays(-x),
}).ToList();
}
public class Order
{
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
}
```

The following GIF represents the confirmation dialog displayed while performing batch operations in DataGrid,

| + Add ✎ Edit 🗑 Delete 📄 Update ✕ Cancel | | | | |
|--|---------------|------------|---------|--|
| Order ID | Customer Name | Order Date | Freight | |
| 1001 | BOLID | 8/28/2019 | \$2.10 | |
| 1002 | ALFKI | 8/27/2019 | \$4.20 | |
| 1003 | ALFKI | 8/26/2019 | \$6.30 | |
| 1004 | ALFKI | 8/25/2019 | \$8.40 | |
| 1005 | BOLID | 8/24/2019 | \$10.50 | |
| 1006 | ANTON | 8/23/2019 | \$12.60 | |
| 1007 | BOLID | 8/22/2019 | \$14.70 | |
| 1008 | BOLID | 8/21/2019 | \$16.80 | |
| 1009 | BOLID | 8/20/2019 | \$18.90 | |
| K < 1 2 3 4 5 6 7 > X 1 of 7 pages (75 items) | | | | |

Enabling [ShowConfirmDialog](#) requires the [Mode](#) property value of the [GridEditSettings](#) component to be **Batch**.

If [ShowConfirmDialog](#) is set to false, then confirmation dialog will not be displayed on batch editing.

Default column values on adding new record

The datagrid provides an option to set the default value for the columns when adding a new record in it. To set a default value for a particular column you need to define it in the [DefaultValue](#) property of the [GridColumn](#) component.

The following sample code demonstrates setting default value as **ANTON** to the **CustomerID** column,

ASPX-CS

```
@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders" Height="315" AllowPaging="true" Toolbar="@ (new
List<string>() { "Add", "Edit", "Delete", "Update", "Cancel" })">
  <GridEditSettings AllowAdding="true" AllowEditing="true"
  AllowDeleting="true"></GridEditSettings>
  <GridColumn>
    <GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
    IsPrimaryKey="true" Width="120"></GridColumn>
    <GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
    Width="120" DefaultValue="@ ("ANTON") "></GridColumn>
    <GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
    EditType="EditType.DatePickerEdit" Format="d" TextAlign="TextAlign.Right"
    Width="130" Type="ColumnType.Date"></GridColumn>
    <GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
    TextAlign="TextAlign.Right" Width="120"></GridColumn>
  </GridColumn>
```

```

</SfGrid>
@code{
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
    Orders = Enumerable.Range(1, 75).Select(x => new Order()
    {
        OrderID = 1000 + x,
        CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
    })[new Random().Next(5)],
        Freight = 2.1 * x,
        OrderDate = DateTime.Now.AddDays(-x),
    }).ToList();
}
public class Order
{
    public int? OrderID { get; set; }
    public string CustomerID { get; set; }
    public DateTime? OrderDate { get; set; }
    public double? Freight { get; set; }
}
}

```

The following image represents the default value displayed in the **CustomerID** column while adding a new record in DataGrid,

| + Add ✎ Edit 🗑 Delete 💾 Update ✕ Cancel | | | | |
|--|---------------|----------------------|---------|-------------------------|
| Order ID | Customer Name | Order Date | Freight | |
| <input type="text"/> | ANTON | <input type="text"/> | | |
| 1001 | BOLID | 8/28/2019 | \$2.10 | |
| 1002 | BLONP | 8/27/2019 | \$4.20 | |
| 1003 | ANTON | 8/26/2019 | \$6.30 | |
| 1004 | ALFKI | 8/25/2019 | \$8.40 | |
| 1005 | ANANTR | 8/24/2019 | \$10.50 | |
| 1006 | ALFKI | 8/23/2019 | \$12.60 | |
| 1007 | ALFKI | 8/22/2019 | \$14.70 | |
| 1008 | ANANTR | 8/21/2019 | \$16.80 | |
| ⏪ < 1 2 3 4 5 6 7 > ⏩ | | | | 1 of 7 pages (75 items) |

Disable editing for particular column

You can disable editing for particular columns by setting value as **false** to the [AllowEditing](#) property of the [GridColumn](#) component.


The following sample code demonstrates editing disabled for the **CustomerID** column,

ASPX-CS

```
@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders" Height="315" AllowPaging="true" Toolbar="@ (new
List<string>() { "Add", "Edit", "Delete", "Update", "Cancel" })">
<GridEditSettings AllowAdding="true" AllowEditing="true"
AllowDeleting="true"></GridEditSettings>
<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
IsPrimaryKey="true" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="120" AllowEditing="false"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
EditType="EditType.DatePickerEdit" Format="d" TextAlign="TextAlign.Right"
Width="130" Type="ColumnType.Date"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumns>
</SfGrid>
@code{
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
OrderDate = DateTime.Now.AddDays(-x),
}).ToList();
}
public class Order
{
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
}
```

Similarly [AllowAdding](#) property at the column level helps us to disable the particular column from inserting value to it.

The following screenshot represents the editing disabled for the **CustomerID** column in DataGrid,

| + Add ✎ Edit 🗑 Delete 💾 Update ✕ Cancel | | | | |
|---|---------------|---|---------|--|
| Order ID | Customer Name | Order Date | Freight | |
| 1001 | ANANTR | 8/28/2019  | 2.1 | |
| 1002 | BLONP | 8/27/2019 | \$4.20 | |
| 1003 | BLONP | 8/26/2019 | \$6.30 | |
| 1004 | ALFKI | 8/25/2019 | \$8.40 | |
| 1005 | ANTON | 8/24/2019 | \$10.50 | |
| 1006 | BLONP | 8/23/2019 | \$12.60 | |
| 1007 | ANTON | 8/22/2019 | \$14.70 | |
| 1008 | BLONP | 8/21/2019 | \$16.80 | |
| 1009 | ALFKI | 8/20/2019 | \$18.90 | |
| ⏪ < 1 2 3 4 5 6 7 > ⏩ 1 of 7 pages (75 items) | | | | |

Troubleshoot: Editing works only for first row

The Editing functionalities can be performed based upon the primary key value of the selected row. If **PrimaryKey** is not defined in the datagrid, then edit or delete action take places in the first row.

Event trace while editing

While editing operation is getting executed the following events will be notified,

- OnActionBegin
- OnActionComplete

In both these events the type of editing operation is returned in the **RequestType** parameter of the event arguments. In addition to this, the event arguments also return the edited row data.

The **RequestType** values for the editing operations are listed in the below table,

| RequestType | OnActionBegin | OnActionComplete |
|-------------|---------------------------------|--------------------------------------|
| BeginEdit | Before editing operation begins | After editing operation is completed |
| Add | Before add operation begins | After add operation is completed |
| Delete | Before delete operation begins | After delete operation is completed |
| Save | Before save operation begins | After save operation is completed |
| Cancel | Before cancel operation begins | After cancel operation is completed |

The following sample code demonstrates the different **RequestType** parameters returned while performing editing operations in the OnActionBegin and OnActionComplete event,

ASPX-CS

```
@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders" AllowPaging="true" Toolbar="@ (new
List<string>() { "Add", "Edit", "Delete", "Cancel", "Update" })"
Height="315">
  <GridEditSettings AllowAdding="true" AllowEditing="true"
  AllowDeleting="true"></GridEditSettings>
  <GridEvents OnActionBegin="ActionBegin" OnActionComplete="ActionComplete"
  TValue="Order"></GridEvents>
  <GridColumns>
    <GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
    IsPrimaryKey="true" Width="120"></GridColumn>
    <GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
    Width="120"></GridColumn>
    <GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
    EditType="EditType.DatePickerEdit" Format="d" TextAlign="TextAlign.Right"
    Width="130" Type="ColumnType.Date"></GridColumn>
    <GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
    TextAlign="TextAlign.Right" Width="120"></GridColumn>
  </GridColumns>
</SfGrid>
@code{
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
  Orders = Enumerable.Range(1, 75).Select(x => new Order()
  {
    OrderID = 1000 + x,
    CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
  })[new Random().Next(5)],
    Freight = 2.1 * x,
    OrderDate = DateTime.Now.AddDays(-x),
  }).ToList();
}
public class Order
{
  public int? OrderID { get; set; }
  public string CustomerID { get; set; }
  public DateTime? OrderDate { get; set; }
  public double? Freight { get; set; }
}
public void ActionBegin(ActionEventArgs<Order> args)
{
  if (args.RequestType == Syncfusion.Blazor.Grids.Action.BeginEdit)
  {
    // Triggers before editing operation starts
  }
  else if (args.RequestType == Syncfusion.Blazor.Grids.Action.Add)
  {
    // Triggers before add operation starts
  }
  else if (args.RequestType == Syncfusion.Blazor.Grids.Action.Cancel)
  {

```

```
// Triggers before cancel operation starts
}
else if (args.RequestType == Syncfusion.Blazor.Grids.Action.Save)
{
// Triggers before save operation starts
}
else if (args.RequestType == Syncfusion.Blazor.Grids.Action.Delete)
{
// Triggers before delete operation starts
}
}
public void ActionComplete(ActionEventArgs<Order> args)
{
if (args.RequestType == Syncfusion.Blazor.Grids.Action.BeginEdit)
{
// Triggers once editing operation completes
}
else if (args.RequestType == Syncfusion.Blazor.Grids.Action.Add)
{
// Triggers once add operation completes
}
else if (args.RequestType == Syncfusion.Blazor.Grids.Action.Cancel)
{
// Triggers once cancel operation completes
}
else if (args.RequestType == Syncfusion.Blazor.Grids.Action.Save)
{
// Triggers once save operation completes
}
else if (args.RequestType == Syncfusion.Blazor.Grids.Action.Delete)
{
// Triggers once delete operation completes
}
}
}
```

Customize the edit dialog

You can use [HeaderTemplate](#) and [FooterTemplate](#) of the [GridEditSettings](#) component to customize the appearance of edit dialog.

In the below example we have changed the dialog's header text and footer button content for editing and adding records.

ASPX-CS

```
@using Syncfusion.Blazor.Buttons
@using Syncfusion.Blazor.Grids
<SfGrid @ref="Grid" DataSource="@Orders" AllowPaging="true" Toolbar="@ (new
List<string>() { "Add", "Edit", "Delete", "Update", "Cancel" })"
Height="315">
<GridEditSettings AllowAdding="true" AllowEditing="true"
AllowDeleting="true" Mode="EditMode.Dialog">
<HeaderTemplate>
@{
var text = GetHeader((context as Order));
<span>@text</span>
```

```

}
</HeaderTemplate>
<FooterTemplate>
<SfButton OnClick="@Save" IsPrimary="true">@ButtonText</SfButton>
<SfButton OnClick="@Cancel">Cancel</SfButton>
</FooterTemplate>
</GridEditSettings>
<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
IsPrimaryKey="true" ValidationRules="@ (new ValidationRules { Required = true
})" TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
ValidationRules="@ (new ValidationRules { Required = true })"
Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
EditType="EditType.DatePickerEdit" Format="d" TextAlign="TextAlign.Right"
Width="130" Type="ColumnType.Date"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" EditType="EditType.NumericEdit"
Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.ShipCountry) HeaderText="Ship Country"
EditType="EditType.DropDownEdit" Width="150"></GridColumn>
</GridColumns>
</SfGrid>
@code{
SfGrid<Order> Grid { get; set; }
public List<Order> Orders { get; set; }
public string Header { get; set; }
public string ButtonText { get; set; }
public string GetHeader(Order Value)
{
if (Value.OrderID == null)
{
ButtonText = "Insert";
return "Insert New Order";
}
else
{
ButtonText = "Save Changes";
return "Edit Record Details of " + Value.OrderID.ToString();
}
}
public async Task Cancel()
{
await Grid.CloseEdit(); //Cancel editing action
}
public async Task Save()
{
await Grid.EndEdit(); //Save the edited/added data to Grid
}
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],

```

```

Freight = 2.1 * x,
OrderDate = DateTime.Now.AddDays(-x),
ShipCountry = (new string[] { "USA", "UK", "CHINA", "RUSSIA", "INDIA" })[new
Random().Next(5)]
}).ToList();
}
public class Order
{
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
public string ShipCountry { get; set; }
}
}

```

You can refer to our [Blazor DataGrid](#) feature tour page for its groundbreaking feature representations. You can also explore our [Blazor DataGrid example](#) to understand how to present and manipulate data.

Perform CRUD operation for complex object using EditTemplate

Before performing CRUD operations with complex Objects, we recommend you to go through the [Complex DataBinding](#) documentation.

To customize the default Grid EditForm TextBox component, we will use [EditTemplate](#) to customize the Grid EditForm.

You can edit the complex objects using EditTemplate inside the GridColumn component by defining two-way (**@bind-Value**) binding to reflect the changes in DataGrid.

For focus and Column Validation to work properly, you have to define the **ID** property EditTemplate components with a value similar to GridColumn Field property value.

Also ensure to define **ID** property for the complex column as () replacing the (.) operator in the Field value.

The following sample code demonstrates the CRUD operation for complex objects with EditTemplate.

ASPX-CS

```

@using Syncfusion.Blazor.Grids
@using Syncfusion.Blazor.DropDowns
@using Syncfusion.Blazor.Inputs
<SfGrid DataSource="@Employees" Height="315" Toolbar="@ (new List<string>() {
"Add", "Edit", "Delete", "Update", "Cancel" }) ">
<GridEditSettings AllowAdding="true" AllowEditing="true"
AllowDeleting="true" Mode="EditMode.Normal"></GridEditSettings>
<GridColumn>
<GridColumn Field=@nameof(EmployeeData.CustomerID) HeaderText="CustomerID"
IsPrimaryKey="true" TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field="Data.EmployeeID" HeaderText="EmployeeID"
ValidationRules="@ (new ValidationRules{ Required=true, Number=true, Range =
new double[] {1, 2000}})" Width="130">
<EditTemplate>
@{
<SfNumericTextBox ID="Data__EmployeeID" @bind-Value="@ ((context as
EmployeeData).Data.EmployeeID)"></SfNumericTextBox>

```

```

}
</EditTemplate>
</GridColumn>
<GridColumn Field="Data.FirstName" HeaderText="First Name"
ValidationRules="@ (new ValidationRules{ Required=true})" Width="150">
<EditTemplate>
@{
<SfDropDownList ID="Data__FirstName" @bind-Value="@((context as
EmployeeData).Data.FirstName)" TItem="EmployeeData" TValue="string"
DataSource="@Employees">
<DropDownListFieldSettings Value="Data.FirstName"
Text="Data.FirstName"></DropDownListFieldSettings>
</SfDropDownList>
}
</EditTemplate>
</GridColumn>
<GridColumn Field=@nameof(EmployeeData.Title) HeaderText="Title" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumns>
</SfGrid>
@code{
public List<EmployeeData> Employees { get; set; }
protected override void OnInitialized()
{
Employees = Enumerable.Range(1, 9).Select(x => new EmployeeData()
{
CustomerID = x,
Data = new EmployeeName()
{
FirstName = (new string[] { "Nancy", "Andrew", "Janet", "Margaret", "Steven"
})[new Random().Next(5)],
EmployeeID = 1000 + x
},
Title = (new string[] { "Sales Representative", "Vice President, Sales",
"Sales Manager",
"Inside Sales Coordinator" })[new Random().Next(4)],
}).ToList();
}
public class EmployeeData
{
public int? CustomerID { get; set; }
public EmployeeName Data { get; set; }
public string Title { get; set; }
}
public class EmployeeName
{
public int? EmployeeID { get; set; }
public string FirstName { get; set; }
}
}

```

Paging in Blazor DataGrid Component

[Paging](#) provides an option to display DataGrid data in page segments. To enable paging, set the [AllowPaging](#) to true. When paging is enabled, pager component renders at the bottom of the datagrid.

Paging options can be configured through the [GridPageSettings](#) component.

In the below sample, [PageSize](#) is calculated based on the datagrid height by using the [OnLoad](#) event.

ASPX-CS

```
@using Syncfusion.Blazor.Grids
<SfGrid @ref="DefaultGrid" DataSource="@Orders" AllowPaging="true"
Height="200">
<GridEvents OnLoad="Load" TValue="Order"></GridEvents>
<GridPageSettings PageCount="2"></GridPageSettings>
<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText="Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight"
Visible="false" Format="C2" TextAlign="TextAlign.Right"
Width="120"></GridColumn>
</GridColumns>
</SfGrid>
@code{
private SfGrid<Order> DefaultGrid;
public int GridHeight;
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
OrderDate = DateTime.Now.AddDays(-x),
}).ToList();
}
public class Order
{
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
public void Load(object args)
{
var RowHeight = 37; //height of the each row
int.TryParse(this.DefaultGrid.Height, out GridHeight); //datagrid height
var PageSize = (this.DefaultGrid.PageSettings as GridPageSettings).PageSize;
//initial page size
decimal PageResize = ((GridHeight) - (PageSize * RowHeight)) / RowHeight;
//new page size is obtained here
#pragma warning disable BL0005
(this.DefaultGrid.PageSettings as GridPageSettings).PageSize = PageSize +
(int)Math.Round(PageResize);
}
```



```
#pragma warning restore BL0005
}
}
```

You can achieve better performance by using datagrid paging to fetch only a pre-defined number of records from the data source.

Pager with page size dropdown

The pager dropdown allows you to change the number of records in the DataGrid dynamically. It can be enabled by defining the [PageSizes](#) property of **GridPageSettings** as true.

By default, dropdown list will show values as `new int[]{ 5, 10, 12, 20 }`. You can customize the dropdown values using the **PageSizes** property itself.

ASPX-CS

```
@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders" AllowPaging="true">
<GridPageSettings PageSizes="true"></GridPageSettings>
<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText="Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumns>
</SfGrid>
@code{
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
OrderDate = DateTime.Now.AddDays(-x),
}).ToList();
}
public class Order {
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
}
```

You can refer to our [Blazor Grid Paging](#) Feature tour page to know about paging and its feature representations.

You can refer to our [Blazor DataGrid](#) feature tour page for its groundbreaking feature representations. You can also explore our [Blazor DataGrid example](#) to understand how to present and manipulate data.

Pager Template

Blazor DataGrid provides a way to customize the pager UI using the pager template feature. We can make use of the **Template** property of the [GridPageSettings](#) component to customize the pager template of DataGrid component.

Inside the **Template** RenderFragment, we can access the parameters passed to the pager templates using implicit parameter named context matching with the [PagerTemplateContent](#) class name.

In the below sample, we have customized the Pager component in DataGrid control with custom button and input element to navigate between Grid pages. We have used **GotoPageAsync** method of Grid to navigate to specific page in DataGrid control.

ASPX-CS

```
@using Syncfusion.Blazor.Grids
<SfGrid @ref="defaultGrid" DataSource="students" AllowPaging="true">
  <GridPageSettings PageSize="10">
    <Template>
      <div class="PagerTemplate">
        <div class="@($"e-first e-icons e-icon-first {ValidateFirst()} align-icons e-firstpage")" @onclick="ShowFirstPage" title="Go to first page">
        </div>
        <div class="@($"e-prev e-icons e-icon-prev {ValidateBack()} align-icons")" @onclick="ShowPreviousPage" title="Go to previous page"></div>
        <div>
          <input class="textbox add-border" type="text" @bind="pageNo" size="2" @oninput="LaunchEnteredPage" />
          <div id="totalpages" class="textbox"> of @totalPages pages </div>
        </div>
        <div class="@($"e-next e-icons e-icon-next {ValidateForward()} align-icons e-next")" @onclick="ShowNextPage" title="Go to next page"></div>
        <div class="@($"e-last e-icons e-icon-last {ValidateLast()} align-icons")" @onclick="ShowLastPage" title="Go to last page"></div>
      </div>
      <style>
        .PagerTemplate {
          width: 1000px;
          height: 64px;
          left: 183px;
          top: 615px;
          border-radius: 0px;
        }
        .textbox {
          margin-top: 9px;
          margin-bottom: 9px;
          margin-right: 2px;
          text-align: center;
        }
        .add-border {
          border: #ddd 1px solid;
        }
        .align-icons {
          margin-top: 9px;
          margin-bottom: 9px;
        }
      </style>
    </Template>
  </GridPageSettings>
</SfGrid>
```

```

margin-right: 16px;
cursor: pointer;
}
.e-firstpage {
margin-left: 6px;
}
.e-next {
margin-left: 16px;
}
.disableFirst {
pointer-events: none;
opacity: 0.3;
}
.disableLast {
pointer-events: none;
opacity: 0.3;
}
.disableFront {
pointer-events: none;
opacity: 0.3;
}
.disableBack {
pointer-events: none;
opacity: 0.3;
}
</style>
</Template>
</GridPageSettings>
<GridColumns>
<GridColumn Field="@nameof(Order.OrderID)" HeaderText="Order ID"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field="@nameof(Order.CustomerID)" HeaderText="Customer ID"
TextAlign="TextAlign.Left" Width="150"></GridColumn>
<GridColumn Field="@nameof(Order.OrderDate)" HeaderText="Order Date"
TextAlign="TextAlign.Center" Width="130" Format="d"
Type="ColumnType.Date"></GridColumn>
<GridColumn Field="@nameof(Order.Freight)" HeaderText="Freight" Format="c2"
TextAlign="TextAlign.Right" Width="130"></GridColumn>
<GridColumn Field="@nameof(Order.ShipCountry)" HeaderText="Ship Country"
TextAlign="TextAlign.Left" Width="140"></GridColumn>
</GridColumns>
</SfGrid>
@code{
public SfGrid<Order> defaultGrid;
public List<Order> students { get; set; }
public int pageNo { get; set; }
public int totalPages { get; set; }
public bool DisableBackIcon = false;
public bool DisableForwardIcon = false;
public bool DisableFirstIcon = false;
public bool DisableLastIcon = false;
protected override void OnInitialized()
{
base.OnInitialized();
students = Enumerable.Range(1, 100).Select((x) => new Order()
{
OrderID = x,

```

```
CustomerID = (new string[] { "ALFKI", "ANATR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
ShipCountry = new string[] { "Denmark", "Germany", "Austria", "Brazil",
"Switzerland" }[new Random().Next(5)],
OrderDate = new DateTime[] { new DateTime(2010, 12, 19), new DateTime(2005,
08, 20), new DateTime(1991, 01, 10), new DateTime(1992, 10, 11), new
DateTime(1999, 12, 14) }[new Random().Next(5)]
}).ToList();
}
protected override void OnAfterRender(bool firstRender)
{
base.OnAfterRender(firstRender);
if (firstRender)
{
totalPages = defaultGrid.TotalItemCount / defaultGrid.PageSettings.PageSize;
}
pageNo = defaultGrid.PageSettings.CurrentPage;
if (defaultGrid.PageSettings.CurrentPage == totalPages)
{
DisableForwardIcon = true;
DisableLastIcon = true;
}
else
{
DisableForwardIcon = false;
DisableLastIcon = false;
}
if (defaultGrid.PageSettings.CurrentPage == 1)
{
DisableBackIcon = true;
DisableFirstIcon = true;
}
else
{
DisableBackIcon = false;
DisableFirstIcon = false;
}
StateHasChanged();
}
public class Order
{
public int OrderID { get; set; }
public string CustomerID { get; set; }
public double Freight { get; set; }
public string ShipCountry { get; set; }
public DateTime OrderDate { get; set; }
}
public string ValidateFirst()
{
if (DisableFirstIcon)
{
return "disableFirst";
}
return "";
}
public string ValidateLast()
```

```
{
    if (DisableLastIcon)
    {
        return "disableLast";
    }
    return "";
}
public string ValidateForward()
{
    if (DisableForwardIcon)
    {
        return "disableFront";
    }
    return "";
}
public string ValidateBack()
{
    if (DisableBackIcon)
    {
        return "disableBack";
    }
    return "";
}
public void ShowNextPage()
{
    defaultGrid.GoToPageAsync((defaultGrid.PageSettings.CurrentPage) + 1);
}
public void ShowPreviousPage()
{
    defaultGrid.GoToPageAsync((defaultGrid.PageSettings.CurrentPage) - 1);
}
public void ShowFirstPage()
{
    defaultGrid.GoToPageAsync(1);
}
public void ShowLastPage()
{
    defaultGrid.GoToPageAsync(totalPages);
}
public void
LaunchEnteredPage(Microsoft.AspNetCore.Components.ChangeEventArgs page)
{
    if (page.Value == null || page.Value.ToString() == "")
    {
        return;
    }
    else
    {
        double enteredPage = Convert.ToDouble(page.Value);
        if (enteredPage <= totalPages)
            defaultGrid.GoToPageAsync(enteredPage);
    }
}
}
```

You can refer to our [Blazor Grid Pager Template](#) online demo of Pager Template feature in Blazor DataGrid.

Scrolling in Blazor DataGrid Component

The scrollbar will be displayed in the datagrid when content exceeds the element [Width](#) or [Height](#). The vertical and horizontal scrollbars will be displayed based on the following criteria:

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

The default value for [Height](#) and [Width](#) is **auto**.

Set width and height

To specify the [Width](#) and [Height](#) of the scroller in the pixel, set the pixel value to a number.

ASPX-CS

```
@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders" Height="315" Width="400">
  <GridColumns>
    <GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
      TextAlign="TextAlign.Right" Width="120"></GridColumn>
    <GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
      Width="150"></GridColumn>
    <GridColumn Field=@nameof(Order.OrderDate) HeaderText="Order Date"
      Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
      Width="130"></GridColumn>
    <GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
      TextAlign="TextAlign.Right" Width="120"></GridColumn>
    <GridColumn Field=@nameof(Order.ShipCountry) HeaderText="Ship Country"
      Width="150"></GridColumn>
  </GridColumns>
</SfGrid>

@code{
  public List<Order> Orders { get; set; }
  protected override void OnInitialized()
  {
    Orders = Enumerable.Range(1, 75).Select(x => new Order()
    {
      OrderID = 1000 + x,
      CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
    })[new Random().Next(5)],
      Freight = 2.1 * x,
      OrderDate = DateTime.Now.AddDays(-x),
      ShipCountry = (new string[] { "USA", "UK", "JAPAN" })[new Random().Next(3)]
    }).ToList();
  }

  public class Order {
    public int? OrderID { get; set; }
    public string CustomerID { get; set; }
    public DateTime? OrderDate { get; set; }
    public double? Freight { get; set; }
  }
}
```

```
public string ShipCountry { get; set; }
}
}
```

Responsive with parent container

Specify the [Width](#) and [Height](#) as **100%** to make the datagrid element fill its parent container.

Setting the [Height](#) to **100%** requires the datagrid parent element to have explicit height or you can use calc function to set explicit height based on the browser layout.

ASPX-CS

```
@using Syncfusion.Blazor.Grids
<div style="width:calc(100vw - 20rem); height:calc(100vh - 7rem);">
  @*Change the rem values based on your browser page layout*@
  <SfGrid DataSource="@Orders" Height="100%" Width="100%">
    <GridColumns>
      <GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
        TextAlign="TextAlign.Right" Width="120"></GridColumn>
      <GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
        Width="150"></GridColumn>
      <GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
        Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
        Width="130"></GridColumn>
      <GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
        TextAlign="TextAlign.Right" Width="120"></GridColumn>
      <GridColumn Field=@nameof(Order.ShipCountry) HeaderText="Ship Country"
        Width="150"></GridColumn>
    </GridColumns>
  </SfGrid>
</div>
@code{
  public List<Order> Orders { get; set; }
  protected override void OnInitialized()
  {
    Orders = Enumerable.Range(1, 75).Select(x => new Order()
    {
      OrderID = 1000 + x,
      CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
    })[new Random().Next(5)],
      Freight = 2.1 * x,
      OrderDate = DateTime.Now.AddDays(-x),
      ShipCountry = (new string[] { "USA", "UK", "JAPAN" })[new Random().Next(3)]
    }).ToList();
  }
  public class Order {
    public int? OrderID { get; set; }
    public string CustomerID { get; set; }
    public DateTime? OrderDate { get; set; }
    public double? Freight { get; set; }
    public string ShipCountry { get; set; }
  }
}
```

Frozen rows and columns

Frozen rows and columns provides an option to make rows and columns always visible in the top and left side of the datagrid while scrolling.

In this demo, the [FrozenColumns](#) is set as '2' and the [FrozenRows](#) is set as '3'. Hence, the left two columns and top three rows are frozen.

ASPX-CS

```
@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders" AllowSelection="false" EnableHover="false"
FrozenColumns="2" FrozenRows="3" Width="100%" Height="400" >
<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.EmployeeID) HeaderText="Employee ID"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.ShipName) HeaderText="Ship Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.ShipAddress) HeaderText="Ship Address"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.ShipCity) HeaderText="Ship City"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText="Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.ShipCountry) HeaderText="Ship Country"
Width="150"></GridColumn>
</GridColumns>
</SfGrid>
@code{
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
EmployeeID = x,
OrderDate = DateTime.Now.AddDays(-x),
ShipCountry = (new string[] { "USA", "UK", "JAPAN" })[new Random().Next(3)],
ShipName = "MOS",
ShipCity = (new string[] { "New york", "London", "Hue" })[new
Random().Next(3)],
ShipAddress = (new string[] { "55, Baker street", "65/5 Kings landing", "56,
Winterfell" })[new Random().Next(3)],
}).ToList();
}
public class Order {
public int? OrderID { get; set; }
```



```

public string CustomerID { get; set; }
public int? EmployeeID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
public string ShipCountry { get; set; }
public string ShipCity { get; set; }
public string ShipName { get; set; }
public string ShipAddress { get; set; }
}
}

```

Freeze particular columns

To freeze particular column in the datagrid, the [IsFrozen](#) property of **GridColumn** component can be used.

In this demo, the columns with field name **OrderID** and **EmployeeID** is frozen using the [IsFrozen](#) property of **GridColumn**.

ASPX-CS

```

@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders" AllowSelection="false" EnableHover="false"
FrozenRows="3" Width="100%" Height="400" >
  <GridColumn>
    <GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
IsFrozen="true" TextAlign="TextAlign.Right" Width="120"></GridColumn>
    <GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>
    <GridColumn Field=@nameof(Order.EmployeeID) HeaderText="Employee ID"
IsFrozen="true" Width="150"></GridColumn>
    <GridColumn Field=@nameof(Order.ShipName) HeaderText="Ship Name"
Width="150"></GridColumn>
    <GridColumn Field=@nameof(Order.ShipAddress) HeaderText="Ship Address"
Width="150"></GridColumn>
    <GridColumn Field=@nameof(Order.ShipCity) HeaderText="Ship City"
Width="150"></GridColumn>
    <GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
    <GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
    <GridColumn Field=@nameof(Order.ShipCountry) HeaderText="Ship Country"
Width="150"></GridColumn>
  </GridColumn>
</SfGrid>
@code{
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
EmployeeID = x,
}

```

```

OrderDate = DateTime.Now.AddDays(-x),
ShipCountry = (new string[] { "USA", "UK", "JAPAN" })[new Random().Next(3)],
ShipName = "MOS",
ShipCity = (new string[] { "New york", "London", "Hue" })[new
Random().Next(3)],
ShipAddress = (new string[] { "55, Baker street", "65/5 Kings landing", "56,
Winterfell" })[new Random().Next(3)],
}).ToList();
}
public class Order {
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public int? EmployeeID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
public string ShipCountry { get; set; }
public string ShipCity { get; set; }
public string ShipName { get; set; }
public string ShipAddress { get; set; }
}
}

```

Frozen rows and columns should not be set outside the datagrid view port.

Change default frozen line color

The following demo shows how to change the default frozen line color.

ASPX-CS

```

@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders" AllowSelection="false" EnableHover="false"
FrozenColumns="2" FrozenRows="3" Width="100%" Height="400">
<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.EmployeeID) HeaderText="Employee ID"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.ShipName) HeaderText="Ship Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.ShipAddress) HeaderText="Ship Address"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.ShipCity) HeaderText="Ship City"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.ShipCountry) HeaderText="Ship Country"
Width="150"></GridColumn>
</GridColumns>
</SfGrid>
<style>
.e-grid .e-frozenheader > .e-table,

```

```

.e-grid .e-frozencontent > .e-table,
.e-grid .e-frozencontent .e-virtualtable > .e-table,
.e-grid .e-frozenheader .e-virtualtable > .e-table {
border-right-color: orangered;
}
.e-grid .e-frozenhdrcont .e-headercontent > .e-table,
.e-grid .e-frozenhdrcont .e-frozenheader > .e-table,
.e-grid .e-frozenhdrcont .e-movableheader > .e-table,
.e-grid .e-frozenhdrcont .e-headercontent .e-virtualtable > .e-table {
border-bottom-color: orangered;
}
</style>
@code{
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
EmployeeID = x,
OrderDate = DateTime.Now.AddDays(-x),
ShipCountry = (new string[] { "USA", "UK", "JAPAN" })[new Random().Next(3)],
ShipName = "MOS",
ShipCity = (new string[] { "New york", "London", "Hue" })[new
Random().Next(3)],
ShipAddress = (new string[] { "55, Baker street", "65/5 Kings landing", "56,
Winterfell" })[new Random().Next(3)],
}).ToList();
}
public class Order
{
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public int? EmployeeID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
public string ShipCountry { get; set; }
public string ShipCity { get; set; }
public string ShipName { get; set; }
public string ShipAddress { get; set; }
}
}

```

The following screenshots represent a datagrid by customizing frozen line color.

| Order ID | Customer Name | Employee ID | Ship Name | Ship Address | Ship City | Order Date | Freight | Ship Country |
|----------|---------------|-------------|-----------|--------------------|-----------|------------|---------|--------------|
| 1001 | ANANTR | 1 | MOS | 56, Winterfell | London | 5/22/2021 | \$2.10 | USA |
| 1002 | BLONP | 2 | MOS | 65/5 Kings landing | New york | 5/21/2021 | \$4.20 | UK |
| 1003 | ANANTR | 3 | MOS | 55, Baker street | New york | 5/20/2021 | \$6.30 | UK |
| 1004 | ANANTR | 4 | MOS | 56, Winterfell | Hue | 5/19/2021 | \$8.40 | USA |
| 1005 | ANTON | 5 | MOS | 55, Baker street | London | 5/18/2021 | \$10.50 | USA |
| 1006 | ANTON | 6 | MOS | 56, Winterfell | Hue | 5/17/2021 | \$12.60 | JAPAN |
| 1007 | ANTON | 7 | MOS | 56, Winterfell | New york | 5/16/2021 | \$14.70 | UK |
| 1008 | ALFKI | 8 | MOS | 65/5 Kings landing | Hue | 5/15/2021 | \$16.80 | JAPAN |
| 1009 | ALFKI | 9 | MOS | 65/5 Kings landing | London | 5/14/2021 | \$18.90 | JAPAN |
| 1010 | ANTON | 10 | MOS | 55, Baker street | London | 5/13/2021 | \$21.00 | USA |
| 1011 | ALFKI | 11 | MOS | 65/5 Kings landing | New york | 5/12/2021 | \$23.10 | USA |

Limitations

The following features are not supported in frozen rows and columns:

- Grouping
- Row Template
- Detail Template
- Hierarchy DataGrid

Freeze Direction

You can freeze the Grid columns on the left or right side by using the [Freeze](#) property and the remaining columns will be movable. The grid will automatically move the columns to the left or right position based on the [Freeze](#) value and also [IsFrozen](#) property should be true.

Types of the [Freeze](#) directions:

- **FreezeDirection.Left:** Allows you to freeze the columns at the left.
- **FreezeDirection.Right:** Allows you to freeze the columns at the right.

In this demo, the **OrderID** column is frozen at the left and the **ShipCountry** column is frozen at the right side of the content table.

CSHARP

```
@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders" Height="364" FrozenRows="2"
EnableHover="false" AllowSelection="false" AllowSorting="true">
<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
Freeze="FreezeDirection.Left" IsFrozen="true" TextAlign="TextAlign.Right"
Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="170"></GridColumn>
<GridColumn Field=@nameof(Order.EmployeeID) HeaderText="Employee ID"
TextAlign="TextAlign.Right" Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.ShipName) HeaderText="Ship Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.ShipCity) HeaderText="Ship City"
Width="150"></GridColumn>
```

```

<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.ShipAddress) HeaderText="Ship Address"
Width="170"></GridColumn>
<GridColumn Field=@nameof(Order.Mail) HeaderText="Email"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.Location) HeaderText="Location"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.ShipCountry) HeaderText="Ship Country"
Freeze="FreezeDirection.Right" IsFrozen="true" Width="150"></GridColumn>
</GridColumns>
</SfGrid>
@code {
public List<Order> Orders { get; set; }
public SfGrid<Order> Grid { get; set; }
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
EmployeeID = x,
OrderDate = DateTime.Now.AddDays(-x),
ShipCountry = (new string[] { "USA", "UK", "JAPAN" })[new Random().Next(3)],
ShipName = "MOS",
ShipCity = (new string[] { "New york", "London", "Hue" })[new
Random().Next(3)],
ShipAddress = (new string[] { "55, Baker street", "65/5 Kings landing", "56,
Winterfell" })[new Random().Next(3)],
Mail = (new string[] { "alf32@arpy.com", "anat81@jourrapide.com",
"anton99@rpy.com", "blonp97@gmail.com", "bolid@mail.com" })[new
Random().Next(5)],
Location = (new string[] { "Morbi Leo", "Sed Blandit", "Congue Nisi",
"Aliquet Lectus", "Viverra Mauris In" })[new Random().Next(5)]
}).ToList();
}
public class Order
{
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public int? EmployeeID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
public string ShipCountry { get; set; }
public string ShipCity { get; set; }
public string ShipName { get; set; }
public string ShipAddress { get; set; }
public string Mail { get; set; }
public string Location { get; set; }
}
}

```

The following screenshots represent a frozen direction.

| Order ID | Customer Name | Employee ID | Ship | Ship Country |
|----------|---------------|-------------|------|--------------|
| 1001 | BLONP | 1 | MOS | USA |
| 1002 | BLONP | 2 | MOS | USA |
| 1003 | ANANTR | 3 | MOS | JAPAN |
| 1004 | ANANTR | 4 | MOS | USA |
| 1005 | BLONP | 5 | MOS | JAPAN |
| 1006 | BLONP | 6 | MOS | JAPAN |
| 1007 | ALFKI | 7 | MOS | USA |
| 1008 | BOLID | 8 | MOS | JAPAN |
| 1009 | BLONP | 9 | MOS | USA |
| 1010 | BOLID | 10 | MOS | UK |
| 1011 | ANANTR | 11 | MOS | USA |
| 1012 | ALFKI | 12 | MOS | UK |

Limitations of Freeze Direction

This feature has the below limitations, along with the above mentioned Frozen Grid limitations.

- Column virtualization
- Infinite scroll cache mode
- Freeze direction in the stacked header is not compatible with column reordering.
- When using cell template/text wrap in any one of the panels will result in variable row height between the panels and this height will be re-calculated based on the DOM offset height and then applied to all the rows in all the panels to make it look even. This may result in a visual glitch. You can resolve this problem by setting static values to the RowHeight property in SfGrid.
- The [Freeze](#) and [FrozenColumns](#) properties are incompatible and cannot be used at the same time.

You can refer to our [Blazor DataGrid](#) feature tour page for its groundbreaking feature representations. You can also explore our [Blazor DataGrid example](#) to understand how to present and manipulate data.

Virtualization in Blazor DataGrid Component

DataGrid allows you to load large amount of data without [performance](#) degradation.

Row Virtualization

Row virtualization allows you to load and render rows only in the content viewport. It is an alternative way of paging in which the data will be loaded while scrolling vertically. To setup the row virtualization, you need to define

[EnableVirtualization](#) as true and content height by [Height](#) property.

The number of records displayed in the DataGrid is determined implicitly by height of the content area. Also, you have an option to define a visible number of records by

the [PageSize](#) property of [GridPageSettings](#) component. The loaded data will be cached and reused when it is needed for next time.

CSHARP

```
@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@GridData" Height="600" EnableVirtualization="true">
<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
IsPrimaryKey="true" Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.EmployeeID) HeaderText="Employee ID"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.ShipCountry) HeaderText="Ship Country"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.ShipCity) HeaderText="Ship City"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.ShipAddress) HeaderText="Ship Address"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.ShippedDate) HeaderText="Shipped Date"
Width="150"></GridColumn>
</GridColumns>
</SfGrid>
@code{
public List<Order> GridData { get; set; }
protected override void OnInitialized()
{
List<Order> Order = new List<Order>();
int Code = 10000;
for (int i = 1; i < 10000; i++)
{
Order.Add(new Order(Code + 1, "ALFKI", i + 0, 2.3 * i, false, new
DateTime(1991, 05, 15), "Berlin", "Denmark", new DateTime(1996, 7, 16),
"Kirchgasse 6"));
Order.Add(new Order(Code + 2, "ANATR", i + 2, 3.3 * i, true, new
DateTime(1990, 04, 04), "Madrid", "Brazil", new DateTime(1996, 9, 11),
"Avda. Azteca 123"));
Order.Add(new Order(Code + 3, "ANTON", i + 1, 4.3 * i, true, new
DateTime(1957, 11, 30), "Cholchester", "Germany", new DateTime(1996, 10, 7),
"Carrera 52 con Ave. Bolívar #65-98 Llano Largo"));
}
```

```
Order.Add(new Order(Code + 4, "BLONP", i + 3, 5.3 * i, false, new
DateTime(1930, 10, 22), "Marseille", "Austria", new DateTime(1996, 12, 30),
"Magazinweg 7"));
Order.Add(new Order(Code + 5, "BOLID", i + 4, 6.3 * i, true, new
DateTime(1953, 02, 18), "Tsawassen", "Switzerland", new DateTime(1997, 12,
3), "1029 - 12th Ave. S.));
Code += 5;
}
GridData = Order;
}
public class Order
{
    public Order(int OrderID, string CustomerID, int EmployeeID, double Freight,
    bool Verified, DateTime OrderDate, string ShipCity, string ShipCountry,
    DateTime ShippedDate, string ShipAddress)
    {
        this.OrderID = OrderID;
        this.CustomerID = CustomerID;
        this.EmployeeID = EmployeeID;
        this.Freight = Freight;
        this.Verified = Verified;
        this.OrderDate = OrderDate;
        this.ShipCity = ShipCity;
        this.ShipCountry = ShipCountry;
        this.ShippedDate = ShippedDate;
        this.ShipAddress = ShipAddress;
    }
    public int? OrderID { get; set; }
    public string CustomerID { get; set; }
    public int? EmployeeID { get; set; }
    public double? Freight { get; set; }
    public DateTime? OrderDate { get; set; }
    public bool Verified { get; set; }
    public DateTime? ShippedDate { get; set; }
    public string ShipCountry { get; set; }
    public string ShipCity { get; set; }
    public string ShipAddress { get; set; }
}
}
```

The following GIF represent a datagrid with Row virtualization.

| Order ID | Customer Name | Employee ID | Order Date | Freight |
|----------|---------------|-------------|------------|---------|
| 10001 | ALFKI | 1 | 5/15/1991 | \$2.30 |
| 10002 | ANATR | 3 | 4/4/1990 | \$3.30 |
| 10003 | ANTON | 2 | 11/30/1957 | \$4.30 |
| 10004 | BLONP | 4 | 10/22/1930 | \$5.30 |
| 10005 | BOLID | 5 | 2/18/1953 | \$6.30 |
| 10006 | ALFKI | 2 | 5/15/1991 | \$4.60 |
| 10007 | ANATR | 4 | 4/4/1990 | \$6.60 |
| 10008 | ANTON | 3 | 11/30/1957 | \$8.60 |
| 10009 | BLONP | 5 | 10/22/1930 | \$10.60 |
| 10010 | BOLID | 6 | 2/18/1953 | \$12.60 |
| 10011 | ALFKI | 3 | 5/15/1991 | \$6.90 |
| 10012 | ANATR | 5 | 4/4/1990 | \$9.90 |
| 10013 | ANTON | 4 | 11/30/1957 | \$12.90 |
| 10014 | BLONP | 6 | 10/22/1930 | \$15.90 |

Column Virtualization

Column virtualization allows you to virtualize columns. It will render columns which are in the viewport. You can scroll horizontally to view more columns.

To setup the column virtualization, set the

[EnableVirtualization](#) and

[EnableColumnVirtualization](#) properties as **true**.

CSHARP

```
@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@GridData" Height="600" Width="300"
EnableVirtualization="true" EnableColumnVirtualization="true">
<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
IsPrimaryKey="true" Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.EmployeeID) HeaderText="Employee ID"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
```

```

<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.ShipCountry) HeaderText="Ship Country"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.ShipCity) HeaderText="Ship City"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.ShipAddress) HeaderText="Ship Address"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.ShippedDate) HeaderText="Shipped Date"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.Verified) HeaderText="Ship Name"
Type="ColumnType.Boolean" Width="150"></GridColumn>
</GridColumn>
</SfGrid>
@code{
public List<Order> GridData { get; set; }
protected override void OnInitialized()
{
List<Order> Order = new List<Order>();
int Code = 10000;
for (int i = 1; i < 10000; i++)
{
Order.Add(new Order(Code + 1, "ALFKI", i + 0, 2.3 * i, false, new
DateTime(1991, 05, 15), "Berlin", "Denmark", new DateTime(1996, 7, 16),
"Kirchgasse 6"));
Order.Add(new Order(Code + 2, "ANATR", i + 2, 3.3 * i, true, new
DateTime(1990, 04, 04), "Madrid", "Brazil", new DateTime(1996, 9, 11),
"Avda. Azteca 123"));
Order.Add(new Order(Code + 3, "ANTON", i + 1, 4.3 * i, true, new
DateTime(1957, 11, 30), "Cholchester", "Germany", new DateTime(1996, 10, 7),
"Carrera 52 con Ave. Bolívar #65-98 Llano Largo"));
Order.Add(new Order(Code + 4, "BLONP", i + 3, 5.3 * i, false, new
DateTime(1930, 10, 22), "Marseille", "Austria", new DateTime(1996, 12, 30),
"Magazinweg 7"));
Order.Add(new Order(Code + 5, "BOLID", i + 4, 6.3 * i, true, new
DateTime(1953, 02, 18), "Tsawassen", "Switzerland", new DateTime(1997, 12,
3), "1029 - 12th Ave. S.));
Code += 5;
}
GridData = Order;
}
public class Order
{
public Order(int OrderID, string CustomerID, int EmployeeID, double Freight,
bool Verified, DateTime OrderDate, string ShipCity, string ShipCountry,
DateTime ShippedDate, string ShipAddress)
{
this.OrderID = OrderID;
this.CustomerID = CustomerID;
this.EmployeeID = EmployeeID;
this.Freight = Freight;
this.Verified = Verified;
this.OrderDate = OrderDate;
this.ShipCity = ShipCity;
this.ShipCountry = ShipCountry;
this.ShippedDate = ShippedDate;
this.ShipAddress = ShipAddress;
}
}
}

```

```

}
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public int? EmployeeID { get; set; }
public double? Freight { get; set; }
public DateTime? OrderDate { get; set; }
public bool Verified { get; set; }
public DateTime? ShippedDate { get; set; }
public string ShipCountry { get; set; }
public string ShipCity { get; set; }
public string ShipAddress { get; set; }
}
}

```

Column's [Width](#) is required for column virtualization. If column's width is not defined then DataGrid will consider its value as **200px**.

The following GIF represent a datagrid with Column virtualization.

| Customer Name | Customer Name | Employee ID | Order Date | Freight |
|---------------|---------------|-------------|------------|---------|
| 10001 | ALFKI | 1 | 5/15/1991 | \$2.30 |
| 10002 | ANATR | 3 | 4/4/1990 | \$3.30 |
| 10003 | ANTON | 2 | 11/30/1957 | \$4.30 |
| 10004 | BLONP | 4 | 10/22/1930 | \$5.30 |
| 10005 | BOLID | 5 | 2/18/1953 | \$6.30 |
| 10006 | ALFKI | 2 | 5/15/1991 | \$4.60 |
| 10007 | ANATR | 4 | 4/4/1990 | \$6.60 |
| 10008 | ANTON | 3 | 11/30/1957 | \$8.60 |
| 10009 | BLONP | 5 | 10/22/1930 | \$10.60 |
| 10010 | BOLID | 6 | 2/18/1953 | \$12.60 |
| 10011 | ALFKI | 3 | 5/15/1991 | \$6.90 |
| 10012 | ANATR | 5 | 4/4/1990 | \$9.90 |
| 10013 | ANTON | 4 | 11/30/1957 | \$12.90 |
| 10014 | BLONP | 6 | 10/22/1930 | \$15.90 |

The collapsed/expanded state will persist only for local dataSource while scrolling.

Enable Cell placeholder during Virtualization

This enable cell placeholder during virtualization feature much of a muchness of row virtualization and column virtualization feature and the difference is loading placeholder indicator was shown on the cells while loading the new data. Also same set of DOM elements is reused to improve performance.

To setup the enable cell placeholder during virtualization, you need to define [EnableVirtualMaskRow](#) as true along with [EnableVirtualization/EnableColumnVirtualization](#) property.

CSHARP

```
@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@GridData" Height="600" Width="300" RowHeight="38"
EnableVirtualMaskRow="true" EnableVirtualization="true"
EnableColumnVirtualization="true">
<GridPageSettings PageSize="32"></GridPageSettings>
<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
IsPrimaryKey="true" Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.EmployeeID) HeaderText="Employee ID"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText="Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.ShipCountry) HeaderText="Ship Country"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.ShipCity) HeaderText="Ship City"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.ShipAddress) HeaderText="Ship Address"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.ShippedDate) HeaderText="Shipped Date"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.Verified) HeaderText="Ship Name"
Type="ColumnType.Boolean" Width="150"></GridColumn>
</GridColumns>
</SfGrid>
@code{
public List<Order> GridData { get; set; }
protected override void OnInitialized()
{
List<Order> Order = new List<Order>();
int Code = 10000;
for (int i = 1; i < 10000; i++)
{
Order.Add(new Order(Code + 1, "ALFKI", i + 0, 2.3 * i, false, new
DateTime(1991, 05, 15), "Berlin", "Denmark", new DateTime(1996, 7, 16),
"Kirchgasse 6"));
Order.Add(new Order(Code + 2, "ANATR", i + 2, 3.3 * i, true, new
DateTime(1990, 04, 04), "Madrid", "Brazil", new DateTime(1996, 9, 11),
"Avda. Azteca 123"));
Order.Add(new Order(Code + 3, "ANTON", i + 1, 4.3 * i, true, new
DateTime(1957, 11, 30), "Cholchester", "Germany", new DateTime(1996, 10, 7),
"Carrera 52 con Ave. Bolívar #65-98 Llano Largo"));
}
```

```
Order.Add(new Order(Code + 4, "BLONP", i + 3, 5.3 * i, false, new
DateTime(1930, 10, 22), "Marseille", "Austria", new DateTime(1996, 12, 30),
"Magazinweg 7"));
Order.Add(new Order(Code + 5, "BOLID", i + 4, 6.3 * i, true, new
DateTime(1953, 02, 18), "Tsawassen", "Switzerland", new DateTime(1997, 12,
3), "1029 - 12th Ave. S.));
Code += 5;
}
GridData = Order;
}
public class Order
{
    public Order(int OrderID, string CustomerID, int EmployeeID, double Freight,
    bool Verified, DateTime OrderDate, string ShipCity, string ShipCountry,
    DateTime ShippedDate, string ShipAddress)
    {
        this.OrderID = OrderID;
        this.CustomerID = CustomerID;
        this.EmployeeID = EmployeeID;
        this.Freight = Freight;
        this.Verified = Verified;
        this.OrderDate = OrderDate;
        this.ShipCity = ShipCity;
        this.ShipCountry = ShipCountry;
        this.ShippedDate = ShippedDate;
        this.ShipAddress = ShipAddress;
    }
    public int? OrderID { get; set; }
    public string CustomerID { get; set; }
    public int? EmployeeID { get; set; }
    public double? Freight { get; set; }
    public DateTime? OrderDate { get; set; }
    public bool Verified { get; set; }
    public DateTime? ShippedDate { get; set; }
    public string ShipCountry { get; set; }
    public string ShipCity { get; set; }
    public string ShipAddress { get; set; }
}
}
```

The following GIF represent a datagrid with Mask row virtualization.

| Order ID | Customer Name | Employee ID |
|----------|---------------|-------------|
| 10001 | ALFKI | 1 |
| 10002 | ANATR | 3 |
| 10003 | ANTON | 2 |
| 10004 | BLONP | 4 |
| 10005 | BOLID | 5 |
| 10006 | ALFKI | 2 |
| 10007 | ANATR | 4 |
| 10008 | ANTON | 3 |
| 10009 | BLONP | 5 |
| 10010 | BOLID | 6 |

Frozen Columns Virtualization

This feature virtualize the row and movable column data. Column virtualization allows you to virtualize the movable columns and cell placeholder renders before new columns loading the viewport.

Row virtualization allows you to virtualize the vertical data with cell placeholder. This placeholder renders before new row data loading in the viewport.

To setup the frozen right/left columns, you need to define Column property of **Freeze** as Right/Left along with enabling the column property of [IsFrozen](#).

ASPX-CS

```
@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@GridData" Height="400" Width="500" EnableHover="false"
RowHeight="38" EnableVirtualization="true" EnableColumnVirtualization="true"
EnableVirtualMaskRow="true">
  <GridPageSettings PageSize="40"></GridPageSettings>
  <GridColumns>
    <GridColumn Field="Field1" HeaderText="Player Name" IsFrozen="true"
Freeze="FreezeDirection.Left" Width="120"></GridColumn>
```

```

<GridColumn Field="Field2" HeaderText="Year" TextAlign="TextAlign.Right"
Width="150"></GridColumn>
<GridColumn Field="Field3" HeaderText="Stint" TextAlign="TextAlign.Right"
Width="150"></GridColumn>
<GridColumn Field="Field4" HeaderText="TMID" TextAlign="TextAlign.Right"
Width="150"></GridColumn>
<GridColumn Field="Field5" HeaderText="LGID" TextAlign="TextAlign.Right"
Width="150"></GridColumn>
<GridColumn Field="Field6" HeaderText="Game Paused"
TextAlign="TextAlign.Right" Width="150"></GridColumn>
<GridColumn Field="Field7" HeaderText="Game Started"
TextAlign="TextAlign.Right" Width="150"></GridColumn>
<GridColumn Field="Field8" HeaderText="Minutes" TextAlign="TextAlign.Right"
Width="150"></GridColumn>
<GridColumn Field="Field9" HeaderText="Points" IsFrozen="true"
Freeze="FreezeDirection.Right" TextAlign="TextAlign.Right"
Width="150"></GridColumn>
<GridColumn Field="Field10" HeaderText="Offensive Rebounds"
TextAlign="TextAlign.Right" Width="150"></GridColumn>
<GridColumn Field="Field11" HeaderText="Defensive Rebounds"
TextAlign="TextAlign.Right" Width="150"></GridColumn>
<GridColumn Field="Field12" HeaderText="Rebounds"
TextAlign="TextAlign.Right" Width="150"></GridColumn>
<GridColumn Field="Field13" HeaderText="Assists" TextAlign="TextAlign.Right"
Width="150"></GridColumn>
<GridColumn Field="Field14" HeaderText="Steals" TextAlign="TextAlign.Right"
Width="150"></GridColumn>
<GridColumn Field="Field15" HeaderText="Blocks" TextAlign="TextAlign.Right"
Width="150"></GridColumn>
<GridColumn Field="Field16" HeaderText="TurnOvers"
TextAlign="TextAlign.Right" Width="150"></GridColumn>
<GridColumn Field="Field17" HeaderText="Power Forward"
TextAlign="TextAlign.Right" Width="150"></GridColumn>
<GridColumn Field="Field18" HeaderText="fgAttempted"
TextAlign="TextAlign.Right" Width="150"></GridColumn>
<GridColumn Field="Field19" HeaderText="fgMade" TextAlign="TextAlign.Right"
Width="150"></GridColumn>
<GridColumn Field="Field20" HeaderText="ftAttempted"
TextAlign="TextAlign.Right" Width="150"></GridColumn>
</GridColumns>
</SfGrid>
@code{
public List<VirtualData> GridData { get; set; } = new List<VirtualData>();
protected override void OnInitialized()
{
List<VirtualData> data = new List<VirtualData>();
Random Random = new Random();
string[] Name = new string[] { "hardire", "abramjo01", "aubucch01", "Hook",
"Rumpelstiltskin", "Belle", "Emma", "Regina", "Aurora", "Elsa",
"Anna", "Snow White", "Prince Charming", "Cora", "Zelena", "August",
"Mulan", "Graham", "Discord", "Will", "Robin Hood",
"Jiminy Cricket", "Henry", "Neal", "Red", "Aaran", "Aaren", "Aarez",
"Aarman", "Aaron", "Aaron-James", "Aarron", "Aaryan", "Aaryn",
"Aayan", "Aazaan", "Abaan", "Abbas", "Abdallah", "Abdalroof", "Abdihakim",
"Abdirahman", "Abdisalam", "Abdul", "Abdul-Aziz",
"Abdulbasir", "Abdulkadir", "Abdulkarem", "Abdulkhader", "Abdullah", "Abdul-
Majeed", "Abdulmalik", "Abdul-Rehman", "Abdur",

```

```

"Abdurraheem", "Abdur-Rahman", "Abdur-Rehmaan", "Abel", "Abhinav",
"Abhisumant", "Abid", "Abir", "Abraham", "Abu", "Abubakar",
"Ace", "Adain", "Adam", "Adam-James", "Addison", "Addisson", "Adegbola",
"Adegbolahan", "Aden", "Adenn", "Adie", "Adil", "Aditya",
"Adnan", "Adrian", "Adrien", "Aedan", "Aedin", "Aedyn", "Aeron", "Afonso",
"Ahmad", "Ahmed", "Ahmed-Aziz", "Ahoua", "Ahtasham",
"Aiadan", "Aidan", "Aiden", "Aiden-Jack", "Aiden-Vee" };
GridData = Enumerable.Range(1, 1000).Select(x => new VirtualData()
{
    Field1 = Name[Random.Next(96)],
    Field2 = 1967 + (x % 10),
    Field3 = (int)Math.Floor(Random.NextDouble() * 200),
    Field4 = (int)Math.Floor(Random.NextDouble() * 100),
    Field5 = (int)Math.Floor(Random.NextDouble() * 2000),
    Field6 = (int)Math.Floor(Random.NextDouble() * 1000),
    Field7 = (int)Math.Floor(Random.NextDouble() * 100),
    Field8 = (int)Math.Floor(Random.NextDouble() * 10),
    Field9 = (int)Math.Floor(Random.NextDouble() * 10),
    Field10 = (int)Math.Floor(Random.NextDouble() * 100),
    Field11 = (int)Math.Floor(Random.NextDouble() * 100),
    Field12 = (int)Math.Floor(Random.NextDouble() * 1000),
    Field13 = (int)Math.Floor(Random.NextDouble() * 10),
    Field14 = (int)Math.Floor(Random.NextDouble() * 10),
    Field15 = (int)Math.Floor(Random.NextDouble() * 1000),
    Field16 = (int)Math.Floor(Random.NextDouble() * 200),
    Field17 = (int)Math.Floor(Random.NextDouble() * 300),
    Field18 = (int)Math.Floor(Random.NextDouble() * 400),
    Field19 = (int)Math.Floor(Random.NextDouble() * 500),
    Field20 = (int)Math.Floor(Random.NextDouble() * 700)
}).ToList();
}
public class VirtualData
{
    public string Field1 { get; set; }
    public int Field2 { get; set; }
    public int Field3 { get; set; }
    public int Field4 { get; set; }
    public int Field5 { get; set; }
    public int Field6 { get; set; }
    public int Field7 { get; set; }
    public int Field8 { get; set; }
    public int Field9 { get; set; }
    public int Field10 { get; set; }
    public int Field11 { get; set; }
    public int Field12 { get; set; }
    public int Field13 { get; set; }
    public int Field14 { get; set; }
    public int Field15 { get; set; }
    public int Field16 { get; set; }
    public int Field17 { get; set; }
    public int Field18 { get; set; }
    public int Field19 { get; set; }
    public int Field20 { get; set; }
}
}

```


The following GIF represent a datagrid with Frozen columns/row virtualization.

| PlayerName | Year | Points |
|-------------|------|--------|
| Aaryn | 1967 | 3 |
| Adain | 1968 | 6 |
| Mulan | 1969 | 8 |
| Abdurraheem | 1970 | 6 |
| Ahmad | 1971 | 7 |
| Ace | 1972 | 7 |
| Adegbolahan | 1973 | 4 |
| Aaren | 1974 | 6 |
| Ahmed | 1975 | 0 |
| Abdihakim | 1976 | 6 |

[Scroll the content by external button](#)

This section shows you how to invoke a [ScrollIntoViewAsync](#) method to scroll the grid content into view externally by passing column index or row index as parameter.

To scroll the grid content in horizontal direction set the [EnableVirtualization](#) and [EnableColumnVirtualization](#) properties as **true**.

To scroll the grid content in vertical direction set [EnableVirtualization](#) property as **true**.

C#

```
@using Syncfusion.Blazor.Grids
@using Syncfusion.Blazor.Buttons
ColumnIndex : <input @bind-value = "@ColumnIndex" />
<SfButton @onclick="Scroll" Content="Scroll Horizontally"></SfButton>
RowIndex : <input @bind-value = "@RowIndex" />
<SfButton @onclick="Scroll" Content="Scroll Vertically"></SfButton>
```

```

<SfGrid DataSource="@GridData" @ref="Grid" Height="500" RowHeight="35"
Width="600" EnableVirtualization="true" EnableColumnVirtualization="true">
<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.EmployeeID) HeaderText="Employee ID"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.ShipCountry) HeaderText="Ship Country"
Width="160" TextAlign="TextAlign.Right"></GridColumn>
<GridColumn Field=@nameof(Order.ShipCity) HeaderText="Ship City"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.ShipAddress) HeaderText="Ship Address"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.ShippedDate) HeaderText="Shipped Date"
TextAlign="TextAlign.Right" Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.Verified) HeaderText="Verified"
Width="200"></GridColumn>
</GridColumns>
</SfGrid>
@code{
public List<Order> GridData { get; set; }
SfGrid<Order> Grid { get; set; }
public int ColumnIndex { get; set; } = -1;
public int RowIndex { get; set; } = -1;
public int RowHeight { get; set; } = -1;
protected override void OnInitialized()
{
List<Order> Order = new List<Order>();
int Code = 10000;
for (int i = 1; i < 10000; i++)
{
Order.Add(new Order(Code + 1, "ALFKI", i + 0, 2.3 * i, false, new
DateTime(1991, 05, 15), "Berlin", "Denmark", new DateTime(1996, 7, 16),
"Kirchgasse 6"));
Order.Add(new Order(Code + 2, "ANATR", i + 2, 3.3 * i, true, new
DateTime(1990, 04, 04), "Madrid", "Brazil", new DateTime(1996, 9, 11),
"Avda. Azteca 123"));
Order.Add(new Order(Code + 3, "ANTON", i + 1, 4.3 * i, true, new
DateTime(1957, 11, 30), "Cholchester", "Germany", new DateTime(1996, 10, 7),
"Carrera 52 con Ave. Bolívar #65-98 Llano Largo"));
Order.Add(new Order(Code + 4, "BLONP", i + 3, 5.3 * i, false, new
DateTime(1930, 10, 22), "Marseille", "Austria", new DateTime(1996, 12, 30),
"Magazinweg 7"));
Order.Add(new Order(Code + 5, "BOLID", i + 4, 6.3 * i, true, new
DateTime(1953, 02, 18), "Tsawassen", "Switzerland", new DateTime(1997, 12,
3), "1029 - 12th Ave. S.));
Code += 5;
}
GridData = Order;
}

```

```
public async Task Scroll()
{
    await Grid.ScrollIntoViewAsync(ColumnIndex, RowIndex, RowHeight);
}

public class Order
{
    public Order(int OrderID, string CustomerID, int EmployeeID, double Freight,
        bool Verified, DateTime OrderDate, string ShipCity, string ShipCountry,
        DateTime ShippedDate, string ShipAddress)
    {
        this.OrderID = OrderID;
        this.CustomerID = CustomerID;
        this.EmployeeID = EmployeeID;
        this.Freight = Freight;
        this.Verified = Verified;
        this.OrderDate = OrderDate;
        this.ShipCity = ShipCity;
        this.ShipCountry = ShipCountry;
        this.ShippedDate = ShippedDate;
        this.ShipAddress = ShipAddress;
    }

    public int? OrderID { get; set; }
    public string CustomerID { get; set; }
    public int? EmployeeID { get; set; }
    public double? Freight { get; set; }
    public DateTime? OrderDate { get; set; }
    public bool Verified { get; set; }
    public DateTime? ShippedDate { get; set; }
    public string ShipCountry { get; set; }
    public string ShipCity { get; set; }
    public string ShipAddress { get; set; }
}
}
```

ColumnIndex :

Scroll Horizontally

RowIndex :

Scroll Vertically

| Order ID | Customer Name | Employee ID | Order Date |
|----------|---------------|-------------|------------|
| 10001 | ALFKI | 1 | 5/15/1991 |
| 10002 | ANATR | 3 | 4/4/1990 |
| 10003 | ANTON | 2 | 11/30/1957 |
| 10004 | BLONP | 4 | 10/22/1930 |
| 10005 | BOLID | 5 | 2/18/1953 |
| 10006 | ALFKI | 2 | 5/15/1991 |
| 10007 | ANATR | 4 | 4/4/1990 |
| 10008 | ANTON | 3 | 11/30/1957 |
| 10009 | BLONP | 5 | 10/22/1930 |
| 10010 | BOLID | 6 | 2/18/1953 |
| 10011 | ALFKI | 3 | 5/15/1991 |

Limitations for Virtualization

- While using column virtualization, column width should be in the pixel. Percentage values are not accepted.
- Due to the element height limitation in browsers, the maximum number of records loaded by the datagrid is limited by the browser capability.
- Cell selection will not be persisted in both row and column virtualization.
- Virtual scrolling is not compatible with detail template, and hierarchy features
- Group expand and collapse state will not be persisted.
- Since data is virtualized in datagrid, the aggregated information and total group items are displayed based on the current view items.
- The page size provided must be two times larger than the number of visible rows in the datagrid. If the page size is failed to meet this condition then the size will be determined by datagrid.
- The height of the datagrid content is calculated using the row height and total number of records in the data source and hence features which changes row height such as text wrapping

are not supported. If you want to increase the row height to accommodate the content then you can specify the row height using **RowHeight** property to ensure all the table rows are in same height.

- Programmatic selection using the **SelectRows** method is not supported in virtual scrolling.

See Also

- [Row virtualization with Lazy load grouping in DataGrid](#)

You can refer to our [Blazor DataGrid](#) feature tour page for its groundbreaking feature representations. You can also explore our [Blazor DataGrid example](#) to understand how to present and manipulate data.

Selection in Blazor DataGrid Component

Selection provides an option to highlight a row or a cell. It can be done through simple mouse down or arrow keys. To disable selection in the DataGrid, set the [AllowSelection](#) property to false.

The datagrid supports two types of selection that can be set by using the [Type](#) property of **GridSelectionSettings** component. They are:

- **Single:** The **Single** value is set by default, and it only allows selection of a single row or a cell.
- **Multiple:** Allows you to select multiple rows or cells.

To perform the multi-selection, press and hold CTRL key and click the desired rows or cells. To select range of rows or cells, press and hold the SHIFT key and click the rows or cells.

ASPX-CS

```
@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders" AllowSelection="true" AllowPaging="true">
  <GridSelectionSettings
    Type="SelectionType.Multiple"></GridSelectionSettings>
  <GridColumn>
    <GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
      TextAlign="TextAlign.Right" Width="120"></GridColumn>
    <GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
      Width="150"></GridColumn>
    <GridColumn Field=@nameof(Order.OrderDate) HeaderText="Order Date"
      Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
      Width="130"></GridColumn>
    <GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
      TextAlign="TextAlign.Right" Width="120"></GridColumn>
  </GridColumn>
</SfGrid>

@code{
  public List<Order> Orders { get; set; }
  protected override void OnInitialized()
  {
    Orders = Enumerable.Range(1, 75).Select(x => new Order()
    {
      OrderID = 1000 + x,
      CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
    })[new Random().Next(5)],
      Freight = 2.1 * x,
      OrderDate = DateTime.Now.AddDays(-x),
```

```

}).ToList();
}
public class Order {
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
}

```

The following represents the multiple selected rows

| Order ID | Customer Name | Order Date | Freight |
|----------|---------------|------------|---------|
| 1001 | BOLID | 7/23/2019 | \$2.10 |
| 1002 | BOLID | 7/22/2019 | \$4.20 |
| 1003 | ALFKI | 7/21/2019 | \$6.30 |
| 1004 | BOLID | 7/20/2019 | \$8.40 |
| 1005 | ANANTR | 7/19/2019 | \$10.50 |
| 1006 | BLONP | 7/18/2019 | \$12.60 |
| 1007 | ALFKI | 7/17/2019 | \$14.70 |
| 1008 | ANTON | 7/16/2019 | \$16.80 |
| 1009 | ANANTR | 7/15/2019 | \$18.90 |
| 1010 | ANTON | 7/14/2019 | \$21.00 |
| 1011 | ANTON | 7/13/2019 | \$23.10 |
| 1012 | ANANTR | 7/12/2019 | \$25.20 |

K < 1 2 3 4 5 6 7 > X
1 of 7 pages (75 items)

Selection mode

The datagrid supports three types of selection mode that can be set by using the [Mode](#) property of **GridSelectionSettings** component. They are:

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

ASPX-CS

```

@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders" AllowSelection="true" AllowPaging="true">
<GridSelectionSettings Mode="SelectionMode.Both"
Type="SelectionType.Multiple"></GridSelectionSettings>
<GridColumns>

```

```

<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumns>
</SfGrid>
@code{
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
OrderDate = DateTime.Now.AddDays(-x),
}).ToList();
}
public class Order {
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
}

```

The following screenshot shows selection mode using both(row and cells selection)

| Order ID | Customer Name | Order Date | Freight |
|----------|---------------|------------|---------|
| 1001 | ANTON | 7/23/2019 | \$2.10 |
| 1002 | BOLID | 7/22/2019 | \$4.20 |
| 1003 | BOLID | 7/21/2019 | \$6.30 |
| 1004 | BLONP | 7/20/2019 | \$8.40 |
| 1005 | BLONP | 7/19/2019 | \$10.50 |
| 1006 | BOLID | 7/18/2019 | \$12.60 |
| 1007 | ALFKI | 7/17/2019 | \$14.70 |
| 1008 | ANTON | 7/16/2019 | \$16.80 |
| 1009 | ANANTR | 7/15/2019 | \$18.90 |
| 1010 | BOLID | 7/14/2019 | \$21.00 |
| 1011 | ANTON | 7/13/2019 | \$23.10 |
| 1012 | ALFKI | 7/12/2019 | \$25.20 |

⏪
<
1
2
3
4
5
6
7
>
⏩

1 of 7 pages (75 items)

Cell selection

Cell selection can be done through simple mouse down or arrow keys (up, down, left, and right).

The datagrid supports two types of cell selection mode that can be set by using

the [CellSelectionMode](#) property of **GridSelectionSettings** component. They are:

- **Flow:** The **Flow** value is set by default. The range of cells are selected between the start index and end index that includes in between cells of rows.
- **Box:** Range of cells are selected from the start and end column indexes that includes in between cells of rows within the range.

ASPX-CS

```
@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders" AllowSelection="true" AllowPaging="true">
<GridSelectionSettings CellSelectionMode="CellSelectionMode.Box"
Mode="SelectionMode.Cell"
Type="SelectionType.Multiple"></GridSelectionSettings>
<GridColumn>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText="Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumn>
</SfGrid>
@code{
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
OrderDate = DateTime.Now.AddDays(-x),
}).ToList();
}
public class Order {
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
}
```

The following screenshot shows selection mode using cell

| Order ID | Customer Name | Order Date | Freight |
|----------|---------------|------------|---------|
| 1001 | BOLID | 7/23/2019 | \$2.10 |
| 1002 | ALFKI | 7/22/2019 | \$4.20 |
| 1003 | BOLID | 7/21/2019 | \$6.30 |
| 1004 | BLONP | 7/20/2019 | \$8.40 |
| 1005 | ANANTR | 7/19/2019 | \$10.50 |
| 1006 | ANANTR | 7/18/2019 | \$12.60 |
| 1007 | BOLID | 7/17/2019 | \$14.70 |
| 1008 | BOLID | 7/16/2019 | \$16.80 |
| 1009 | ANANTR | 7/15/2019 | \$18.90 |
| 1010 | ALFKI | 7/14/2019 | \$21.00 |
| 1011 | ANANTR | 7/13/2019 | \$23.10 |
| 1012 | ALFKI | 7/12/2019 | \$25.20 |

K < 1 2 3 4 5 6 7 > X 1 of 7 pages (75 items)

Cell selection requires the [Mode](#) to be **Cell** or **Both**, and

[Type](#) should be **Multiple**.

Checkbox selection

Checkbox selection provides an option to select multiple datagrid records with help of checkbox in each row.

To render the checkbox in each datagrid row, you need to assign the type as **CheckBox** using the column [Type](#) property of **GridColumn** component.

ASPX-CS

```
@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders" AllowSelection="true" AllowPaging="true">
  <GridSelectionSettings
    Type="SelectionMode.Multiple"></GridSelectionSettings>
  <GridColumn>
    <GridColumn Type="ColumnType.CheckBox" Width="50"></GridColumn>
    <GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
      TextAlign="TextAlign.Right" Width="120"></GridColumn>
    <GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
      Width="150"></GridColumn>
    <GridColumn Field=@nameof(Order.OrderDate) HeaderText="Order Date"
      Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
      Width="130"></GridColumn>
    <GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
      TextAlign="TextAlign.Right" Width="120"></GridColumn>
  </GridColumn>
</SfGrid>

@code{
  public List<Order> Orders { get; set; }
  protected override void OnInitialized()
  {
```

```

Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
    OrderID = 1000 + x,
    CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
    })[new Random().Next(5)],
    Freight = 2.1 * x,
    OrderDate = DateTime.Now.AddDays(-x),
}).ToList();
}
public class Order {
    public int? OrderID { get; set; }
    public string CustomerID { get; set; }
    public DateTime? OrderDate { get; set; }
    public double? Freight { get; set; }
}
}

```

The following screenshot shows CheckBox Selection

| <input checked="" type="checkbox"/> | Order ID | Customer Name | Order Date | Freight |
|-------------------------------------|----------|---------------|------------|---------|
| <input checked="" type="checkbox"/> | 1001 | ALFKI | 7/23/2019 | \$2.10 |
| <input checked="" type="checkbox"/> | 1002 | ANANTR | 7/22/2019 | \$4.20 |
| <input checked="" type="checkbox"/> | 1003 | BLONP | 7/21/2019 | \$6.30 |
| <input checked="" type="checkbox"/> | 1004 | BOLID | 7/20/2019 | \$8.40 |
| <input checked="" type="checkbox"/> | 1005 | BOLID | 7/19/2019 | \$10.50 |
| <input checked="" type="checkbox"/> | 1006 | BLONP | 7/18/2019 | \$12.60 |
| <input checked="" type="checkbox"/> | 1007 | BLONP | 7/17/2019 | \$14.70 |
| <input checked="" type="checkbox"/> | 1008 | ANANTR | 7/16/2019 | \$16.80 |
| <input checked="" type="checkbox"/> | 1009 | BOLID | 7/15/2019 | \$18.90 |
| <input checked="" type="checkbox"/> | 1010 | ALFKI | 7/14/2019 | \$21.00 |
| <input checked="" type="checkbox"/> | 1011 | ANTON | 7/13/2019 | \$23.10 |
| <input checked="" type="checkbox"/> | 1012 | ANANTR | 7/12/2019 | \$25.20 |

K < 1 2 3 4 5 6 7 > X
1 of 7 pages (75 items)

By default, selection is allowed by clicking a datagrid row or checkbox in that row. To allow selection only through checkbox, you can set the

[CheckboxOnly](#) property of **GridSelectionSettings** as true.

Selection can be persisted in all the operations using the [PersistSelection](#) property of **GridSelectionSettings**.

For persisting selection on the datagrid, any one of the columns should be defined as a primary key using the [IsPrimaryKey](#) property.

Checkbox selection mode

In checkbox selection, selection can also be done by clicking on rows. This selection provides two types of Checkbox Selection mode which can be set by using the following API,

[CheckboxMode](#). The modes are;

- **Default:** This is the default value of the checkboxMode. In this mode, user can select multiple rows by clicking rows one by one.
- **ResetOnRowClick:** In ResetOnRowClick mode, when user clicks on a row it will reset previously selected row. Also you can perform multiple-selection in this mode by press

and hold CTRL key and click the desired rows. To select range of rows, press and hold the SHIFT key and click the rows.

ASPX-CS

```
@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders" AllowSelection="true" AllowPaging="true">
  <GridSelectionSettings CheckboxMode="CheckboxSelectionType.ResetOnRowClick"
    Type="SelectionType.Multiple"></GridSelectionSettings>
  <GridColumns>
    <GridColumn Type="ColumnType.CheckBox" Width="50"></GridColumn>
    <GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
      TextAlign="TextAlign.Right" Width="120"></GridColumn>
    <GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
      Width="150"></GridColumn>
    <GridColumn Field=@nameof(Order.OrderDate) HeaderText="Order Date"
      Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
      Width="130"></GridColumn>
    <GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
      TextAlign="TextAlign.Right" Width="120"></GridColumn>
  </GridColumns>
</SfGrid>
@code{
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
  Orders = Enumerable.Range(1, 75).Select(x => new Order()
  {
    OrderID = 1000 + x,
    CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
  })[new Random().Next(5)],
    Freight = 2.1 * x,
    OrderDate = DateTime.Now.AddDays(-x),
  }).ToList();
}
public class Order {
  public int? OrderID { get; set; }
  public string CustomerID { get; set; }
  public DateTime? OrderDate { get; set; }
  public double? Freight { get; set; }
}
```

Toggle selection

The Toggle selection allows to perform selection and unselection of the particular row or cell. To enable toggle selection, set [EnableToggle](#) property of **GridSelectionSettings** as true. If you click on the selected row or cell then it will be unselected and vice versa.

ASPX-CS

```
@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders" AllowSelection="true" AllowPaging="true">
  <GridSelectionSettings EnableToggle="true"
  Type="SelectionType.Multiple"></GridSelectionSettings>
  <GridColumns>
    <GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
    TextAlign="TextAlign.Right" Width="120"></GridColumn>
    <GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
    Width="150"></GridColumn>
    <GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
    Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
    Width="130"></GridColumn>
    <GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
    TextAlign="TextAlign.Right" Width="120"></GridColumn>
  </GridColumns>
</SfGrid>
@code{
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
  Orders = Enumerable.Range(1, 75).Select(x => new Order()
  {
    OrderID = 1000 + x,
    CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
  })[new Random().Next(5)],
    Freight = 2.1 * x,
    OrderDate = DateTime.Now.AddDays(-x),
  }).ToList();
}
public class Order {
  public int? OrderID { get; set; }
  public string CustomerID { get; set; }
  public DateTime? OrderDate { get; set; }
  public double? Freight { get; set; }
}
```

The following shows selection and deselection of row

| Order ID | Customer Name | Order Date | Freight |
|----------|---------------|------------|---------|
| 1001 | ALFKI | 7/23/2019 | \$2.10 |
| 1002 | BLONP | 7/22/2019 | \$4.20 |
| 1003 | ALFKI | 7/21/2019 | \$6.30 |
| 1004 | BOLID | 7/20/2019 | \$8.40 |
| 1005 | ANTON | 7/19/2019 | \$10.50 |
| 1006 | ALFKI | 7/18/2019 | \$12.60 |
| 1007 | BOLID | 7/17/2019 | \$14.70 |
| 1008 | ALFKI | 7/16/2019 | \$16.80 |
| 1009 | ANANTR | 7/15/2019 | \$18.90 |
| 1010 | BOLID | 7/14/2019 | \$21.00 |
| 1011 | ALFKI | 7/13/2019 | \$23.10 |
| 1012 | ANANTR | 7/12/2019 | \$25.20 |

⏪ < 1 2 3 4 5 6 7 > ⏩
1 of 7 pages (75 items)

If multi selection is enabled, then first click on any selected row (without pressing Ctrl key), it will clear the multi selection and in second click on the same row, it will be unselected.

Drag selection

The Drag selection allows to perform the selection of the particular row or cell by performing mouse or touch dragging. To enable drag selection, set the [AllowDragSelection](#) property of the [GridSelectionSettings](#) as true.

CSHARP

```
@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders" EnableHover="false" AllowSelection="true"
AllowPaging="true">
<GridSelectionSettings AllowDragSelection="true"
Type="SelectionType.Multiple"></GridSelectionSettings>
<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumns>
</SfGrid>
@code{
public List<Order> Orders { get; set; }
```

```
protected override void OnInitialized()
{
    Orders = Enumerable.Range(1, 75).Select(x => new Order()
    {
        OrderID = 1000 + x,
        CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
    })[new Random().Next(5)],
        Freight = 2.1 * x,
        OrderDate = DateTime.Now.AddDays(-x),
    }).ToList();
}

public class Order {
    public int? OrderID { get; set; }
    public string CustomerID { get; set; }
    public DateTime? OrderDate { get; set; }
    public double? Freight { get; set; }
}
```

| Order ID | Customer Name | Order Date | Freight |
|----------|---------------|------------|---------|
| 1001 | ANANTR | 12/16/2021 | \$2.10 |
| 1002 | ANANTR | 12/15/2021 | \$4.20 |
| 1003 | ALFKI | 12/14/2021 | \$6.30 |
| 1004 | BLONP | 12/13/2021 | \$8.40 |
| 1005 | ALFKI | 12/12/2021 | \$10.50 |
| 1006 | ANTON | 12/11/2021 | \$12.60 |
| 1007 | ANTON | 12/10/2021 | \$14.70 |
| 1008 | BLONP | 12/9/2021 | \$16.80 |
| 1009 | BOLID | 12/8/2021 | \$18.90 |
| 1010 | BOLID | 12/7/2021 | \$21.00 |
| 1011 | BLONP | 12/6/2021 | \$23.10 |
| 1012 | BLONP | 12/5/2021 | \$25.20 |

« < 1 2 3 4 5 6 7 > »

1 of 7 pages (75 items)

* Drag selection supports both **Flow** and **Box** cell selection modes.

* The selection [Type](#) property should be set as Multiple, to select multiple rows or cells in grid by mouse/touch dragging.

* Also, drag selection supports checkbox selection.

Perform Toggle selection programmatically

You can perform toggle selection programmatically by using the **SelectRowAsync** method by passing **true** value as the second argument. In the below code example, we have programmatically toggle the row index 2.

ASPX-CS

```
@using Syncfusion.Blazor.Grids
@using Syncfusion.Blazor.Buttons
<SfButton Content="Perform Toggle Selection programmatically for RowIndex 3"
@onclick="Click"></SfButton>
<SfGrid @ref="Grid" DataSource="@Orders" AllowSelection="true"
AllowPaging="true">
<GridSelectionSettings EnableToggle="true"></GridSelectionSettings>
<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumns>
</SfGrid>
@code{
SfGrid<Order> Grid;
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
OrderDate = DateTime.Now.AddDays(-x),
}).ToList();
}
public class Order {
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
private async Task Click(Microsoft.AspNetCore.Components.Web.MouseEventArgs
args)
{
await Grid.SelectRowAsync(2, true);
}
}
```

Select row at initial rendering

To select a row at initial rendering, set the [SelectedRowIndex](#) value.

ASPX-CS

```
@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders" AllowSelection="true" SelectedRowIndex="2"
AllowPaging="true">
<GridSelectionSettings Mode="SelectionMode.Both"
Type="SelectionType.Multiple">
</GridSelectionSettings>
<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumns>
</SfGrid>
@code{
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
OrderDate = DateTime.Now.AddDays(-x),
}).ToList();
}
public class Order {
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
}
```

The following image will be displayed with row selected in initial Rendering

| Order ID | Customer Name | Order Date | Freight |
|----------|---------------|------------|----------|
| 10248 | VINET | 7/4/1996 | \$32.38 |
| 10249 | TOMSP | 7/5/1996 | \$11.61 |
| 10250 | HANAR | 7/8/1996 | \$65.83 |
| 10251 | VICTE | 7/8/1996 | \$41.34 |
| 10252 | SUPRD | 7/9/1996 | \$51.30 |
| 10253 | HANAR | 7/10/1996 | \$58.17 |
| 10254 | CHOPS | 7/11/1996 | \$22.98 |
| 10255 | RICSU | 7/12/1996 | \$148.33 |
| 10256 | WELLI | 7/15/1996 | \$13.97 |
| 10257 | HILAA | 7/16/1996 | \$81.91 |
| 10258 | ERNSH | 7/17/1996 | \$140.51 |
| 10259 | CENTC | 7/18/1996 | \$3.25 |

K < 1 2 3 4 5 6 7 8 9 10 ... > X 1 of 70 pages (830 items)

Get selected row indexes

You can get the selected row indexes by using [GetSelectedRowIndexes](#) method.

ASPX-CS

```
@using Syncfusion.Blazor
@using Syncfusion.Blazor.Grids
<span> The selected row index is : @SelectedValue </span>
<SfGrid @ref="@Grid" DataSource="@Orders" AllowFiltering="true"
AllowPaging="true" Height="315">
<GridSelectionSettings
Type="SelectionType.Multiple"></GridSelectionSettings>
<GridEvents RowSelected="GetSelectedRecords" TValue="Order"></GridEvents>
<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
TextAlign="TextAlign.Center" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
TextAlign="TextAlign.Center" Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Center"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Center" Width="140"></GridColumn>
</GridColumns>
</SfGrid>
@code{
SfGrid<Order> Grid;
public List<Order> Orders { get; set; }
public List<double> SelectedRowIndexes { get; set; }
```

```

public double[] TotalValue { get; set; }
public string SelectedValue;
protected override void OnInitialized()
{
    Orders = Enumerable.Range(1, 75).Select(x => new Order()
    {
        OrderID = 1000 + x,
        CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
    })[new Random().Next(5)],
        Freight = 2.1 * x,
        OrderDate = DateTime.Now.AddDays(-x),
    }).ToList();
}
public async Task GetSelectedRecords(RowSelectEventArgs<Order> args)
{
    SelectedRowIndexes = await this.Grid.GetSelectedRowIndexes();
    TotalValue = SelectedRowIndexes.ToArray();
    SelectedValue = "";
    foreach (var data in TotalValue)
    {
        SelectedValue = SelectedValue + " " + data;
    }
    StateHasChanged();
}
public class Order
{
    public int? OrderID { get; set; }
    public string CustomerID { get; set; }
    public DateTime? OrderDate { get; set; }
    public double? Freight { get; set; }
}
}

```

The following image will be displayed with selected row and its indexes

The selected row index is : 3 6 5 8


| Order ID | Customer Name | Order Date | Freight |
|----------|---------------|------------|---------|
| 1001 | BOLID | 9/3/2019 | \$2.10 |
| 1002 | ANANTR | 9/2/2019 | \$4.20 |
| 1003 | ANTON | 9/1/2019 | \$6.30 |
| 1004 | BOLID | 8/31/2019 | \$8.40 |
| 1005 | ALFKI | 8/30/2019 | \$10.50 |
| 1006 | BOLID | 8/29/2019 | \$12.60 |
| 1007 | ALFKI | 8/28/2019 | \$14.70 |
| 1008 | BOLID | 8/27/2019 | \$16.80 |
| 1009 | ANTON | 8/26/2019 | \$18.90 |

< 1 2 3 4 5 6 7 >
 1 of 7 pages (75 items)

Touch interaction

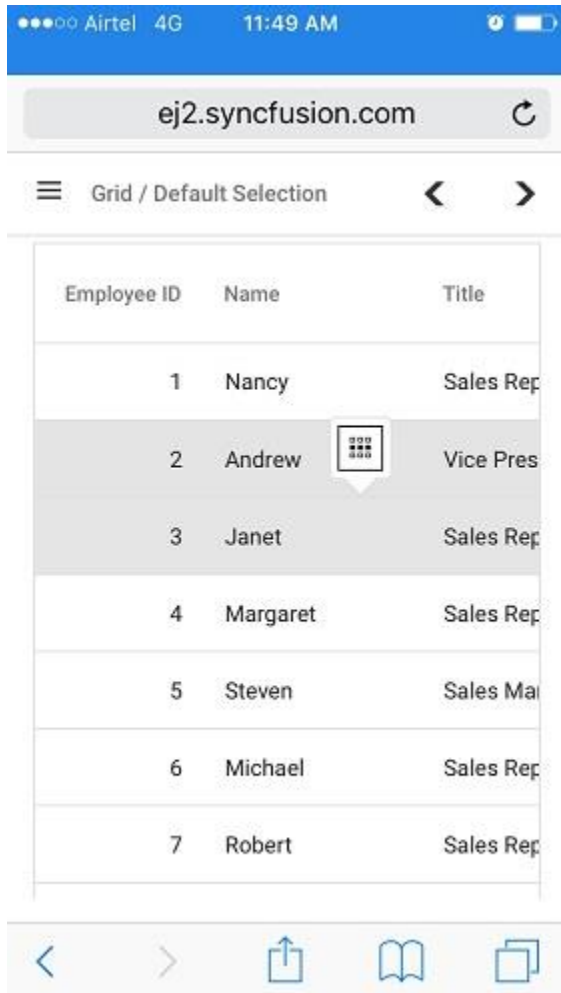
When you tap a datagrid row on touchscreen device, the tapped row is selected.

It also shows a popup  for multi-row selection.

To select multiple rows or cells, tap the popup  and then tap the desired rows or cells.

Multi-selection requires the selection [type](#) to be **multiple**.

The following screenshot represents a datagrid touch selection in the device.



Multiple selection based on condition

You can select multiple rows at the initial rendering of the datagrid by using [SelectRows](#) method. The initial selection is based on the condition which we given. Here the initial selection is based on the row which is having the **CustomerID** as **ALFKI**. We have used [GetCurrentViewRecords](#) method to get current page records and applied the condition.

ASPX-CS

```
@using Syncfusion.Blazor.Grids
<SfGrid @ref="@Grid" DataSource="@Orders" AllowFiltering="true"
AllowPaging="true" Height="315">
```

```

<GridSelectionSettings
Type="SelectionType.Multiple"></GridSelectionSettings>
<GridEvents DataBound="Data" TValue="Order"></GridEvents>
<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
TextAlign="TextAlign.Center" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer ID"
TextAlign="TextAlign.Center" Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Center"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Center" Width="120"></GridColumn>
</GridColumns>
</SfGrid>
@code{
SfGrid<Order> Grid;
public List<Order> Orders { get; set; }
List<double> SelectIndex { get; set; }
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
OrderDate = DateTime.Now.AddDays(-x),
}).ToList();
}
public async Task Data(object args)
{
var Source = await Grid.GetCurrentViewRecords();
var IndexNum = 0;
SelectIndex = new List<double>();
foreach (var record in Source)
{
if (record.CustomerID == "ALFKI")
{
SelectIndex.Add(IndexNum);
}
IndexNum++;
}
await Grid.SelectRows(SelectIndex.ToArray());
}
public class Order
{
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
}

```

The following image will be displayed with Multiple selection

| Order ID | Customer ID | Order Date | Freight |
|----------|-------------|------------|---------|
| 1004 | ANANTR | 8/31/2019 | \$8.40 |
| 1005 | ALFKI | 8/30/2019 | \$10.50 |
| 1006 | ALFKI | 8/29/2019 | \$12.60 |
| 1007 | ANANTR | 8/28/2019 | \$14.70 |
| 1008 | ALFKI | 8/27/2019 | \$16.80 |
| 1009 | BLONP | 8/26/2019 | \$18.90 |
| 1010 | ANTON | 8/25/2019 | \$21.00 |
| 1011 | BLONP | 8/24/2019 | \$23.10 |
| 1012 | ALFKI | 8/23/2019 | \$25.20 |

K < 1 2 3 4 5 6 7 > > 1 of 7 pages (75 items)

Simple multiple row selection

You can select multiple rows by clicking on rows one by one. This will not deselect the previously selected rows. To deselect the previously selected row, you can click on the selected row. You can enable this behavior by using [EnableSimpleMultiRowSelection](#) property of **GridSelectionSettings** component.

ASPX-CS

```
@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders" AllowSelection="true" AllowPaging="true">
  <GridSelectionSettings EnableSimpleMultiRowSelection="true"
    Type="SelectionType.Multiple"></GridSelectionSettings>
  <GridColumns>
    <GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
      TextAlign="TextAlign.Right" Width="120"></GridColumn>
    <GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
      Width="150"></GridColumn>
    <GridColumn Field=@nameof(Order.OrderDate) HeaderText="Order Date"
      Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
      Width="130"></GridColumn>
    <GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
      TextAlign="TextAlign.Right" Width="120"></GridColumn>
  </GridColumns>
</SfGrid>

@code{
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
    Orders = Enumerable.Range(1, 75).Select(x => new Order()
    {
        OrderID = 1000 + x,
        CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
    })[new Random().Next(5)],
        Freight = 2.1 * x,
        OrderDate = DateTime.Now.AddDays(-x),
    }).ToList();
    }
    public class Order {
    public int? OrderID { get; set; }
    }
```

```
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
```

You can refer to our [Blazor DataGrid](#) feature tour page for its groundbreaking feature representations. You can also explore our [Blazor DataGrid example](#) to understand how to present and manipulate data.

Aggregates in Blazor DataGrid Component

Aggregate values are displayed in the footer, group footer, or group caption of the Grid. It can be configured through [GridAggregates](#) component. [Field](#) and

[Type](#) are the minimum properties required to represent an aggregate column.

Built-in aggregate types

The aggregate type should be specified in the [Type](#) property to configure an aggregate column.

The built-in aggregates are,

- Sum
- Average
- Min
- Max
- Count
- TrueCount
- FalseCount

* Multiple types for a column is supported only when one of the aggregate templates is used.

Footer aggregate

Footer aggregate value is calculated for all the rows, and it is displayed in the footer cells. Use the **FooterTemplate** tag directive to render the aggregate value in footer cells. **FooterTemplate** should be provided within the **GridAggregateColumn** directive.

To access the aggregate values inside the **FooterTemplate**, you can use the implicit named parameter **context**. You can type cast the **context** as **AggregateTemplateContext** to get aggregate values inside template.

ASPX-CS

```
@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders" AllowPaging="true">
  <GridPageSettings PageSize="8"></GridPageSettings>
  <GridAggregates>
    <GridAggregate>
      <GridAggregateColumns>
        <GridAggregateColumn Field=@nameof(Order.Freight) Type="AggregateType.Sum"
          Format="C2">
          <FooterTemplate>
            @ {
              var aggregate = (context as AggregateTemplateContext);
            }
          </div>
          <p>Sum: @aggregate.Sum</p>
        </GridAggregateColumn>
      </GridAggregateColumns>
    </GridAggregate>
  </GridAggregates>
</SfGrid>
```

```

</div>
}
</FooterTemplate>
</GridAggregateColumn>
</GridAggregateColumns>
</GridAggregate>
<GridAggregate>
<GridAggregateColumns>
<GridAggregateColumn Field=@nameof(Order.Freight)
Type="AggregateType.Average" Format="C2">
<FooterTemplate>
@{
var aggregate = (context as AggregateTemplateContext);
<div>
<p>Average: @aggregate.Average</p>
</div>
}
</FooterTemplate>
</GridAggregateColumn>
</GridAggregateColumns>
</GridAggregate>
</GridAggregates>
<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText="Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumns>
</SfGrid>
@code{
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
OrderDate = DateTime.Now.AddDays(-x),
}).ToList();
}
public class Order
{
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
}

```

The following image represents the FooterTemplate with aggregates

| Order ID | Customer Name | Order Date | Freight |
|-----------------------------|---------------|------------|--------------------------|
| 1001 | ANANTR | 7/29/2019 | \$2.10 |
| 1002 | BLONP | 7/28/2019 | \$4.20 |
| 1003 | ANTON | 7/27/2019 | \$6.30 |
| 1004 | BOLID | 7/26/2019 | \$8.40 |
| 1005 | ANTON | 7/25/2019 | \$10.50 |
| 1006 | BLONP | 7/24/2019 | \$12.60 |
| 1007 | BLONP | 7/23/2019 | \$14.70 |
| 1008 | ANANTR | 7/22/2019 | \$16.80 |
| | | | Sum: \$75.60 |
| | | | Average: \$9.45 |
| K < 1 2 3 4 5 6 7 8 ... > X | | | 1 of 10 pages (75 items) |

The aggregate values must be accessed inside the template using their corresponding [Type](#) name.

How to format aggregate value

You can format the aggregate value result by using the [Format](#) property.

To access the aggregate values inside the **FooterTemplate**, you can use the implicit named parameter **context**. You can type cast the ***context** as **AggregateTemplateContext** to get aggregate values inside template.

ASPX-CS

```
@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders" AllowPaging="true">
  <GridPageSettings PageSize="8"></GridPageSettings>
  <GridAggregates>
    <GridAggregate>
      <GridAggregateColumns>
        <GridAggregateColumn Field=@nameof(Order.Freight) Type="AggregateType.Sum"
          Format="C2">
          <FooterTemplate>
            @ {
              var aggregate = (context as AggregateTemplateContext);
            <div>
              <p>Sum: @aggregate.Sum</p>
            </div>
          }
          </FooterTemplate>
        </GridAggregateColumn>
      </GridAggregateColumns>
    </GridAggregate>
  </GridAggregates>
</SfGrid>
```



```

<GridAggregate>
  <GridAggregateColumns>
    <GridAggregateColumn Field=@nameof(Order.Freight)
      Type="AggregateType.Average" Format="C2">
      <FooterTemplate>
        @{
          var aggregate = (context as AggregateTemplateContext);
        }
        <div>
          <p>Average: @aggregate.Average</p>
        </div>
      </FooterTemplate>
    </GridAggregateColumn>
  </GridAggregateColumns>
</GridAggregate>
</GridAggregates>
<GridColumns>
  <GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
    TextAlign="TextAlign.Right" Width="120"></GridColumn>
  <GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
    Width="150"></GridColumn>
  <GridColumn Field=@nameof(Order.OrderDate) HeaderText="Order Date"
    Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
    Width="130"></GridColumn>
  <GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
    TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumns>
</SfGrid>
@code{
  public List<Order> Orders { get; set; }
  protected override void OnInitialized()
  {
    Orders = Enumerable.Range(1, 75).Select(x => new Order()
    {
      OrderID = 1000 + x,
      CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
    })[new Random().Next(5)],
      Freight = 2.1 * x,
      OrderDate = DateTime.Now.AddDays(-x),
    }).ToList();
  }
  public class Order
  {
    public int? OrderID { get; set; }
    public string CustomerID { get; set; }
    public DateTime? OrderDate { get; set; }
    public double? Freight { get; set; }
  }
}

```

Group and caption aggregate

Group and caption aggregate values are calculated from the current group items. If

GroupFooterTemplate is provided, the aggregate values will be displayed in the group footer cells and if **GroupCaptionTemplate** is provided, aggregate values will be displayed in the group caption cells.

Both **GroupCaptionTemplate** and **GroupFooterTemplate** should be provided within the **GridAggregateColumn** directive.

To access the aggregate values inside the **GroupFooterTemplate** and **GroupCaptionTemplate**, you can use the implicit named parameter **context**. You can type cast the **context** as **AggregateTemplateContext** to get aggregate values inside template.

ASPX-CS

```
@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Products" AllowGrouping="true" AllowPaging="true">
  <GridGroupSettings Columns=@Units></GridGroupSettings>
  <GridAggregates>
    <GridAggregate>
      <GridAggregateColumns>
        <GridAggregateColumn Field=@nameof(Product.UnitsInStock)
          Type="AggregateType.Sum">
          <GroupFooterTemplate>
            @{
              var aggregate = (context as AggregateTemplateContext);
            }
            <div>
              <p>Total units: @aggregate.Sum</p>
            </div>
          </GroupFooterTemplate>
        </GridAggregateColumn>
        <GridAggregateColumn Field=@nameof(Product.Discontinued)
          Type="AggregateType.TrueCount">
          <GroupFooterTemplate>
            @{
              var aggregate = (context as AggregateTemplateContext);
            }
            <div>
              <p>Truecount: @aggregate.TrueCount</p>
            </div>
          </GroupFooterTemplate>
        </GridAggregateColumn>
        <GridAggregateColumn Field=@nameof(Product.UnitsInStock)
          Type="AggregateType.Max">
          <GroupCaptionTemplate>
            @{
              var aggregate = (context as AggregateTemplateContext);
            }
            <div>
              <p>Maximum: @aggregate.Max</p>
            </div>
          </GroupCaptionTemplate>
        </GridAggregateColumn>
      </GridAggregateColumns>
    </GridAggregate>
  </GridAggregates>
  <GridColumn>
    <GridCaptionTemplate>
      <div>
        <p>Product Name</p>
      </div>
    </GridCaptionTemplate>
    <GridFooterTemplate>
      <div>
        <p>Product Name</p>
      </div>
    </GridFooterTemplate>
  </GridColumn>
  <GridColumn Field=@nameof(Product.QuantityPerUnit) HeaderText="Quantity Per
    Unit" Width="150"></GridColumn>
```

```
<GridColumn Field=@nameof(Product.UnitsInStock) HeaderText="Units In Stock"
TextAlign="TextAlign.Right" Width="130"></GridColumn>
<GridColumn Field=@nameof(Product.Discontinued) HeaderText="Discontinued"
TextAlign="TextAlign.Right" DisplayAsCheckBox="true"
Type="ColumnType.Boolean"></GridColumn>
</GridColumn>
</SfGrid>
@code{
public List<Product> Products { get; set; }
private string[] Units = (new string[] { "QuantityPerUnit" });
protected override void OnInitialized()
{
Products = Enumerable.Range(1, 10).Select(x => new Product
{
ProductName = (new string[] { "Chai", "Chang", "Aniseed Syrup", "Chef
Anton's Cajun Seasoning", "Chef Anton's Gumbo Mix" })[new Random().Next(5)],
QuantityPerUnit = (new string[] { "10 boxes x 20 bags", "24 - 12 oz
bottles", "12 - 550 ml bottles", "48 - 6 oz jars", "36 boxes" })[new
Random().Next(5)],
UnitsInStock = x,
Discontinued = (new bool[] { true, false})[new Random().Next(2)]
}).ToList();
}
public class Product
{
public string ProductName { get; set; }
public string QuantityPerUnit { get; set; }
public int UnitsInStock { get; set; }
public bool Discontinued { get; set; }
}
}
```

The following image represents the Group and Caption template with aggregates.

| Quanti... ↑ × | | |
|--------------------|----------------|-------------------------------------|
| Product Name | Units In Stock | Discontinued |
| ^ Quantity Per ... | Maximum: 7 | |
| Chef Anton's Ca... | 7 | <input checked="" type="checkbox"/> |
| Total units: 7 | | Truecount: 1 |
| ^ Quantity Per ... | Maximum: 8 | |
| Aniseed Syrup | 5 | <input checked="" type="checkbox"/> |
| Chang | 8 | <input type="checkbox"/> |
| Total units: 13 | | Truecount: 1 |
| ^ Quantity Per ... | Maximum: 9 | |
| Chai | 9 | <input checked="" type="checkbox"/> |
| Total units: 9 | | Truecount: 1 |

The aggregate values must be accessed inside the template using their corresponding [Type](#) name.

Custom aggregate

To calculate the aggregate value with your own aggregate functions, use the custom aggregate option.

To use Custom aggregate, specify the **AggregateType** as **Custom** in **GridAggregateColumn** directive and provide custom aggregate function inside the **FooterTemplate** as follows,

ASPX-CS

```
@using Syncfusion.Blazor.Grids
<SfGrid @ref="Grid" DataSource="@Products" AllowPaging="true">
<GridAggregates>
<GridAggregate>
<GridAggregateColumns>
<GridAggregateColumn Field=@nameof(Product.TotalSales)
Type="AggregateType.Custom">
<FooterTemplate>
@{
<div>
<p>Custom: @GetWeightedAggregate()</p>
</div>
}
</FooterTemplate>
</GridAggregateColumn>
</GridAggregateColumns>
</GridAggregate>
</GridAggregates>
</GridColumns>
```

```

<GridColumn Field=@nameof(Product.ProductName) HeaderText="Product Name"
TextAlign="TextAlign.Right" Width="150"></GridColumn>
<GridColumn Field=@nameof(Product.QuantityPerUnit) HeaderText="Quantity Per
Unit" Width="150"></GridColumn>
<GridColumn Field=@nameof(Product.TotalSales) HeaderText="TotalSales"
TextAlign="TextAlign.Right" Width="130"></GridColumn>
<GridColumn Field=@nameof(Product.TotalCosts) HeaderText="TotalCosts"
TextAlign="TextAlign.Right" Width="180"></GridColumn>
</GridColumn>
</SfGrid>
@code{
SfGrid<Product> Grid { get; set; }
public List<Product> Products { get; set; }
public string GetWeightedAggregate()
{
// Here, we can calculate custom aggregate operations and return the result
return Queryable.Sum(Products.Select(x => (x.TotalSales + x.TotalCosts) /
x.TotalSales).AsQueryable()).ToString();
}
protected override void OnInitialized()
{
Products = Enumerable.Range(1, 5).Select(x => new Product
{
ProductName = (new string[] { "Chai", "Chang", "Aniseed Syrup", "Chef
Anton's Cajun Seasoning", "Chef Anton's Gumbo Mix" })[new Random().Next(5)],
QuantityPerUnit = (new string[] { "10 boxes x 20 bags", "24 - 12 oz
bottles", "12 - 550 ml bottles", "48 - 6 oz jars", "36 boxes" })[new
Random().Next(5)],
TotalSales = 100 * x,
TotalCosts = 200 * x,
Discontinued = (new bool[] { true, false })[new Random().Next(2)]
}).ToList();
}
public class Product
{
public string ProductName { get; set; }
public string QuantityPerUnit { get; set; }
public int TotalSales { get; set; }
public int TotalCosts { get; set; }
public bool Discontinued { get; set; }
}
}

```

You can refer to the [Blazor DataGrid](#) feature tour page for its groundbreaking feature representations. You can also explore [Blazor DataGrid example](#) to understand how to present and manipulate data.

Handling Aggregates in Custom Adaptor

When using Custom Adaptor, the aggregates has to be handled in the Read/ReadAsync method of Custom adaptor.

The following sample code demonstrates implementing the aggregates for the custom bounded data,

ASPX-CS

```

@using Syncfusion.Blazor.Grids
@using Syncfusion.Blazor.Data

```

```

@using Syncfusion.Blazor
<SfGrid TValue="Order" AllowPaging="true">
<SfDataManager AdaptorInstance="@typeof(CustomAdaptor) "
Adaptor="Adaptors.CustomAdaptor"></SfDataManager>
<GridPageSettings PageSize="8"></GridPageSettings>
<GridAggregates>
<GridAggregate>
<GridAggregateColumns>
<GridAggregateColumn Field=@nameof(Order.Freight) Type="AggregateType.Sum"
Format="C2">
<FooterTemplate>
@{
var aggregate = (context as AggregateTemplateContext);
<div>
<p>Sum: @aggregate.Sum</p>
</div>
}
</FooterTemplate>
</GridAggregateColumn>
</GridAggregateColumns>
</GridAggregate>
<GridAggregate>
<GridAggregateColumns>
<GridAggregateColumn Field=@nameof(Order.Freight)
Type="AggregateType.Average" Format="C2">
<FooterTemplate>
@{
var aggregate = (context as AggregateTemplateContext);
<div>
<p>Average: @aggregate.Average</p>
</div>
}
</FooterTemplate>
</GridAggregateColumn>
</GridAggregateColumns>
</GridAggregate>
</GridAggregates>
<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumns>
</SfGrid>
@code{
public static List<Order> Orders { get; set; }
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,

```

```
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
OrderDate = DateTime.Now.AddDays(-x),
}).ToList();
}
public class Order
{
    public int? OrderID { get; set; }
    public string CustomerID { get; set; }
    public DateTime? OrderDate { get; set; }
    public double? Freight { get; set; }
}
public class CustomAdaptor : DataAdaptor
{
    // Performs data Read operation
    public override object Read(DataManagerRequest dm, string key = null)
    {
        IEnumerable<Order> DataSource = Orders;
        if (dm.Search != null && dm.Search.Count > 0)
        {
            // Searching
            DataSource = DataOperations.PerformSearching(DataSource, dm.Search);
        }
        if (dm.Sorted != null && dm.Sorted.Count > 0)
        {
            // Sorting
            DataSource = DataOperations.PerformSorting(DataSource, dm.Sorted);
        }
        if (dm.Where != null && dm.Where.Count > 0)
        {
            // Filtering
            DataSource = DataOperations.PerformFiltering(DataSource, dm.Where,
            dm.Where[0].Operator);
        }
        int count = DataSource.Cast<Order>().Count();
        if (dm.Skip != 0)
        {
            //Paging
            DataSource = DataOperations.PerformSkip(DataSource, dm.Skip);
        }
        if (dm.Take != 0)
        {
            DataSource = DataOperations.PerformTake(DataSource, dm.Take);
        }
        DataResult DataObject = new DataResult();
        if (dm.Aggregates != null) // Aggregation
        {
            DataObject.Result = DataSource;
            DataObject.Count = count;
            DataObject.Aggregates = DataUtil.PerformAggregation(DataSource,
            dm.Aggregates);
            return dm.RequiresCounts ? DataObject : (object)DataSource;
        }
        return dm.RequiresCounts ? new DataResult() { Result = DataSource, Count =
        count } : (object)DataSource;
    }
}
```

```
}
}
```

<!-- Reactive aggregate update

When using batch editing, the aggregate values will be refreshed on every cell save. The footer, group footer, and group caption aggregate values will be refreshed.

Adding a new record to the grouped DataGrid will not refresh the aggregate values.


ASPX-CS

-->

ASPX-CS

```
@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders" Toolbar="@ (new List<object>() { "Print" })"
AllowPaging="true">
<GridPageSettings PageSize="8"></GridPageSettings>
<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" TextAlign="TextAlign.Right" Width="130"
Type="ColumnType.Date"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumns>
</SfGrid>
@code{
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
OrderDate = DateTime.Now.AddDays(-x),
}).ToList();
}
public class Order
{
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
}
```


The following image represents DataGrid with print toolbar item,

<div>  Print </div>				
	Order ID	Customer Name	Order Date	Freight
	1001	ANTON	9/9/2019	\$2.10
	1002	ANTON	9/8/2019	\$4.20
	1003	ANTON	9/7/2019	\$6.30
	1004	ANTON	9/6/2019	\$8.40
	1005	ALFKI	9/5/2019	\$10.50
	1006	BOLID	9/4/2019	\$12.60
	1007	BLONP	9/3/2019	\$14.70
	1008	ANANTR	9/2/2019	\$16.80
<div> <div> K < 1 2 3 4 5 6 7 8 9 10 > X </div> <div>1 of 10 pages (75 items)</div> </div>				

Page setup

Some of the print options cannot be configured through code. So, you have to customize the layout, paper size, and margin options using the browser page setup dialog. Please refer to the following links to know more about this,

- [Chrome](#)
- [Firefox](#)
- [Safari](#)
- [IE](#)

Print using an external button

To print the datagrid from an external button, invoke the print method using the datagrid reference.

The below sample code demonstrates invoking print using an external button,

ASPX-CS

```
@using Syncfusion.Blazor.Buttons
@using Syncfusion.Blazor.Grids
<SfButton Content="Print" OnClick="PrintContent"></SfButton>
<SfGrid @ref="DefaultGrid" DataSource="@Orders" AllowPaging="true">
<GridPageSettings PageSize="8"></GridPageSettings>
<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="120"></GridColumn>
```

```
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" TextAlign="TextAlign.Right" Width="130"
Type="ColumnType.Date"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumn>
</SfGrid>
@code{
public List<Order> Orders { get; set; }
private SfGrid<Order> DefaultGrid;
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
OrderDate = DateTime.Now.AddDays(-x),
}).ToList();
}
public class Order
{
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
public void PrintContent()
{
this.DefaultGrid.Print();
}
}
```

The following image represents DataGrid with external button for invoking print operation,

Print

Order ID	Customer Name	Order Date	Freight
1001	BLONP	9/9/2019	\$2.10
1002	ANTON	9/8/2019	\$4.20
1003	ALFKI	9/7/2019	\$6.30
1004	BLONP	9/6/2019	\$8.40
1005	BLONP	9/5/2019	\$10.50
1006	ALFKI	9/4/2019	\$12.60
1007	ANANTR	9/3/2019	\$14.70
1008	ANANTR	9/2/2019	\$16.80

K < 1 2 3 4 5 6 7 8 9 10 > X
1 of 10 pages (75 items)

Print the visible page

By default, the datagrid prints all the pages. To print the current page alone, set the [PrintMode](#) value as **CurrentPage**.

The below sample code demonstrates this,

ASPX-CS

```
@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders" Toolbar="@ (new List<object>() { "Print" })"
PrintMode=PrintMode.CurrentPage AllowPaging="true">
<GridPageSettings PageSize="8"></GridPageSettings>
<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" TextAlign="TextAlign.Right" Width="130"
Type="ColumnType.Date"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumns>
</SfGrid>
@code{
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
```

```

CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
OrderDate = DateTime.Now.AddDays(-x),
}).ToList();
}
public class Order
{
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
}

```

<!-- Print the hierarchy datagrid

By default, the datagrid will print the master and expanded child grids alone. You can change the print option by using the [HierarchyPrintMode](#) property of the Grid component. The available options are,

Mode | Behavior

Expanded | Prints the master datagrid with expanded child grids.

All | Prints the master datagrid with all the child grids.

None | Prints the master datagrid alone.

This is demonstrated in the below sample code,

ASPX-CS

```

@using Syncfusion.Blazor.Grids
@{
GridModel<object> ChildGridData = new GridModel<object>()
{
DataSource = Orders,
QueryString = "EmployeeID",
Columns = new List<GridColumn> {
new GridColumn() { Field="OrderID", HeaderText="OrderID", Width="110" },
new GridColumn() { Field="CustomerName", HeaderText="CustomerName",
Width="110"},
new GridColumn() { Field="ShipCountry", HeaderText="ShipCountry",
Width="110" }
}
};
}
<SfGrid DataSource="@Employees" Toolbar="@ (new List<object>() { "Print" })"
HierarchyPrintMode=HierarchyGridPrintMode.Expanded ChildGrid="ChildGridData"
Height="315px">
<GridColumns>
<GridColumn Field=@nameof(EmployeeData.EmployeeID) HeaderText="EmployeeID"
Width="110"> </GridColumn>
<GridColumn Field=@nameof(EmployeeData.FirstName) HeaderText="First Name"
Width="110"> </GridColumn>
<GridColumn Field=@nameof(EmployeeData.City) HeaderText="Last Name"
Width="110"></GridColumn>

```

```

<GridColumn Field=@nameof(EmployeeData.Country) HeaderText="Country"
Width="110"></GridColumn>
</GridColumn>
</SfGrid>
@code{
public List<EmployeeData> Employees { get; set; }
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
Employees = Enumerable.Range(1, 9).Select(x => new EmployeeData()
{
EmployeeID = x,
FirstName = (new string[] { "Nancy", "Andrew", "Janet", "Margaret", "Steven"
})[new Random().Next(5)],
City = (new string[] { "Seattle", "Tacoma", "Redmond", "Kirkland", "London"
})[new Random().Next(5)],
Country = (new string[] { "USA", "UK" })[new Random().Next(2)],
}).ToList();
Orders = Enumerable.Range(1, 9).Select(x => new Order()
{
EmployeeID = x,
OrderID = 1000 + x,
CustomerName = (new string[] { "Nancy", "Andrew" })[new Random().Next(2)],
ShipCountry = (new string[] { "USA", "UK" })[new Random().Next(2)],
}).ToList();
}
public class EmployeeData
{
public int? EmployeeID { get; set; }
public string FirstName { get; set; }
public string LastName { get; set; }
public string Title { get; set; }
public DateTime? HireDate { get; set; }
public string City { get; set; }
public string Country { get; set; }
}
public class Order
{
public int? EmployeeID { get; set; }
public int? OrderID { get; set; }
public string CustomerName { get; set; }
public string ShipCountry { get; set; }
}
}

```

The following image represents Hierarchial Grid with print toolbar item,

Print

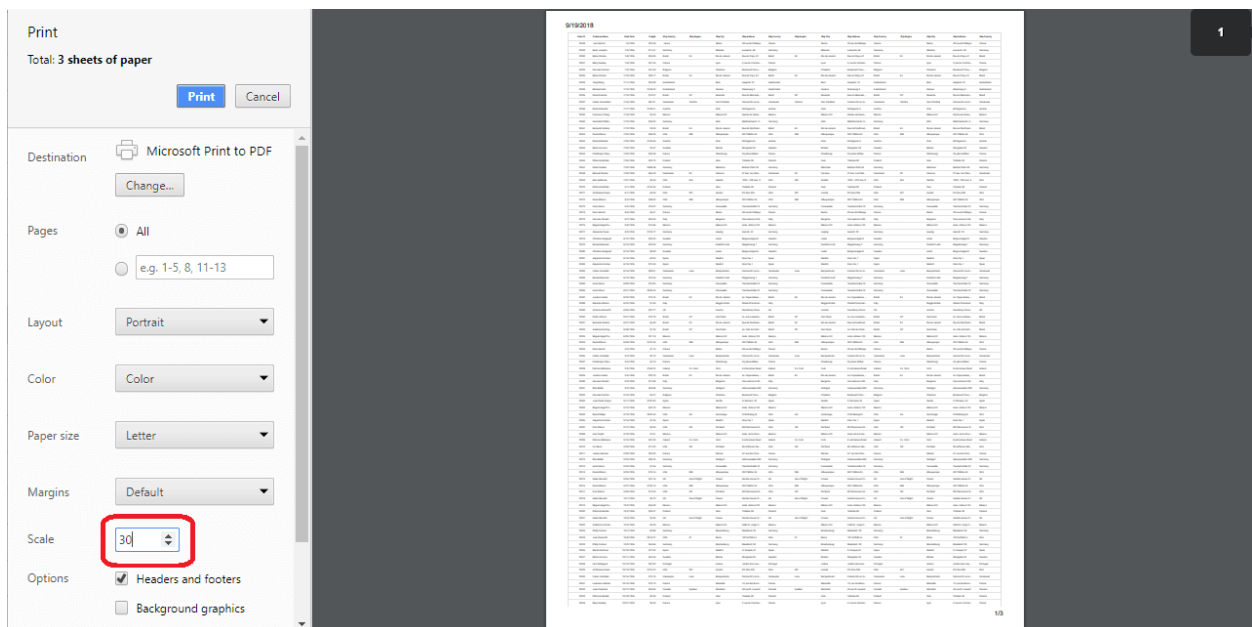
| EmployeeID | First Name | Last Name | Country | | | | | | |
|---|--------------|-------------|---------|---------|--------------|-------------|------|--------|----|
| 1 | Janet | Seattle | USA | | | | | | |
| <table><thead><tr><th>OrderID</th><th>CustomerName</th><th>ShipCountry</th></tr></thead><tbody><tr><td>1001</td><td>Andrew</td><td>UK</td></tr></tbody></table> | | | | OrderID | CustomerName | ShipCountry | 1001 | Andrew | UK |
| OrderID | CustomerName | ShipCountry | | | | | | | |
| 1001 | Andrew | UK | | | | | | | |
| 2 | Margaret | Tacoma | UK | | | | | | |
| 3 | Steven | Redmond | UK | | | | | | |
| 4 | Janet | Redmond | UK | | | | | | |
| 5 | Nancy | Seattle | UK | | | | | | |
| 6 | Janet | Kirkland | USA | | | | | | |
| 7 | Margaret | Tacoma | UK | | | | | | |

-->

Print large number of columns

By default, the browser uses A4 as page size option to print pages and in order to adapt to the size of the page, the browser print preview will auto-hide the overflowed contents. Hence datagrid with large number of columns will be cut off for adapting to the size of the print page.

To print large number of columns, adjust the scale option from print option panel based on your content size.



<!-- Show or hide columns while Printing

You can show a hidden column or hide a visible column while printing the datagrid using ToolbarClick and PrintComplete events.

In the ToolbarClick event, we can show or hide columns by modifying the Visible property value of the GridColumn component.

Then in the PrintComplete event, we can reverse the state back to the previous state.

In the below example, we have **CustomerID** as a hidden column in the datagrid. While printing, we have changed **CustomerID** to visible column and **Freight** as hidden column.

ASPX-CS

-->

CSHARP

```
@using Syncfusion.Blazor.Grids
@using Syncfusion.Blazor.Buttons
<div style="position:relative; min-height: 500px;">
<SfGrid DataSource="@Orders" AllowSorting="true" AllowFiltering="true"
EnableAdaptiveUI="true" Toolbar="@ (new List<string>() { "Add", "Edit",
"Delete", "Cancel", "Update", "Search" })" Height="100%" Width="100%"
AllowPaging="true">
<GridFilterSettings Type="@FilterType.Excel"></GridFilterSettings>
<GridEditSettings AllowAdding="true" AllowEditing="true"
AllowDeleting="true" Mode="EditMode.Dialog"></GridEditSettings>
<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
IsPrimaryKey="true" Width="80"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
Width="120"></GridColumn>
</GridColumns>
</SfGrid>
</div>
@code{
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
OrderDate = DateTime.Now.AddDays(-x),
}).ToList();
}
public class Order
{
public int? OrderID { get; set; }
```

```
public string CustomerID { get; set; }  
public DateTime? OrderDate { get; set; }  
public double? Freight { get; set; }  
}  
}
```

O...	Custom...	Order Date
1001	ANTON	10/12/2021
1002	BLONP	10/11/2021
1003	ANANTR	10/10/2021
1004	ANANTR	10/9/2021
1005	BOLID	10/8/2021
1006	ANTON	10/7/2021
1007	ANTON	10/6/2021
1008	ALFKI	10/5/2021
1009	ANTON	10/4/2021
1010	ANTON	10/3/2021

1 of 7 pages

1. This UI is common for both horizontal and vertical mode of rendering when EnableAdaptiveUI is enabled.









Vertical Mode

The DataGrid will render the row elements vertically while setting the [RowRenderingMode](#) property value as **Vertical**.

CSHARP

```
@using Syncfusion.Blazor.Grids
@using Syncfusion.Blazor.Buttons
<div style="position:relative; min-height: 500px;">
<SfGrid DataSource="@Orders" AllowSorting="true" AllowFiltering="true"
EnableAdaptiveUI="true" Toolbar="@ (new List<string>() { "Add", "Edit",
"Delete", "Cancel", "Update", "Search" })"
RowRenderingMode="RowDirection.Vertical" Height="100%" Width="100%"
AllowPaging="true">
<GridFilterSettings Type="@FilterType.Excel"></GridFilterSettings>
<GridEditSettings AllowAdding="true" AllowEditing="true"
AllowDeleting="true" Mode="EditMode.Dialog"></GridEditSettings>
<GridAggregates>
<GridAggregate>
<GridAggregateColumns>
<GridAggregateColumn Field=@nameof(Order.Freight) Type="AggregateType.Sum"
Format="C2">
<FooterTemplate>
@{
var aggregate = (context as AggregateTemplateContext);
<div>
<p>Sum: @aggregate.Sum</p>
</div>
}
</FooterTemplate>
</GridAggregateColumn>
</GridAggregateColumns>
</GridAggregate>
</GridAggregates>
<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
IsPrimaryKey="true" Width="80"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText="Order Date"
Format="d" Type="ColumnType.Date" Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
Width="120"></GridColumn>
</GridColumns>
</SfGrid>
</div>
@code{
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
OrderDate = DateTime.Now.AddDays(-x),
```

```
}).ToList();  
}  
public class Order  
{  
    public int? OrderID { get; set; }  
    public string CustomerID { get; set; }  
    public DateTime? OrderDate { get; set; }  
    public double? Freight { get; set; }  
}  
}
```

   	
Order ID	1001
Customer Name	BOLID
Order Date	10/12/2021
Freight	\$2.10
Order ID	1002
Customer Name	BOLID
Order Date	10/11/2021
Freight	\$4.20
Order ID	1003
Customer Name	ANTON
Freight	Sum: \$5,985.00
  1 of 7 pages  	

Supported features in vertical mode

The following features are only supported in vertical row rendering:

- Paging
- Sorting
- Filtering
- Selection
- Dialog Editing
- Aggregate
- Virtual scroll
- Toolbar

State Management in Blazor DataGrid Component

State management allows users to save and load grid state. The grid will use user-provided state to render instead of its properties provided declaratively.

Below properties can be saved and loaded into grid later.

Property |

Columns |

GridFilterSettings |

GridSortSettings |

GridGroupSettings |

GridPageSettings |

Enabling persistence in Grid

State persistence allows the Grid to retain the current grid state in the browser local storage for state maintenance. This action is handled through the `EnablePersistence` property which is set to false by default. When it is set to true, some properties of the Grid will be retained even after refreshing the page.

The state will be persisted based on **ID** property. So, it is recommended to explicitly set the **ID** property for Grid.

ASPX-CS

```
@using Syncfusion.Blazor.Grids
<SfGrid ID="Grid" DataSource="@Orders" Height="315" EnablePersistence="true"
AllowPaging="true" AllowFiltering="true" AllowGrouping="true"
AllowSorting="true">
  <GridColumns>
    <GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
    TextAlign="TextAlign.Right" Width="120"></GridColumn>
    <GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
    Width="150"></GridColumn>
    <GridColumn Field=@nameof(Order.OrderDate) HeaderText="Order Date"
    Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
    Width="130"></GridColumn>
    <GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
    TextAlign="TextAlign.Right" Width="120"></GridColumn>
  </GridColumns>
```

```

</SfGrid>
@code {
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
OrderDate = DateTime.Now.AddDays(-x),
}).ToList();
}
public class Order
{
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
}

```

You can use [ResetPersistData](#) method to reset grid state to its original state. This will clear persisted data in window local storage and renders grid with its original property values.

Handling grid state manually

You can handle the grid's state manually by using in-built state persistence methods. You can use [GetPersistData](#), [SetPersistData](#), [ResetPersistData](#) methods of grid to save, load and reset the Grid's persisted state manually. [GetPersistData](#) method will return grid current state as a string value, which is suitable for sending them over network and storing in data bases.

ASPX-CS

```

@using Syncfusion.Blazor.Grids
@using Syncfusion.Blazor.Buttons
<SfButton OnClick="@ (async () => _state = await Grid.GetPersistData())">Save
State</SfButton>
<SfButton OnClick="@ (() => Grid.SetPersistData(_state))">Set
State</SfButton>
<SfButton OnClick="@ (() => Grid.ResetPersistData())">Reset State</SfButton>
<SfGrid @ref="Grid" ID="GridOneTwo" DataSource="@Orders" Height="315"
EnablePersistence="true" AllowPaging="true" AllowFiltering="true"
AllowGrouping="true" AllowSorting="true">
<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText="Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumns>
</SfGrid>

```

```
@code {
    SfGrid<Order> Grid;
    public List<Order> Orders { get; set; }
    public string _state;
    protected override void OnInitialized()
    {
        Orders = Enumerable.Range(1, 75).Select(x => new Order()
        {
            OrderID = 1000 + x,
            CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
            })[new Random().Next(5)],
            Freight = 2.1 * x,
            OrderDate = DateTime.Now.AddDays(-x),
        }).ToList();
    }
    public class Order
    {
        public int? OrderID { get; set; }
        public string CustomerID { get; set; }
        public DateTime? OrderDate { get; set; }
        public double? Freight { get; set; }
    }
}
```

You can refer to our [Blazor DataGrid](#) feature tour page for its groundbreaking feature representations. You can also explore our [Blazor DataGrid example](#) to understand how to present and manipulate data.

Globalization in Blazor DataGrid Component

Add **UseRequestLocalization** middle-ware in Configure method in **Startup.cs** file to get browser Culture Info.

Refer the following code to add configuration in Startup.cs file

CSHARP

```
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Localization;
namespace BlazorApplication
{
    public class Startup
    {
        ....
        ....
        public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
        {
            app.UseRequestLocalization();
            ....
            ....
        }
    }
}
```

Localization

The **Localization** library allows you to localize default text content of the DataGrid. The DataGrid component has static text on some features (like group drop area text, pager information text, etc.) that can be changed to other cultures (Arabic, Deutsch, French, etc.).

We have used Resource file (**.resx**) to translate the static text of the DataGrid.

The Resource file is an XML file which contains the strings(key and value pairs) that you want to translate into different language. You can also refer [Localization](#) link to know more about how to configure and use localization in the ASP.Net Core application framework.

- Add **.resx** file to [Resources](#) folder and enter the key value (Locale Keywords) in the **Name** column and the translated string in the **Value** column as follows.

Name | Value (in Deutsch culture)

Grid_Add | Hinzufügen.

Grid_AddFormTitle | Neuen Datensatz hinzufügen.

Grid_AND | UND.

Grid_AutoFit | Diese Spalte automatisch anpassen.

Grid_AutoFitAll | Automatisch alle Spalten anpassen.

Grid_BatchSaveConfirm | Möchten Sie die Änderungen wirklich speichern?.

Grid_BatchSaveLostChanges | Nicht gespeicherte Änderungen gehen verloren. Sind Sie sicher, dass Sie fortfahren wollen?

Grid_Between | Zwischen

Grid_Blanks | Leerzeichen

Grid_Cancel | Stornieren

Grid_CancelButton | Stornieren

Grid_CancelEdit | Möchten Sie die Änderungen wirklich abbrechen?

Grid_ChooseColumns | Spalte auswählen

Grid_ChooseDate | Wählen Sie ein Datum

Grid_ClearButton | klar

Grid_ClearFilter | Filter löschen

Grid_Columnchooser | Säulen

Grid_ConfirmDelete | Möchten Sie den Datensatz wirklich löschen?

Grid_Contains | Enthält

Grid_Copy | Kopieren

Grid_Csvexport | CSV-Export

Grid_CustomFilter | Benutzerdefinierte Filter

Grid_CustomFilterDatePlaceholder | Wählen Sie ein Datum

Grid_CustomFilterPlaceholder | Geben Sie den Wert ein

Grid_DateFilter | Datumsfilter

Grid_DateTimeFilter | DateTime-Filter

Grid_Delete | Löschen

Grid_DeleteOperationAlert | Keine Datensätze zum Löschen ausgewählt

Grid_DeleteRecord | Aufzeichnung löschen

Grid_Edit | Bearbeiten

Grid_EditFormTitle | Details von

Grid_EditOperationAlert | Keine Datensätze zum Bearbeiten ausgewählt

Grid_EditRecord | Datensatz bearbeiten

Grid_EmptyDataSourceError | DataSource darf beim ersten Laden nicht leer sein, da Spalten aus dataSource in AutoGenerate Column Grid generiert werden

Grid_EmptyRecord | Keine Datensätze zum Anzeigen

Grid_EndsWith | Endet mit

Grid_EnterValue | Geben Sie den Wert ein

Grid_Equal | Gleich

Grid_ExcelExport | Excel-Export

Grid_Export | Export

Grid_False | falsch

Grid_FilterBarTitle | Filterbalkenzelle

Grid_FilterButton | Filter

Grid_FilterFalse | Falsch

Grid_FilterMenu | Filter

Grid_FilterTrue | Wahr

Grid_FirstPage | Erste Seite

Grid_GreaterThan | Größer als

Grid_GreaterThanOrEqual | Größer als oder gleich

Grid_Group | Nach dieser Spalte gruppieren

Grid_GroupDisable | Die Gruppierung ist für diese Spalte deaktiviert

Grid_GroupDropArea | Ziehen Sie eine Spaltenüberschrift hierher, um die Spalte zu gruppieren

Grid_InvalidFilterMessage | Ungültige Filterdaten

Grid_Item | Artikel

Grid_Items | Artikel
Grid_LastPage | Letzte Seite
Grid_LessThan | Weniger als
Grid_LessThanOrEqual | Weniger als oder gleich
Grid_MatchCase | Match-Fall
Grid_Matches | Keine Treffer gefunden
Grid_NextPage | Nächste Seite
Grid_NoResult | Keine Treffer gefunden
Grid_NotEqual | Nicht gleich
Grid_NumberFilter | Anzahl Filter
Grid_OKButton | in Ordnung
Grid_OR | ODER
Grid_Pdfexport | PDF-Export
Grid_PreviousPage | Vorherige Seite
Grid_Print | Drucken
Grid_Save | speichern
Grid_SaveButton | speichern
Grid_Search | Suche
Grid_SearchColumns | Spalten durchsuchen
Grid_SelectAll | Wählen Sie Alle
Grid_ShowRowsWhere | Zeilen anzeigen, in denen:
Grid_SortAscending | Aufsteigend sortieren
Grid_SortDescending | Absteigend sortieren
Grid_StartsWith | Beginnt mit
Grid_TextFilter | Textfilter
Grid_True | wahr
Grid_Ungroup | Gruppierung nach dieser Spalte aufheben
Grid_UnGroupButton | Klicken Sie hier, um die Gruppierung aufzuheben
Grid_Update | Aktualisieren
Grid_Wordexport | Word-Export
Pager_All | Alle
Pager_CurrentPageInfo | {0} von {1} Seiten
Pager_FirstPageTooltip | Gehe zur ersten Seite

Pager_LastPageTooltip | Gehe zur letzten Seite

Pager_NextPagerTooltip | Zum nächsten Pager gehen

Pager_NextPageTooltip | Gehe zur nächsten Seite

Pager_PagerAllDropDown | Artikel

Pager_PagerDropDown | Objekte pro Seite

Pager_PreviousPagerTooltip | Zum vorherigen Pager wechseln

Pager_PreviousPageTooltip | Zurück zur letzten Seite

Pager_TotalItemsInfo | ({0} Artikel)

Blazor Server Side

In the following examples, demonstrate how to enable **Localization** for DataGrid in server side Blazor samples.

- Open the **Startup.cs** file and add the below configuration in the **ConfigureServices** function as follows.

CSHARP

```
using Syncfusion.Blazor;
using System.Globalization;
using Microsoft.AspNetCore.Localization;
namespace BlazorApplication
{
    public class Startup
    {
        ....
        ....
        public void ConfigureServices(IServiceCollection services)
        {
            ....
            ....
            services.AddSyncfusionBlazor();
            services.AddLocalization(options => options.ResourcesPath = "Resources");
            services.Configure<RequestLocalizationOptions>(options =>
            {
                // define the list of cultures your app will support
                var supportedCultures = new List<CultureInfo>()
                {
                    new CultureInfo("de")
                };
                // set the default culture
                options.DefaultRequestCulture = new RequestCulture("de");
                options.SupportedCultures = supportedCultures;
                options.SupportedUICultures = supportedCultures;
                options.RequestCultureProviders = new List<IRequestCultureProvider>() {
                    new QueryStringRequestCultureProvider() // Here, You can also use other
                    localization provider
                };
            });
            services.AddSingleton(typeof(ISyncfusionStringLocalizer),
                typeof(SampleLocalizer));
        }
    }
}
```

```
}
}
}
```

Add [UseRequestLocalization\(\)](#) middle-ware in Configure method in **Startup.cs** file to get browser Culture Information.

- Then, write a **class** by inheriting **ISyncfusionStringLocalizer** interface and override the Manager property to get the resource file details from the application end.

CSHARP

```
using Syncfusion.Blazor;
namespace BlazorServer
{
    public class SampleLocalizer : ISyncfusionStringLocalizer
    {
        public string GetText(string key)
        {
            return this.ResourceManager.GetString(key);
        }
        public System.Resources.ResourceManager ResourceManager
        {
            get
            {
                return BlazorServer.Resources.SfResources.ResourceManager;
            }
        }
    }
}
```

BlazorServer denotes the ApplicationNameSpace of your project.

- Finally, Specify the culture for DataGrid using [locale](#) property.

ASPX-CS

```
@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders" AllowPaging="true" Locale="de"
AllowGrouping="true" Height="400">
    <GridPageSettings PageSizes="true"></GridPageSettings>
    <GridGroupSettings ShowDropArea="true"></GridGroupSettings>
    <GridColumns>
        <GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
        TextAlign="TextAlign.Right" Width="120"></GridColumn>
        <GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
        Width="150"></GridColumn>
        <GridColumn Field=@nameof(Order.OrderDate) HeaderText="Order Date"
        Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
        Width="130"></GridColumn>
        <GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
        TextAlign="TextAlign.Right" Width="120"></GridColumn>
    </GridColumns>
```

```

</SfGrid>
@code{
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
    Orders = Enumerable.Range(1, 75).Select(x => new Order()
    {
        OrderID = 1000 + x,
        CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
    })[new Random().Next(5)],
        Freight = 2.1 * x,
        OrderDate = DateTime.Now.AddDays(-x),
    }).ToList();
    }
    public class Order
    {
        public int? OrderID { get; set; }
        public string CustomerID { get; set; }
        public DateTime? OrderDate { get; set; }
        public double? Freight { get; set; }
    }
}

```

Blazor WebAssembly

In the following examples, demonstrate how to enable **Localization** for DataGrid in Client side Blazor samples.

- Open the **Program.cs** file and add the below configuration in the **Main** function as follows.

CSHARP

```

using Syncfusion.Blazor;
using System.Globalization;
namespace ClientApplication
{
    public class Program
    {
        public static async Task Main(string[] args)
        {
            var builder = WebAssemblyHostBuilder.CreateDefault(args);
            builder.RootComponents.Add<App>("app");
            builder.Services.AddTransient(sp => new HttpClient { BaseAddress = new
            Uri(builder.HostEnvironment.BaseAddress) });
            builder.Services.AddSyncfusionBlazor();
            // Register the Syncfusion locale service to customize the SyncfusionBlazor
            component locale culture
            builder.Services.AddSingleton(typeof(ISyncfusionStringLocalizer),
            typeof(SyncfusionLocalizer));
            // Set the default culture of the application
            CultureInfo.DefaultThreadCurrentCulture = new CultureInfo("de");
            CultureInfo.DefaultThreadCurrentUICulture = new CultureInfo("de");
            await builder.Build().RunAsync();
        }
    }
}

```

- Then, create a `~/Shared/SyncfusionLocalizer.cs` file and implement `ISyncfusionStringLocalizer` interface to the class and override the `ResourceManager` property to get the resource file details from the application end.

CSHARP

```
using Syncfusion.Blazor;
public class SyncfusionLocalizer : ISyncfusionStringLocalizer
{
    // To get the locale key from mapped resources file
    public string GetText(string key)
    {
        return this.ResourceManager.GetString(key);
    }
    // To access the resource file and get the exact value for locale key
    public System.Resources.ResourceManager ResourceManager
    {
        get
        {
            // Replace the ApplicationNamespace with your application name.
            return ClientApplication.Resources.SfResources.ResourceManager;
        }
    }
}
```

ClientApplication denotes the ApplicationNameSpace of your project.

- Now, Specify the culture for DataGrid using [locale](#) property.

ASPX-CS

```
@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders" AllowPaging="true" Locale="de"
AllowGrouping="true" Height="400">
<GridPageSettings PageSizes="true"></GridPageSettings>
<GridGroupSettings ShowDropArea="true"></GridGroupSettings>
<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumns>
</SfGrid>
@code{
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
}
```

```

Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
    OrderID = 1000 + x,
    CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
    })[new Random().Next(5)],
    Freight = 2.1 * x,
    OrderDate = DateTime.Now.AddDays(-x),
}).ToList();
}
public class Order
{
    public int? OrderID { get; set; }
    public string CustomerID { get; set; }
    public DateTime? OrderDate { get; set; }
    public double? Freight { get; set; }
}
}

```

Internationalization

- The Syncfusion Blazor UI components are specific to the **American English (en-US)** culture by default. It utilizes the **Blazor Internationalization** package to parse and format the number and date objects based on the chosen culture.
- Suppose, if you want to change any specific culture, then add the corresponding culture resource (.resx) file by using the below reference.

[Changing culture and Adding Resx file in the application](#)

Right to left (RTL)

RTL provides an option to switch the text direction and layout of the DataGrid component from right to left. It improves the user experiences and accessibility for users who use right-to-left languages (Arabic, Farsi, Urdu, etc.). In the Below sample **EnableRtl** property is used to enable RTL in the DataGrid.

ASPX-CS

```

@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders" AllowSorting="true" EnableRtl="true"
AllowPaging="true">
    <GridColumns>
        <GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
        TextAlign="TextAlign.Right" Width="120"></GridColumn>
        <GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
        Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
        Width="130"></GridColumn>
        <GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
        TextAlign="TextAlign.Right" Width="120"></GridColumn>
    </GridColumns>
</SfGrid>
@code{
    public List<Order> Orders { get; set; }
    protected override void OnInitialized()
    {
        Orders = Enumerable.Range(1, 75).Select(x => new Order()
        {

```

```

    OrderID = 1000 + x,
    CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
    })[new Random().Next(5)],
    Freight = 2.1 * x,
    OrderDate = DateTime.Now.AddDays(-x),
  }).ToList();
}
public class Order
{
    public int? OrderID { get; set; }
    public string CustomerID { get; set; }
    public DateTime? OrderDate { get; set; }
    public double? Freight { get; set; }
}
}

```

You can refer to our [Blazor DataGrid](#) feature tour page for its groundbreaking feature representations. You can also explore our [Blazor DataGrid example](#) to understand how to present and manipulate data.

Toolbar in Blazor DataGrid Component

The DataGrid provides tool bar support to handle datagrid actions. The [Toolbar](#) property accepts either the collection of built-in tool bar items and [ItemModel](#)

Built-in Tool Bar Item

The DataGrid provides tool bar support to handle datagrid actions. The [Toolbar](#) as a collection of built-in items. It renders the button with icon and text.

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

Built-in Toolbar Items	Actions
Add	Adds a new record.
Edit	Edits the selected record.
Update	Updates the edited record.
Delete	Deletes the selected record.
Cancel	Cancels the edit state.
Search	Searches the records by the given key.
Print	Prints the datagrid.
ExcelExport	Exports the datagrid to Excel.
PdfExport	Exports the datagrid to PDF.
CsvExport	Exports the datagrid to CSV.

CSHARP

```

@using Syncfusion.Blazor.Grids
@{
    var Tool = (new List<string>() { "Search", "Print" });
}

```

```

<SfGrid DataSource="@Orders" AllowPaging="true" Height="200" Toolbar=@Tool>
<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumns>
</SfGrid>
@code{
public List<Order>Orders{ get; set; }
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
OrderDate = DateTime.Now.AddDays(-x),
}).ToList();
}
public class Order
{
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
}

```

The following screenshots represent a datagrid with Built-in toolbar,

Print		Search		
Order ID	Customer Name	Order Date	Freight	
1001	ALFKI	8/13/2019	\$2.10	
1002	ANANTR	8/12/2019	\$4.20	
1003	ALFKI	8/11/2019	\$6.30	
1004	ANTON	8/10/2019	\$8.40	
1005	ANANTR	8/9/2019	\$10.50	
1006	ALFKI	8/8/2019	\$12.60	
K < 1 2 3 4 5 6 7 > X				1 of 7 pages (75 items)

Custom Toolbar Items

Custom tool bar items can be added by defining the [Toolbar](#). Custom toolbar items can be achieved by using Expand all and Collapse all functions.

CSHARP

```
@using Syncfusion.Blazor
@using Syncfusion.Blazor.Grids
@using Syncfusion.Blazor.Navigations
<SfGrid DataSource="@Orders" @ref="Grid" AllowGrouping="true"
AllowPaging="true" Height="200" Toolbar="ToolbarItems">
<GridEvents OnToolbarClick="ToolbarClickHandler"
TValue="Order"></GridEvents>
<GridGroupSettings Columns=@Tool></GridGroupSettings>
<GridColumn>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumn>
</SfGrid>
<style>
.e-expand::before {
content: '\e82e';
}
.e-collapse::before {
content: '\e834';
}
</style>
@code{
SfGrid<Order> Grid;
public List<Order>Orders{ get; set; }
private List<ItemModel> ToolbarItems = new List<ItemModel>();
private string[] Tool = (new string[] { "OrderID" });
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
OrderDate = DateTime.Now.AddDays(-x),
}).ToList();
ToolbarItems.Add(new ItemModel() { Text = "Expand all", TooltipText =
"Expand all", PrefixIcon = "e-expand" });
ToolbarItems.Add(new ItemModel() { Text = "Collapse all", TooltipText =
"Collapse all", PrefixIcon = "e-collapse", Align =
(Syncfusion.Blazor.Navigations.ItemAlign.Right) });
}
public class Order
{

```

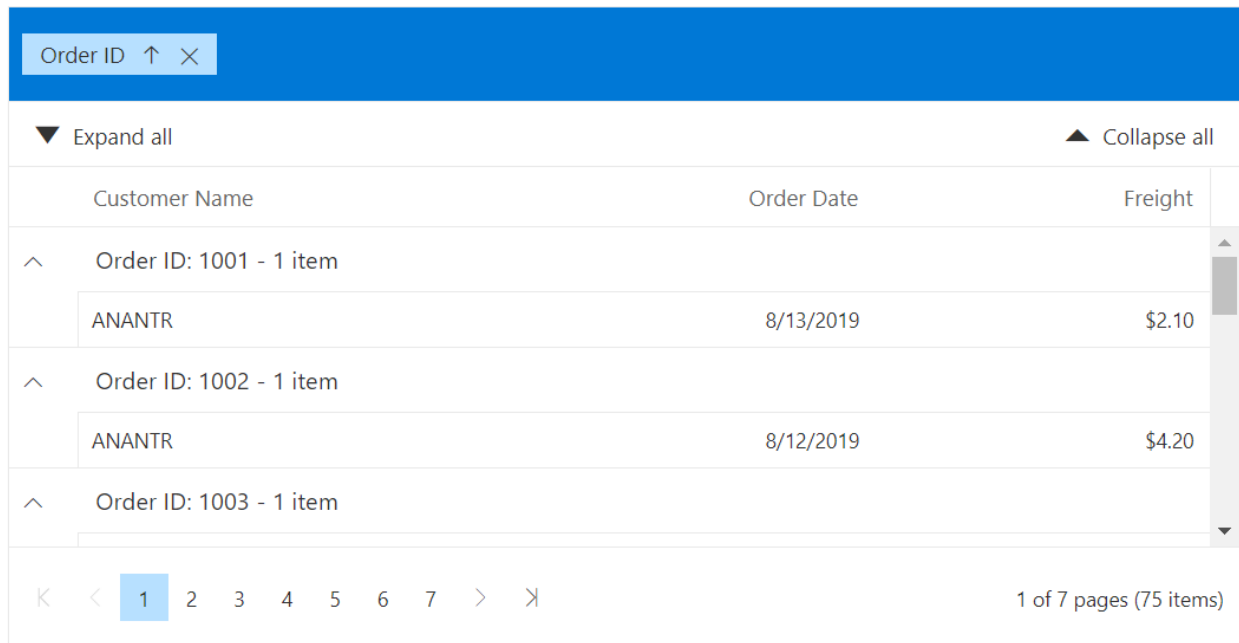


```

public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
public async Task
ToolbarClickHandler(Syncfusion.Blazor.Navigations.ClickEventArgs args)
{
    if (args.Item.Text == "Expand all")
    {
        await this.Grid.GroupExpandAll();
    }
    if (args.Item.Text == "Collapse all")
    {
        await this.Grid.GroupCollapseAll();
    }
}
}

```

The following screenshots represent a datagrid with Custom toolbar items,



Customer Name	Order Date	Freight
▼ Expand all ▲ Collapse all		
^ Order ID: 1001 - 1 item		
ANANTR	8/13/2019	\$2.10
^ Order ID: 1002 - 1 item		
ANANTR	8/12/2019	\$4.20
^ Order ID: 1003 - 1 item		
K < 1 2 3 4 5 6 7 > X		
1 of 7 pages (75 items)		

Built-in and Custom Items in Toolbar

DataGrid have an option to use both built-in and custom tool bar items at same time.

In the below example, **Add**, **Edit**, **Delete**, **Update**, **Cancel** are built-in toolbar items and **Click** is custom toolbar item.

ASPX-CS

```

@using Syncfusion.Blazor.Grids
@using Syncfusion.Blazor.Navigations
<SfGrid DataSource="@Orders" AllowPaging="true" Height="200"
Toolbar="Toolbaritems">

```

```

<GridEvents OnToolbarClick="ToolbarClickHandler"
TValue="Order"></GridEvents>
<GridEditSettings AllowAdding="true" AllowEditing="true"
AllowDeleting="true" Mode="EditMode.Normal"></GridEditSettings>
<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
TextAlign="TextAlign.Right" IsPrimaryKey="true" ValidationRules="new
ValidationRules() { Required = true }" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
ValidationRules="new ValidationRules() { Required = true }"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" EditType="EditType.NumericEdit"
ValidationRules="new ValidationRules() { Required = true }"
Width="120"></GridColumn>
</GridColumns>
</SfGrid>
<style>
.e-click::before {
content: '\e525';
}
</style>
@code{
public List<Order> Orders { get; set; }
private List<Object> ToolbarItems = new List<Object>() { "Add", "Edit",
"Delete", "Update", "Cancel", new ItemModel() { Text = "Click", TooltipText
= "Click", PrefixIcon = "e-click", Id = "Click" } };
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
OrderDate = DateTime.Now.AddDays(-x),
}).ToList();
}
public class Order
{
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
public void ToolbarClickHandler(Syncfusion.Blazor.Navigations.ClickEventArgs
args)
{
if (args.Item.Id == "Click")
{
//You can customized your code here....
}
}
}

```

The following screenshots represent a datagrid with Built-in and custom items in toolbar,

Add Edit Delete Update Cancel Click				
Order ID	Customer Name	Order Date	Freight	
1001	BLONP	8/13/2019	\$2.10	
1002	ALFKI	8/12/2019	\$4.20	
1003	ANTON	8/11/2019	\$6.30	
1004	ANANTR	8/10/2019	\$8.40	
1005	ANTON	8/9/2019	\$10.50	
1006	BOLID	8/8/2019	\$12.60	

K < 1 2 3 4 5 6 7 > X
 1 of 7 pages (75 items)

Custom Toolbar

Custom tool bar items can be added by defining the [Toolbar Template](#) .Custom toolbar can be placed inside datagrid using [ToolbarTemplate`] as below.

CSHARP

```
@using Syncfusion.Blazor.Grids
@using Syncfusion.Blazor.Navigations
<SfGrid DataSource="@Orders" AllowPaging="true" Height="200" @ref="Grid"
AllowGrouping="true">
<GridTemplates>
<ToolbarTemplate>
<SfToolbar>
<ToolbarEvents Clicked="ToolbarClickHandler"></ToolbarEvents>
<ToolbarItems>
<ToolbarItem Type="@ItemType.Button" PrefixIcon="e-icons e-collapse"
Id="collapseall" TooltipText="Collapse"></ToolbarItem>
</ToolbarItems>
</SfToolbar>
</ToolbarTemplate>
</GridTemplates>
<GridEvents OnToolbarClick="ToolbarClickHandler"
TValue="Order"></GridEvents>
<GridGroupSettings Columns=@Tool></GridGroupSettings>
<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order D ate"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumns>
```

```
</SfGrid>
<style>
.e-collapse::before {
content: '\e834';
}
</style>
@code{
SfGrid<Order> Grid;
public List<Order> Orders { get; set; }
private string[] Tool = (new string[] { "OrderID" });
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
OrderDate = DateTime.Now.AddDays(-x),
}).ToList();
}
public class Order
{
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
public async Task
ToolbarClickHandler(Syncfusion.Blazor.Navigations.ClickEventArgs args)
{
await this.Grid.GroupCollapseAll();
}
}
```

The following screenshots represent a datagrid with Custom toolbar,

Order ID ↑ ×		
▲		
Customer Name	Order Date	Freight
^ Order ID: 1001 - 1 item		
BLONP	8/13/2019	\$2.10
^ Order ID: 1002 - 1 item		
BLONP	8/12/2019	\$4.20
^ Order ID: 1003 - 1 item		
K < 1 2 3 4 5 6 7 > X		1 of 7 pages (75 items)

The custom toolbar can be placed inside datagrid using Toolbar template component. The contents in the DataGrid can be collapsed by clicking the Collapse icon button.

Custom Toolbar with dropdown list

Drop down list can be added by using dropdownlist in the tool bar section.

CSHARP

```
@using Syncfusion.Blazor.Grids
@using Syncfusion.Blazor.Navigations
@using Syncfusion.Blazor.DropDowns
<SfGrid DataSource="@Orders" AllowPaging="true" Height="200" @ref="Grid">
<GridEvents TValue="Order"></GridEvents>
<SfToolbar>
<ToolbarItems>
<ToolbarItem Type="ItemType.Input">
<Template>
<SfDropDownList TValue="string" TItem="Select" DataSource=@LocalData
Width="200">
<DropDownListFieldSettings Text="text" Value="text">
</DropDownListFieldSettings>
<DropDownListEvents TValue="string" TItem="Select" ValueChange="OnChange">
</DropDownListEvents>
</SfDropDownList>
</Template>
</ToolbarItem>
</ToolbarItems>
</SfToolbar>
<GridColumn>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>
```

```

<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumn>
</SfGrid>
@code{
SfGrid<Order> Grid;
public List<Order> Orders { get; set; }
public class Select
{
public string text { get; set; }
}
List<Select> LocalData = new List<Select>
{
new Select() { text = "0"},
new Select() { text = "1"},
new Select() { text = "2"},
new Select() { text = "3"},
new Select() { text = "4"},
new Select() { text = "5"},
new Select() { text = "6"},
new Select() { text = "7"},
new Select() { text = "8"},
new Select() { text = "9"},
};
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 10).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
OrderDate = DateTime.Now.AddDays(-x),
}).ToList();
}
public async Task
OnChange(Syncfusion.Blazor.DropDowns.ChangeEventArgs<string, Select> args)
{
await this.Grid.SelectRow(int.Parse(args.Value));
}
public class Order
{
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
}

```

The following screenshots represent a datagrid with Custom toolbar dropdown list,

0 ▾

Order ID	Customer Name	Order Date	Freight
1001	ALFKI	8/13/2019	\$2.10
1002	ALFKI	8/12/2019	\$4.20
1003	BOLID	8/11/2019	\$6.30
1004	ANTON	8/10/2019	\$8.40
1005	BOLID	8/9/2019	\$10.50
1006	BLONP	8/8/2019	\$12.60

⏪

<

1

>

⏩

1 of 1 pages (10 items)

Enable/Disable Toolbar Items

You can enable / disable tool bar items by using the [EnableToolbarItems method](#).

CSHARP

```
@using Syncfusion.Blazor.Grids
@using Syncfusion.Blazor.Buttons
<div>
<div style="float:left;">
<SfButton id="Enable" Content="Enable" @onclick="enable"></SfButton>
</div>
<div style="padding-left: 90px">
<SfButton id="Disable" Content="Disable" @onclick="disable"></SfButton>
</div>
</div>
@{
var Tool = (new string[] { "Expand", "Collapse" });
}
<SfGrid id="Grid" DataSource="@Orders" AllowPaging="true" Height="200"
@ref="Grid" AllowGrouping="true" Toolbar=@Tool>
<GridGroupSettings Columns=@GroupCol></GridGroupSettings>
<GridEvents OnToolbarClick="ToolbarClickHandler"
TValue="Order"></GridEvents>
<GridColumn>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumn>
</SfGrid>
<style>
.e-expand::before {
```

```

content: '\e82e';
}
.e-collapse::before {
content: '\e834';
}
</style>
@code{
SfGrid<Order> Grid;
public List<Order> Orders { get; set; }
private string[] GroupCol = (new string[] { "OrderID" });
public async Task enable()
{
await this.Grid.EnableToolbarItems(new List<string>() { "Grid_Expand",
"Grid_Collapse" }, true);
}
public async Task disable()
{
await this.Grid.EnableToolbarItems(new List<string>() { "Grid_Expand",
"Grid_Collapse" }, false);
}
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
OrderDate = DateTime.Now.AddDays(-x),
}).ToList();
}
public class Order
{
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
public async Task
ToolbarClickHandler(Syncfusion.Blazor.Navigations.ClickEventArgs args)
{
if (args.Item.Text == "Expand")
{
await this.Grid.GroupExpandAll();
}
if (args.Item.Text == "Collapse")
{
await this.Grid.GroupCollapseAll();
}
}
}

```

The following screenshots represent a datagrid with Enable/disable toolbar items,

Enable Disable		
Order ID ↑ ×		
Expand Collapse		
Customer Name	Order Date	Freight
^ Order ID: 1001 - 1 item		
ANTON	8/13/2019	\$2.10
^ Order ID: 1002 - 1 item		
ANTON	8/12/2019	\$4.20
^ Order ID: 1003 - 1 item		
K < 1 2 3 4 5 6 7 > X		
1 of 7 pages (75 items)		

Customize Toolbar Text

You can able to customize the toolbar text by using the [ItemModel](#) properties.

CSHARP

```
@using Syncfusion.Blazor.Grids
@using Syncfusion.Blazor.Navigations
<SfGrid ID="Grid" DataSource="@Orders" AllowPaging="true"
Toolbar=@ToolbarItems>
<GridEditSettings AllowAdding="true" AllowDeleting="true"
AllowEditing="true"></GridEditSettings>
<GridColumn>
<GridColumn Field=@nameof(Order.OrderID) IsPrimaryKey="true"
HeaderText="Order ID" TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumn>
</SfGrid>
@code{
public List<Order> Orders { get; set; }
private List<object> ToolbarItems = new List<object>() {
new ItemModel() { Text = "Add Record", PrefixIcon = "e-add", Id =
"Grid_add"},//Here Grid is SfGrid ID
new ItemModel(){ Text = "Edit Record", PrefixIcon= "e-edit",
Id="Grid_edit"},
new ItemModel(){ Text = "Delete Record", PrefixIcon= "e-delete",
Id="Grid_delete"},
}
```

```

new ItemModel() { Text = "Update Record", PrefixIcon= "e-update",
Id="Grid_update"},
new ItemModel() { Text = "Cancel Changes", PrefixIcon= "e-cancel",
Id="Grid_cancel"}
};
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
OrderDate = DateTime.Now.AddDays(-x),
}).ToList();
}
public class Order
{
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
}

```

The following screenshots represent a datagrid by customizing toolbar text.

+ Add Record ✎ Edit Record 🗑 Delete Record 📄 Update Record ✕ Cancel Changes				
Order ID	Customer Name	Order Date	Freight	
1001	BOLID	5/22/2021	\$2.10	
1002	BOLID	5/21/2021	\$4.20	
1003	ANANTR	5/20/2021	\$6.30	
1004	BLONP	5/19/2021	\$8.40	
1005	ANTON	5/18/2021	\$10.50	
1006	BOLID	5/17/2021	\$12.60	
1007	BLONP	5/16/2021	\$14.70	
1008	BLONP	5/15/2021	\$16.80	
1009	BLONP	5/14/2021	\$18.90	
1010	ANANTR	5/13/2021	\$21.00	
1011	BLONP	5/12/2021	\$23.10	
1012	ANTON	5/11/2021	\$25.20	

<< < 1 2 3 4 5 6 7 > >>
1 of 7 pages (75 items)

Customize toolbar styles

You can able to customize the toolbar styles by using the toolbar class (`.e-toolbar-item .e-tbar-btn`)

CSHARP

```

@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders" AllowPaging="true" Toolbar="@ (new
List<string>() { "Add", "Edit", "Delete", "Cancel", "Update" })"
Height="315">

```

```

<GridEditSettings AllowAdding="true" AllowEditing="true"
AllowDeleting="true" Mode="EditMode.Normal"></GridEditSettings>
<GridColumn>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
IsPrimaryKey="true" ValidationRules="@ (new ValidationRules{ Required=true})"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
ValidationRules="@ (new ValidationRules{ Required=true})"
Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
EditType="EditType.DatePickerEdit" Format="d" TextAlign="TextAlign.Right"
Width="130" Type="ColumnType.Date"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" EditType="EditType.NumericEdit"
Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.ShipCountry) HeaderText="Ship Country"
EditType="EditType.DropDownEdit" Width="150"></GridColumn>
</GridColumn>
</SfGrid>
<style>
.e-toolbar-item .e-tbar-btn {
background-color: powderblue;
}
</style>
@code{
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
OrderDate = DateTime.Now.AddDays(-x),
ShipCountry = (new string[] { "USA", "UK", "CHINA", "RUSSIA", "INDIA" })[new
Random().Next(5)]
}).ToList();
}
public class Order
{
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
public string ShipCountry { get; set; }
}
}

```

The following screenshots represent a datagrid by customizing toolbar styles.

<div> <div>+ Add</div> <div>Edit</div> <div>Delete</div> <div>Cancel</div> <div>Update</div> </div>					
Order ID	Customer Name	Order Date	Freight	Ship Country	
1001	BLONP	5/22/2021	\$2.10	RUSSIA	
1002	ANANTR	5/21/2021	\$4.20	CHINA	
1003	BLONP	5/20/2021	\$6.30	CHINA	
1004	ANTON	5/19/2021	\$8.40	USA	
1005	ANANTR	5/18/2021	\$10.50	RUSSIA	
1006	ANTON	5/17/2021	\$12.60	USA	
1007	ANTON	5/16/2021	\$14.70	CHINA	
1008	BOLID	5/15/2021	\$16.80	USA	
1009	BOLID	5/14/2021	\$18.90	RUSSIA	

« < 1 2 3 4 5 6 7 > »

1 of 7 pages (75 items)

You can refer to our [Blazor DataGrid](#) feature tour page for its groundbreaking feature representations. You can also explore our [Blazor DataGrid example](#) to understand how to present and manipulate data.

<!-- markdownlint-disable MD033 -->

Pdf Export in Blazor DataGrid Component

PDF export allows exporting DataGrid data to PDF document. You need to use the

PdfExport method for exporting. To enable PDF export in the datagrid, set the [AllowPdfExport](#) as true.

ASPX-CS

```
@using Syncfusion.Blazor.Grids
<SfGrid ID="Grid" @ref="DefaultGrid" DataSource="@Orders" Toolbar="@ (new
List<string>() { "PdfExport" })" AllowPdfExport="true" AllowPaging="true">
<GridEvents OnToolbarClick="ToolbarClickHandler"
TValue="Order"></GridEvents>
<GridColumn>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumn>
</SfGrid>
@code{
private SfGrid<Order> DefaultGrid;
public List<Order> Orders { get; set; }
public async Task
ToolbarClickHandler(Syncfusion.Blazor.Navigations.ClickEventArgs args)
{
if (args.Item.Id == "Grid_pdfexport") //Id is combination of Grid's ID and
itemname
{
await this.DefaultGrid.PdfExport();
}
}
}
protected override void OnInitialized()
```

```
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
OrderDate = DateTime.Now.AddDays(-x),
}).ToList();
}
public class Order
{
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
}
```

<!-- Multiple exporting

PDF export provides an option for exporting multiple grids to same file. In this exported document, each datagrid will be exported to new page of document in same file.

This is demonstrated in the below sample code block,

ASPX-CS

-->

ASPX-CS

```
@using Syncfusion.Blazor.Grids
<SfGrid ID="Grid" @ref="DefaultGrid" DataSource="@Orders" Toolbar="@ (new
List<string>() { "PdfExport" })" AllowPdfExport="true" AllowPaging="true">
<GridEvents OnToolbarClick="ToolbarClickHandler"
TValue="Order"></GridEvents>
<GridColumn>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumn>
</SfGrid>
@code{
private SfGrid<Order> DefaultGrid;
public List<Order> Orders { get; set; }
public async Task
ToolbarClickHandler(Syncfusion.Blazor.Navigations.ClickEventArgs args)
```

```

{
    if (args.Item.Id == "Grid_pdfexport") //Id is combination of Grid's ID and
        itemname
    {
        PdfExportProperties ExportProperties = new PdfExportProperties();
        ExportProperties.FileName = "test.pdf";
        await this.DefaultGrid.PdfExport(ExportProperties);
    }
}
protected override void OnInitialized()
{
    Orders = Enumerable.Range(1, 75).Select(x => new Order()
    {
        OrderID = 1000 + x,
        CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
    })[new Random().Next(5)],
        Freight = 2.1 * x,
        OrderDate = DateTime.Now.AddDays(-x),
    }).ToList();
}
public class Order
{
    public int? OrderID { get; set; }
    public string CustomerID { get; set; }
    public DateTime? OrderDate { get; set; }
    public double? Freight { get; set; }
}
}

```

<!-- Default fonts for pdf exporting

By default, datagrid uses **Helvetica** font in the exported document. You can change the default font by using [Theme](#) property of [PdfExportProperties](#).

The available fonts are,

- Helvetica
- TimesRoman
- Courier
- Symbol
- ZapfDingbats

The following sample code demonstrates changing the default font value on exported document,

ASPX-CS

-->

ASPX-CS

-->

ASPX-CS

```

@using Syncfusion.Blazor.Grids
<SfGrid ID="Grid" @ref="DefaultGrid" DataSource="@Orders" Toolbar="@ (new
List<string>() { "PdfExport" })" AllowPdfExport="true" AllowPaging="true">
<GridEvents OnToolbarClick="ToolbarClickHandler"
TValue="Order"></GridEvents>
<GridColumn>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumn>
</SfGrid>
@code{
private SfGrid<Order> DefaultGrid;
public List<Order> Orders { get; set; }
public List<PdfHeaderFooterContent> HeaderContent = new
List<PdfHeaderFooterContent>
{
new PdfHeaderFooterContent() { Type = ContentType.Text, Value = "Northwind
Traders", Position = new PdfPosition() { X = 0, Y = 50 }, Style = new
PdfContentStyle() { TextBrushColor = "#000000", FontSize = 13 } }
};
public async Task
ToolbarClickHandler(Syncfusion.Blazor.Navigations.ClickEventArgs args)
{
if (args.Item.Id == "Grid_pdfexport") //Id is combination of Grid's ID and
itemname
{
PdfExportProperties ExportProperties = new PdfExportProperties();
PdfHeader Header = new PdfHeader()
{
FromTop = 0,
Height = 130,
Contents = HeaderContent
};
ExportProperties.Header = Header;
await this.DefaultGrid.PdfExport(ExportProperties);
}
}
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],

```

```

Freight = 2.1 * x,
OrderDate = DateTime.Now.AddDays(-x),
}).ToList();
}
public class Order
{
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
}

```

How to draw a line in header/footer

You can add line either in the Header or Footer area of the exported PDF document using [Header](#) and [Footer](#) properties of the [PdfExportProperties](#) class.

Supported line styles are,

- Dash
- Dot
- DashDot
- DashDotDot
- Solid

The following sample code demonstrates adding line in the Header section of the exported document,

ASPX-CS

```

@using Syncfusion.Blazor.Grids
<SfGrid ID="Grid" @ref="DefaultGrid" DataSource="@Orders" Toolbar="@ (new
List<string>() { "PdfExport" })" AllowPdfExport="true" AllowPaging="true">
<GridEvents OnToolbarClick="ToolbarClickHandler"
TValue="Order"></GridEvents>
<GridColumn>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumn>
</SfGrid>
@code{
private SfGrid<Order> DefaultGrid;
public List<Order> Orders { get; set; }
public List<PdfHeaderFooterContent> HeaderContent = new
List<PdfHeaderFooterContent>
{
new PdfHeaderFooterContent() { Type = ContentType.Line, Points = new
PdfPoints() { X1 = 0, Y1 = 4, X2 = 685, Y2 = 4 }, Style = new
PdfContentStyle() { PenColor = "#000080", DashStyle = PdfDashStyle.Solid } }
}
}

```



```

};
public async Task
ToolbarClickHandler(Syncfusion.Blazor.Navigations.ClickEventArgs args)
{
    if (args.Item.Id == "Grid_pdfexport") //Id is combination of Grid's ID and
        itemname
    {
        PdfExportProperties ExportProperties = new PdfExportProperties();
        PdfHeader Header = new PdfHeader()
        {
            FromTop = 0,
            Height = 130,
            Contents = HeaderContent
        };
        ExportProperties.Header = Header;
        await this.DefaultGrid.PdfExport(ExportProperties);
    }
}
protected override void OnInitialized()
{
    Orders = Enumerable.Range(1, 75).Select(x => new Order()
    {
        OrderID = 1000 + x,
        CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
        })[new Random().Next(5)],
        Freight = 2.1 * x,
        OrderDate = DateTime.Now.AddDays(-x),
    }).ToList();
}
public class Order
{
    public int? OrderID { get; set; }
    public string CustomerID { get; set; }
    public DateTime? OrderDate { get; set; }
    public double? Freight { get; set; }
}
}

```

How to add repeat headers in PDF Export

You can add headers for every page of PDF exported document by enabling [IsRepeatHeader](#) property of the [PdfExportProperties](#) class.

ASPX-CS

```

@using Syncfusion.Blazor.Grids
<SfGrid ID="Grid" @ref="DefaultGrid" DataSource="@Orders" Toolbar="@ (new
List<string>() { "PdfExport" })" AllowPdfExport="true" AllowPaging="true">
<GridEvents OnToolbarClick="ToolbarClickHandler"
TValue="Order"></GridEvents>
<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>

```

```

<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumns>
</SfGrid>
@code{
private SfGrid<Order> DefaultGrid;
public List<Order> Orders { get; set; }
public async Task
ToolbarClickHandler(Syncfusion.Blazor.Navigations.ClickEventArgs args)
{
if (args.Item.Id == "Grid_pdfexport") //Id is combination of Grid's ID and
itemname
{
PdfExportProperties ExportProperties = new PdfExportProperties();
ExportProperties.IsRepeatHeader = true;
await this.DefaultGrid.PdfExport(ExportProperties);
}
}
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
OrderDate = DateTime.Now.AddDays(-x),
}).ToList();
}
public class Order
{
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
}

```

How to export the Grid with specific columns

You can export the PDF grid with specific columns instead of all columns which are defined in the Grid definition. To achieve this scenario by using [Columns](#) property of the [PdfExportProperties](#) class.

ASPX-CS

```

@using Syncfusion.Blazor.Grids
<SfGrid ID="Grid" @ref="DefaultGrid" DataSource="@Orders" Toolbar="@ (new
List<string>() { "PdfExport" })" AllowPdfExport="true" AllowPaging="true">
<GridEvents OnToolbarClick="ToolbarClickHandler"
TValue="Order"></GridEvents>
<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
TextAlign="TextAlign.Right" Width="120"></GridColumn>

```

```

<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumn>
</SfGrid>
@code{
private SfGrid<Order> DefaultGrid;
public List<Order> Orders { get; set; }
public async Task
ToolbarClickHandler(Syncfusion.Blazor.Navigations.ClickEventArgs args)
{
if (args.Item.Id == "Grid_pdfexport") //Id is combination of Grid's ID and
itemname
{
PdfExportProperties ExportProperties = new PdfExportProperties();
List<GridColumn> ExportColumns = new List<GridColumn>();
#pragma warning disable BL0005
ExportColumns.Add(new GridColumn() { Field = "CustomerID", HeaderText =
"Customer Name", Width = "100" });
ExportColumns.Add(new GridColumn() { Field = "OrderDate", HeaderText =
"Date", Width = "120", Format = "d" });
ExportColumns.Add(new GridColumn() { Field = "Freight", HeaderText =
"Freight", Width = "120", Format = "C2", TextAlign = TextAlign.Right });
#pragma warning restore BL0005
ExportProperties.Columns = ExportColumns;
await this.DefaultGrid.PdfExport(ExportProperties);
}
}
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
OrderDate = DateTime.Now.AddDays(-x),
}).ToList();
}
public class Order
{
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
}

```

Add page number in header/footer

You can add page number either in Header or Footer area of exported PDF document using [Header](#) and [Footer](#) properties of the [PdfExportProperties](#) class.

Supported page number types are,

- LowerLatin - a, b, c,
- UpperLatin - A, B, C,
- LowerRoman - i, ii, iii,
- UpperRoman - I, II, III,
- Number - 1,2,3.

The following sample code demonstrates adding page number in the Header section of the exported document,

ASPX-CS

```
@using Syncfusion.Blazor.Grids
<SfGrid ID="Grid" @ref="DefaultGrid" DataSource="@Orders" Toolbar="@ (new
List<string>() { "PdfExport" })" AllowPdfExport="true" AllowPaging="true">
<GridEvents OnToolbarClick="ToolbarClickHandler"
TValue="Order"></GridEvents>
<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumns>
</SfGrid>
@code{
private SfGrid<Order> DefaultGrid;
public List<Order> Orders { get; set; }
public List<PdfHeaderFooterContent> HeaderContent = new
List<PdfHeaderFooterContent>
{
new PdfHeaderFooterContent() { Type = ContentType.PageNumber, PageNumberType
= PdfPageNumberType.Arabic, Position = new PdfPosition() { X = 0, Y = 25 },
Style = new PdfContentStyle() { TextBrushColor = "#0000ff", FontSize = 12,
HAlign = PdfHorizontalAlign.Center } }
};
public async Task
ToolbarClickHandler(Syncfusion.Blazor.Navigations.ClickEventArgs args)
{
if (args.Item.Id == "Grid_pdfexport") //Id is combination of Grid's ID and
itemname
{
PdfExportProperties ExportProperties = new PdfExportProperties();
PdfHeader Header = new PdfHeader()
{
FromTop = 0,
Height = 130,
Contents = HeaderContent
};
ExportProperties.Header = Header;
```

```

await this.DefaultGrid.PdfExport(ExportProperties);
}
}
protected override void OnInitialized()
{
    Orders = Enumerable.Range(1, 75).Select(x => new Order()
    {
        OrderID = 1000 + x,
        CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
    })[new Random().Next(5)],
        Freight = 2.1 * x,
        OrderDate = DateTime.Now.AddDays(-x),
    }).ToList();
}
public class Order
{
    public int? OrderID { get; set; }
    public string CustomerID { get; set; }
    public DateTime? OrderDate { get; set; }
    public double? Freight { get; set; }
}
}

```

Insert an image in header/footer

Image (Base64 string) can be added in header/footer area of the exported PDF document using [Header](#) and [Footer](#) properties of the [PdfExportProperties](#) class.

The following sample code demonstrates inserting image in the Header section of the exported document,

ASPX-CS

```

@using Syncfusion.Blazor.Grids
<SfGrid ID="Grid" @ref="DefaultGrid" DataSource="@Orders" Toolbar="@ (new
List<string>() { "PdfExport" })" AllowPdfExport="true" AllowPaging="true">
<GridEvents OnToolbarClick="ToolbarClickHandler"
TValue="Order"></GridEvents>
<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumns>
</SfGrid>
@code{
private SfGrid<Order> DefaultGrid;
public List<Order> Orders { get; set; }
public List<PdfHeaderFooterContent> HeaderContent = new
List<PdfHeaderFooterContent>
{

```

1642

```

OsaWFL/0Me50UuUt+lCm9KGNbJHcl2hQAC6FWAAAAAABEGWfWGVHlkprmkvkQZYXAZUeWSmuaSaL
xDOkfJ4fsqUjABciAAAAD1UbxxI8phddDynqo3jiR5TC66GF1GUOmJzzxs4Ybu52j9ZToYc88bOG
G7udo/WUrl1i947gSp9SGoAAsREAAABbbIW3i3FzumohlSS22QtvFuLndNRD0davl19vk2w7ZJuNG
9eV5czVxCJCW8aN68ry5mriESHwLpZ/MV4IfTujngk4qAAVo7oAAB7rd3xUzlsHWNLClerd3xUz1
sHWNLCn0HoT3UvFCmdKu8j4KAAXcqgAAAAAAGywuAyo8s1Nc0l8iDLC4DKjyyU1zSTReIZxT5PD
9lSkYALkQAAAAeqjeOJHlMLroeU9VG8cSPKYXXQwuoyh0xOeeNnDDd300frKdDDnnjZww3dztH6y
lesXvHcCVPqQ1AAFiIgAAALbZC28W4ud01EMqSW2yFt4txc7pqIZzrV8Mvt8m2HbJNxo3ryvLmau
IRIS3jRvXleXMlcQiQ+BdLP5ivBD6d0c8EnFQACTHdAAAPdbu+KmcTg6xpYUrlbu+KmcTg6xpYU+
g9Ce6l4oUzpV3kfBQAC7lUAAAAAABruItUq+rWjW7WYk1Dk40SHEc6Wejh52ORyZlVFTwp5DYjV
cV7l17AsuYuaakI09CgRYUNYMJ6NcqvejEXOu5uZ85siR6vTI17jC3XaSONq7h19uuLpcP3Y2ruH
X264ulw/dmC22FE0OqnSoY22FE0OqnSoZlMi0fXmhpviM7tXcOvt1xdLh+7Gldw6+3XF0uH7swW2
womh1U6VDG2womh1U6VDGRaPrzQXxGd2ruHX264ulw/dn0lsmPD2XmYUdk9cKvhPa9uebh5s6LnT
/AA/Qa9tsKJodVOLQz+5fKsokWPDhJZ9URXvRuf4zD3M65jGRaPrzQXxFiyH7nydrFuG46hXZ6cr
rZqfmHzEZIUzDRiOcudcyLDXmn+pMBBF6ZSlJtm7apb0alqjMRKdMul3RWTDEa9W8aIu6hCpUnVy
5jWbH5N36j7bV3Dr7dcXS4fuxtXcOvt1xdLh+7MftsKJodVOLQxtsKJodVOLQydkWj680Nd8Rndq
7h19uuLpcP3Y2ruHX264ulw/dmC22FE0OqnSoY22FE0OqnSoYyLR9eaC+Izuldw6+3XF0uH7skHC
zDqhYc0qcp1BjT8WDNzHxiIs3Fa9yO2KN3FRrdzMlCiDthRNDqp0qGSjgtiZJ4mUioVGTpUzTmyU
yku5kaI16uVWI7Ombi3TRUNrEjXO35J6asd/6TabmoUnX5BknOvjNhsipFRYtKrc6IqcaLufOU17
vZ2/+3qP81vZP3GPEGVw3taBXpumx6hDjTjJVIUGIjHirmPdss68XzP8AcixbYUTQ6qdKhnIf0fi
rlzzoUcuJOitOembkMkVEJZ72dv8A7eo/zW9kd7O3/wBvUf5reyRNTsKJodVOLQxtsKJodVOLQzz
/AAjT/wBun0NbnlX/AFVJZ72dv/t6j/Nb2R3s7f8A29R/mt7JE22womh1U6VDG2womh1U6VDH8I0
/9un0HblX/VU16Sw6oUpOQJqHGn1fAitiNR0VqpnaqKmf5voNwK/UDKgo1Wr9NpLLTqUJ8/OQZVs
R0zDVGLEejEcvoRXYzwJsjsplnfpZGjL8DRNWylSosjsq4AA2GkAAAAAAEQ5YHAXU+VymvaS8RD1
gcBdT5XKa9pJovEM4p8niTYUpEAC5EAAAAH3p3jCW/jM6yHwPvTvGet/GZ1kMLqModNDnvjpwYXb
zpF/M6EHPfHThku3nSL+ZXrF713D7kqfZNMABYiIAAAC2eQrvLuTnVupYVMLZ5Cu8u5OdW6lhzrV
8Mvt8m2HbMplucEtP57g6mMU4Lj5bnBLT+e4OpjFODfk+H91MzbQAB0jSAAAZzDvhEtfnuS/qGHR
85wYd8Ilr89yX9Qw6PlftraZ7kqn1KAACQkAAAAAAiHLA4C6nyuU17SXiIcsDgLqfK5TXtJNF4h
nFPk8SbClIgAXIgAAAA+9O8YS38ZnWQ+B96d4wlV4zOshhdRlDpoc98dOGS7edIv5nQg5746cMl2
86RfzK9Yveu4fclT7JpgALERAAAAWzyFd5dyc6t1LCphbPIV3l3Jzq3UsOdavl19vk2w7ZlMtzgl
p/PcHUXinBcfLc4Jafz3B1MYpwYsnw/upmbaAAokaQAADOYd8Ilr89yX9Qw6PnODDvhEtfnuS/qG
HR8r9tbTPclU+pQADiEgAAAAAAEQ5YHAXU+VymvaS8Rn1OUGsXJhDP0qg0+NUJ6JMyz2QIWBZORs
ZquXdVE3ERVJFIqJOxVxT5PD9lSiAN77zmKOhFu9n2h3nMUdCKp7PtFu6x5F05oQs12BogN77zmK
OhFU9n2h3nMUdCKp7PtDrEXnTmgyXYGiH3p3jCW/jM6yG6d5zFHQiqez7R9ZHB/E9k7Ae+yqojWx
Wqq/R7iIqfeMLURXbac0CNDgX70e+OnDJdvOkX8zoQUoxewsxEquKNy1Om2jUZqTmahEiQIzNhsX
tVdxUzuznAsd7WSOVy3aPuSZ0VU0ENG3vvOYo6EVT2faHecxR0Iqns+0WDrEXnTmhGyXYGiA3vvO
Yo6EVT2faHecxR0Iqns+0OsRedOaDjdgaIWzyFd5dyc6t1LCB+85ijoRVPZ9osjkg2pcdqWrXZa5
KPM0uNMVFsSEyNsc72fBNTOmZV40VDn2nNG6nVGURdW/1NsLVR2o+WW5wS0/nuDqYxTgu5lZ23Xr
ow2kqdbtLj1KbZVoUZ0KDm2SMSFFRXbqpuZ3J+JV7vOYo6EVT2faMWXLG2nuc5E0rvEzVV2hDRAb
33nMUdCKp7PtDvOYo6EVT2faOj1iLzpzQ1ZLsDRAb33nMUdCKp7PtDvOYo6EVT2faHWIvOnNBkuw
MBh3wiWvz3Jf1DDo+UWsjCXEqSve352as2pwpeXqspGjRHLdZMY2MxznL87wiIkpek4VsSME5uSq
KSYEVEW8AA4xvAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAP/9k=",
Position = new PdfPosition() { X = 40, Y = 10 }, Size = new PdfSize() {
Height = 100, Width = 250 } }
};

public async Task
ToolbarClickHandler(Syncfusion.Blazor.Navigations.ClickEventArgs args)
{
    if (args.Item.Id == "Grid_pdfexport") //Id is combination of Grid's ID and
    itemname
    {
        PdfExportProperties ExportProperties = new PdfExportProperties();
        PdfHeader Header = new PdfHeader()
        {
            FromTop = 0,
            Height = 130,
            Contents = HeaderContent
        };
    }
}

```

```

ExportProperties.Header = Header;
await this.DefaultGrid.PdfExport(ExportProperties);
}
}
protected override void OnInitialized()
{
    Orders = Enumerable.Range(1, 75).Select(x => new Order()
    {
        OrderID = 1000 + x,
        CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
    })[new Random().Next(5)],
        Freight = 2.1 * x,
        OrderDate = DateTime.Now.AddDays(-x),
    }).ToList();
}
public class Order
{
    public int? OrderID { get; set; }
    public string CustomerID { get; set; }
    public DateTime? OrderDate { get; set; }
    public double? Freight { get; set; }
}
}

```

How to change page orientation

Page orientation can be changed Landscape(Default Portrait) for the exported document using the export properties.

ASPX-CS

```

@using Syncfusion.Blazor.Grids
<SfGrid ID="Grid" @ref="DefaultGrid" DataSource="@Orders" Toolbar="@ (new
List<string>() { "PdfExport" })" AllowPdfExport="true" AllowPaging="true">
<GridEvents OnToolbarClick="ToolbarClickHandler"
TValue="Order"></GridEvents>
<GridColumn>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumn>
</SfGrid>
@code{
private SfGrid<Order> DefaultGrid;
public List<Order> Orders { get; set; }
public async Task
ToolbarClickHandler(Syncfusion.Blazor.Navigations.ClickEventArgs args)
{
    if (args.Item.Id == "Grid_pdfexport") //Id is combination of Grid's ID and
    itemname
    {

```



```
PdfExportProperties ExportProperties = new PdfExportProperties();
ExportProperties.PageOrientation =
Syncfusion.Blazor.Grids.PageOrientation.Landscape;
await this.DefaultGrid.PdfExport(ExportProperties);
}
}
protected override void OnInitialized()
{
    Orders = Enumerable.Range(1, 75).Select(x => new Order()
    {
        OrderID = 1000 + x,
        CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
    })[new Random().Next(5)],
        Freight = 2.1 * x,
        OrderDate = DateTime.Now.AddDays(-x),
    }).ToList();
}
public class Order
{
    public int? OrderID { get; set; }
    public string CustomerID { get; set; }
    public DateTime? OrderDate { get; set; }
    public double? Freight { get; set; }
}
}
```

[How to change page size](#)

Page size can be customized for the exported document using the export properties.

Supported page sizes are:

- Letter
- Note
- Legal
- A0
- A1
- A2
- A3
- A5
- A6
- A7
- A8
- A9
- B0
- B1
- B2
- B3
- B4
- B5
- ArchA
- ArchB
- ArchC

- Archd
- Arche
- Flsa
- HalfLetter
- Letter11x17
- Ledger

ASPX-CS

```
@using Syncfusion.Blazor.Grids
<SfGrid ID="Grid" @ref="DefaultGrid" DataSource="@Orders" Toolbar="@ (new
List<string>() { "PdfExport" })" AllowPdfExport="true" AllowPaging="true">
<GridEvents OnToolbarClick="ToolbarClickHandler"
TValue="Order"></GridEvents>
<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumns>
</SfGrid>
@code{
private SfGrid<Order> DefaultGrid;
public List<Order> Orders { get; set; }
public async Task
ToolbarClickHandler(Syncfusion.Blazor.Navigations.ClickEventArgs args)
{
if (args.Item.Id == "Grid_pdfexport") //Id is combination of Grid's ID and
itemname
{
PdfExportProperties ExportProperties = new PdfExportProperties();
ExportProperties.PageSize = PdfPageSize.Letter;
await this.DefaultGrid.PdfExport(ExportProperties);
}
}
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
OrderDate = DateTime.Now.AddDays(-x),
}).ToList();
}
public class Order
{
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
}
```

```
public double? Freight { get; set; }
}
}
```

[Export current page](#)

PDF export provides an option to export the current page into PDF. To export current page, define the **exportType** to **CurrentPage**.

ASPX-CS

```
@using Syncfusion.Blazor.Grids
<SfGrid ID="Grid" @ref="DefaultGrid" DataSource="@Orders" Toolbar="@ (new
List<string>() { "PdfExport" })" AllowPdfExport="true" AllowPaging="true">
<GridEvents OnToolbarClick="ToolbarClickHandler"
TValue="Order"></GridEvents>
<GridColumn>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumn>
</SfGrid>
@code{
private SfGrid<Order> DefaultGrid;
public List<Order> Orders { get; set; }
public async Task
ToolbarClickHandler(Syncfusion.Blazor.Navigations.ClickEventArgs args)
{
if (args.Item.Id == "Grid_pdfexport") //Id is combination of Grid's ID and
itemname
{
PdfExportProperties ExportProperties = new PdfExportProperties();
ExportProperties.ExportType = ExportType.CurrentPage;
await this.DefaultGrid.PdfExport(ExportProperties);
}
}
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
OrderDate = DateTime.Now.AddDays(-x),
}).ToList();
}
public class Order
{
public int? OrderID { get; set; }
public string CustomerID { get; set; }
```

```
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
}
```

Export hidden columns

PDF export provides an option to export hidden columns of DataGrid by defining the **includeHiddenColumn** as **true**.

ASPX-CS

```
@using Syncfusion.Blazor.Grids
<SfGrid ID="Grid" @ref="DefaultGrid" DataSource="@Orders" Toolbar="@ (new
List<string>() { "PdfExport" })" AllowPdfExport="true" AllowPaging="true">
<GridEvents OnToolbarClick="ToolbarClickHandler"
TValue="Order"></GridEvents>
<GridColumn>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight"
Visible="false" Format="C2" TextAlign="TextAlign.Right"
Width="120"></GridColumn>
</GridColumn>
</SfGrid>
@code{
private SfGrid<Order> DefaultGrid;
public List<Order> Orders { get; set; }
public async Task
ToolbarClickHandler(Syncfusion.Blazor.Navigations.ClickEventArgs args)
{
if (args.Item.Id == "Grid_pdfexport") //Id is combination of Grid's ID and
itemname
{
PdfExportProperties ExportProperties = new PdfExportProperties();
ExportProperties.IncludeHiddenColumn = true;
await this.DefaultGrid.PdfExport(ExportProperties);
}
}
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
OrderDate = DateTime.Now.AddDays(-x),
}).ToList();
}
public class Order
{
}
```

```

public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
}

```

Theme

PDF export provides an option to include theme for exported PDF document.

To apply theme in exported PDF, define the **theme** in export properties.

ASPX-CS

```

@using Syncfusion.Blazor.Grids
<SfGrid ID="Grid" @ref="DefaultGrid" DataSource="@Orders" Toolbar="@ (new
List<string>() { "PdfExport" })" AllowPdfExport="true" AllowPaging="true">
<GridEvents OnToolbarClick="ToolbarClickHandler"
TValue="Order"></GridEvents>
<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight"
Visible="false" Format="C2" TextAlign="TextAlign.Right"
Width="120"></GridColumn>
</GridColumns>
</SfGrid>
@code{
private SfGrid<Order> DefaultGrid;
public List<Order> Orders { get; set; }
public async Task
ToolbarClickHandler(Syncfusion.Blazor.Navigations.ClickEventArgs args)
{
if (args.Item.Id == "Grid_pdfexport") //Id is combination of Grid's ID and
itemname
{
PdfExportProperties ExportProperties = new PdfExportProperties();
PdfTheme Theme = new PdfTheme();
PdfBorder HeaderBorder = new PdfBorder();
HeaderBorder.Color = "#64FA50";
PdfThemeStyle HeaderThemeStyle = new PdfThemeStyle()
{
FontColor = "#64FA50",
FontName = "Calibri",
FontSize = 17,
Bold = true,
Border = HeaderBorder
};
Theme.Header = HeaderThemeStyle;
PdfThemeStyle RecordThemeStyle = new PdfThemeStyle()
{

```

```

FontColor = "#64FA50",
FontName = "Calibri",
FontSize = 17
};
Theme.Record = RecordThemeStyle;
PdfThemeStyle CaptionThemeStyle = new PdfThemeStyle()
{
FontColor = "#64FA50",
FontName = "Calibri",
FontSize = 17,
Bold = true
};
Theme.Caption = CaptionThemeStyle;
ExportProperties.Theme = Theme;
await this.DefaultGrid.PdfExport(ExportProperties);
}
}
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
OrderDate = DateTime.Now.AddDays(-x),
}).ToList();
}
public class Order
{
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
}

```

By default, material theme is applied to exported PDF document.

<!-- Show or hide columns on exported pdf

You can show a hidden column or hide a visible column while exporting the datagrid by customizing the visibility settings while exporting.

This is demonstrated in the below sample code where initially the **CustomerID** is hidden. While exporting, we have changed CustomerID to visible column and Freight as hidden column. Then in the PdfExportComplete event, we have reversed the column's visibility state back to the previous state.

ASPX-CS

-->

ASPX-CS

```

@using Syncfusion.Blazor.Grids
<SfGrid ID="Grid" @ref="DefaultGrid" DataSource="@Orders" Toolbar="@ (new
List<string>() { "PdfExport" })" AllowPdfExport="true" AllowPaging="true">
<GridEvents OnToolbarClick="ToolbarClickHandler"
TValue="Order"></GridEvents>
<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight"
Visible="false" Format="C2" TextAlign="TextAlign.Right"
Width="120"></GridColumn>
</GridColumns>
</SfGrid>
@code{
private SfGrid<Order> DefaultGrid;
public List<Order> Orders { get; set; }
public async Task
ToolbarClickHandler(Syncfusion.Blazor.Navigations.ClickEventArgs args)
{
if (args.Item.Id == "Grid_pdfexport") //Id is combination of Grid's ID and
itemname
{
PdfExportProperties PdfProperties = new PdfExportProperties();
PdfProperties.DataSource = Orders;
await this.DefaultGrid.PdfExport(PdfProperties);
}
}
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
OrderDate = DateTime.Now.AddDays(-x),
}).ToList();
}
public class Order
{
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
}

```

You can refer to our [Blazor DataGrid](#) feature tour page for its groundbreaking feature representations. You can also explore our [Blazor DataGrid example](#) to understand how to present and manipulate data.

<!-- markdownlint-disable MD033 -->

Excel Export in Blazor DataGrid Component

The excel export allows exporting DataGrid data to Excel document. You need to use the

ExcelExport method for exporting. To enable Excel export in the datagrid, set the [AllowExcelExport](#) property as true.

ASPX-CS

```
@using Syncfusion.Blazor.Grids
<SfGrid ID="Grid" @ref="DefaultGrid" DataSource="@Orders"
AllowSorting="true" Toolbar="@ (new List<string>() { "ExcelExport" })"
AllowExcelExport="true" AllowPaging="true">
<GridEvents OnToolbarClick="ToolbarClickHandler"
TValue="Order"></GridEvents>
<GridColumn>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumn>
</SfGrid>
@code{
private SfGrid<Order> DefaultGrid;
public List<Order> Orders { get; set; }
public async Task
ToolbarClickHandler(Syncfusion.Blazor.Navigations.ClickEventArgs args)
{
if (args.Item.Id == "Grid_excelelexport") //Id is combination of Grid's ID and
itemname
{
await this.DefaultGrid.ExcelExport();
}
}
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
OrderDate = DateTime.Now.AddDays(-x),
}).ToList();
}
public class Order
{
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
}
```


To customize excel export

The excel export provides an option to customize mapping of the datagrid to excel document.

Export current page

The excel export provides an option to export the current page into excel. To export current page, define **exportType** to **CurrentPage**.

ASPX-CS

```
@using Syncfusion.Blazor.Grids
<SfGrid ID="Grid" @ref="DefaultGrid" DataSource="@Orders"
AllowSorting="true" Toolbar="@new List<string>() { "ExcelExport" })"
AllowExcelExport="true" AllowPaging="true">
<GridEvents OnToolbarClick="ToolbarClickHandler"
TValue="Order"></GridEvents>
<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumns>
</SfGrid>
@code{
private SfGrid<Order> DefaultGrid;
public List<Order> Orders { get; set; }
public async Task
ToolbarClickHandler(Syncfusion.Blazor.Navigations.ClickEventArgs args)
{
if (args.Item.Id == "Grid_excelext") //Id is combination of Grid's ID and
itemname
{
ExcelExportProperties ExportProperties = new ExcelExportProperties();
ExportProperties.ExportType = ExportType.CurrentPage;
await this.DefaultGrid.ExcelExport(ExportProperties);
}
}
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
OrderDate = DateTime.Now.AddDays(-x),
}).ToList();
}
public class Order
{
public int? OrderID { get; set; }
```

```

public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
}

```

Export hidden columns

The excel export provides an option to export hidden columns of datagrid by defining **includeHiddenColumn** as **true**.

ASPX-CS

```

@using Syncfusion.Blazor.Grids
<SfGrid ID="Grid" @ref="DefaultGrid" DataSource="@Orders"
AllowSorting="true" Toolbar="@ (new List<string>() { "ExcelExport" })"
AllowExcelExport="true" AllowPaging="true">
<GridEvents OnToolbarClick="ToolbarClickHandler"
TValue="Order"></GridEvents>
<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight"
Visible="false" Format="C2" TextAlign="TextAlign.Right"
Width="120"></GridColumn>
</GridColumns>
</SfGrid>
@code{
private SfGrid<Order> DefaultGrid;
public List<Order> Orders { get; set; }
public async Task
ToolbarClickHandler(Syncfusion.Blazor.Navigations.ClickEventArgs args)
{
if (args.Item.Id == "Grid_excelexport") //Id is combination of Grid's ID and
itemname
{
ExcelExportProperties ExportProperties = new ExcelExportProperties();
ExportProperties.IncludeHiddenColumn = true;
await this.DefaultGrid.ExcelExport(ExportProperties);
}
}
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
OrderDate = DateTime.Now.AddDays(-x),
}).ToList();
}
}

```

```
public class Order
{
    public int? OrderID { get; set; }
    public string CustomerID { get; set; }
    public DateTime? OrderDate { get; set; }
    public double? Freight { get; set; }
}
}
```

<!-- Show or hide columns on exported excel

You can show a hidden column or hide a visible column while exporting the datagrid by customizing the visibility settings while exporting.

This is demonstrated in the below sample code where initially the **CustomerID** is hidden. While exporting, we have changed CustomerID to visible column and Freight as hidden column. Then in the **ExcelExportComplete** event, we have reversed the column's visibility state back to the previous state.

ASPX-CS

-->

ASPX-CS

```
@using Syncfusion.Blazor.Grids
<SfGrid ID="Grid" @ref="DefaultGrid" DataSource="@Orders"
AllowSorting="true" Toolbar="@ (new List<string>() { "ExcelExport" })"
AllowExcelExport="true" AllowPaging="true">
<GridEvents OnToolbarClick="ToolbarClickHandler"
TValue="Order"></GridEvents>
<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumns>
</SfGrid>
@code{
private SfGrid<Order> DefaultGrid;
public List<Order> Orders { get; set; }
public async Task
ToolbarClickHandler(Syncfusion.Blazor.Navigations.ClickEventArgs args)
{
    if (args.Item.Id == "Grid_excelext") //Id is combination of Grid's ID and
    itemname
    {
        ExcelExportProperties ExcelProperties = new ExcelExportProperties();
        ExcelTheme Theme = new ExcelTheme();
```

```

ExcelStyle ThemeStyle = new ExcelStyle()
{
    FontName = "Segoe UI",
    FontColor = "#666666",
    FontSize = 20
};
Theme.Header = ThemeStyle;
Theme.Record = ThemeStyle;
Theme.Caption = ThemeStyle;
ExcelProperties.Theme = Theme;
await this.DefaultGrid.ExcelExport(ExcelProperties);
}
}
protected override void OnInitialized()
{
    Orders = Enumerable.Range(1, 75).Select(x => new Order()
    {
        OrderID = 1000 + x,
        CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
    })[new Random().Next(5)],
        Freight = 2.1 * x,
        OrderDate = DateTime.Now.AddDays(-x),
    }).ToList();
}
public class Order
{
    public int? OrderID { get; set; }
    public string CustomerID { get; set; }
    public DateTime? OrderDate { get; set; }
    public double? Freight { get; set; }
}
}

```

By default, material theme is applied to exported excel document.

<!-- To add header and footer

The excel export provides an option to include header and footer content for exported excel document using the [ExcelExportProperties](#) class.

This is demonstrated in the below sample code,

ASPX-CS

-->

ASPX-CS

```

@using Syncfusion.Blazor.Grids
<SfGrid ID="Grid" @ref="DefaultGrid" DataSource="@Orders"
AllowSorting="true" Toolbar="@ (new List<string>() { "ExcelExport" })"
AllowExcelExport="true" AllowPaging="true">
<GridEvents OnToolbarClick="ToolbarClickHandler"
TValue="Order"></GridEvents>

```

```

<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumns>
</SfGrid>
@code{
private SfGrid<Order> DefaultGrid;
public List<Order> Orders { get; set; }
public async Task
ToolbarClickHandler(Syncfusion.Blazor.Navigations.ClickEventArgs args)
{
if (args.Item.Id == "Grid_excelexport") //Id is combination of Grid's ID and
itemname
{
ExcelExportProperties ExcelProperties = new ExcelExportProperties();
ExcelProperties.FileName = "new.xlsx";
await this.DefaultGrid.ExcelExport(ExcelProperties);
}
}
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
OrderDate = DateTime.Now.AddDays(-x),
}).ToList();
}
public class Order
{
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
}

```

Exporting grouped records

The excel export provides outline option for grouped records which hides the detailed data for better viewing.

In datagrid, we have provided the outline option for the exported document when the data's are grouped.

ASPX-CS

```
@using Syncfusion.Blazor.Grids
```

```

<SfGrid ID="Grid" @ref="DefaultGrid" DataSource="@Orders"
AllowGrouping="true" Toolbar="@ (new List<string>() { "ExcelExport" })"
AllowExcelExport="true" AllowPaging="true">
<GridEvents OnToolbarClick="ToolbarClickHandler"
TValue="Order"></GridEvents>
<GridGroupSettings Columns="@ (new string[]
{ "OrderDate" })"></GridGroupSettings>
<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText="Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight"
Visible="false" Format="C2" TextAlign="TextAlign.Right"
Width="120"></GridColumn>
</GridColumns>
</SfGrid>
@code{
private SfGrid<Order> DefaultGrid;
public List<Order> Orders { get; set; }
public async Task
ToolbarClickHandler(Syncfusion.Blazor.Navigations.ClickEventArgs args)
{
if (args.Item.Id == "Grid_excelexport") //Id is combination of Grid's ID and
itemname
{
await this.DefaultGrid.ExcelExport();
}
}
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
OrderDate = DateTime.Now.AddDays(-x),
}).ToList();
}
public class Order
{
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
}

```

Limitations

Microsoft Excel permits up to seven nested levels in outlines. So that in the datagrid we can able to provide only up to seven nested levels and if it exceeds more than seven levels then the document will be exported without outline option. Please refer the [Microsoft Limitation](#)

How to export the Grid with specific columns

You can export the Excel grid with specific columns instead of all columns which are defined in the Grid definition. To achieve this scenario by using [Columns](#) property of the [ExcelExportProperties](#) class.

ASPX-CS

```
@using Syncfusion.Blazor.Grids
<SfGrid ID="Grid" @ref="DefaultGrid" DataSource="@Orders"
AllowExcelExport="true" AllowPaging="true" Toolbar="@ (new List<string>() {
"ExcelExport" }) ">
<GridEvents OnToolbarClick="OnToolbarClick" TValue="Order"></GridEvents>
<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumns>
</SfGrid>
@code{
private SfGrid<Order> DefaultGrid;
public List<Order> Orders { get; set; }
public async Task
OnToolbarClick(Syncfusion.Blazor.Navigations.ClickEventArgs args)
{
if (args.Item.Id == "Grid_excelexport") //Id is combination of Grid's ID and
itemname
{
ExcelExportProperties ExportProperties = new ExcelExportProperties();
List<GridColumn> ExportColumns = new List<GridColumn>();
#pragma warning disable BL0005
ExportColumns.Add(new GridColumn() { Field = "CustomerID", HeaderText =
"Customer Name", Width = "100" });
ExportColumns.Add(new GridColumn() { Field = "OrderDate", HeaderText =
"Date", Width = "120", Format = "d" });
ExportColumns.Add(new GridColumn() { Field = "Freight", HeaderText =
"Freight", Width = "120", Format = "C2", TextAlign = TextAlign.Right });
#pragma warning restore BL0005
ExportProperties.Columns = ExportColumns;
await this.DefaultGrid.ExcelExport(ExportProperties);
}
}
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 15).Select(x => new Order()
{
OrderID = 1000 + x,
```

```

CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
ShipCountry = (new string[] { "Germany", "UK", "USA", "Italy", "France"
})[new Random().Next(5)],
Freight = 2.1 * x,
OrderDate = DateTime.Now.AddDays(-x),
}).ToList();
}
public class Order
{
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public string ShipCountry { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
}

```

<!-- Multiple datagrid exporting

The excel export provides option to export multiple datagrid data in the same excel file.

Same sheet

The excel export provides support to export multiple grids in the same sheet. To export in same sheet, set the [Type](#) value of [MultipleExport](#) property as **AppendToSheet** in the [ExcelExportProperties](#) class. It is possible to provide blank rows between the grids. The blank rows count can be set by defining the value in [BlankRows](#) of [MultipleExport](#) property.

This is demonstrated in the below sample code block,

ASPX-CS

```

@using Syncfusion.Blazor.Buttons
@using Syncfusion.Blazor.Grids
<SfButton OnClick="ExcelExport" Content="Excel Export"></SfButton>
<SfGrid @ref="FirstGrid" DataSource="@Orders1" AllowExcelExport="true"
AllowPaging="true">
<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumns>
</SfGrid>
<SfGrid @ref="SecondGrid" DataSource="@Orders2" AllowExcelExport="true"
AllowPaging="true">
<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>

```



```

<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumns>
</SfGrid>
@code{
private SfGrid<Order> FirstGrid;
private SfGrid<Order> SecondGrid;
public List<Order> Orders1 { get; set; }
public List<Order> Orders2 { get; set; }
protected override void OnInitialized()
{
Orders1 = Enumerable.Range(1, 15).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
OrderDate = DateTime.Now.AddDays(-x),
}).ToList();
Orders2 = Enumerable.Range(1, 15).Select(x => new Order()
{
OrderID = 2000 + x,
CustomerID = (new string[] { "JANET", "MARGARET", "NANCY", "STEVEN", "VINET"
})[new Random().Next(5)],
Freight = 3.1 * x,
OrderDate = DateTime.Now.AddDays(-x),
}).ToList();
}
public class Order
{
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
public void ExcelExport()
{
ExcelExportProperties ExcelProperties = new ExcelExportProperties();
MultipleExport MultipleExportProperties = new MultipleExport()
{
Type = MultipleExportType.AppendToSheet,
BlankRows = 2
};
ExcelProperties.MultipleExport = MultipleExportProperties;
var ExportData = this.FirstGrid.ExcelExport(ExcelProperties, true);
ExportData.ContinueWith((ExcelData) =>
{
this.SecondGrid.ExcelExport(ExcelProperties, false, ExcelData);
});
}
}

```

By default, [BlankRows](#) value is 5. -->

<!-- New sheet

The excel export provides support to export multiple grids in new sheet. To export in new sheet, set the [Type](#) value of [MultipleExport](#) property as **NewSheet** in the [ExcelExportProperties](#) class.

This is demonstrated in the below sample code block,

ASPX-CS

-->

ASPX-CS

```
@using Syncfusion.Blazor.Grids
<SfGrid ID="Grid" @ref="DefaultGrid" DataSource="@Orders"
AllowExcelExport="true" AllowPaging="true" Toolbar="@ (new List<string>() {
"ExcelExport" }) ">
<GridEvents OnToolbarClick="OnToolbarClick" TValue="Order"></GridEvents>
<GridColumn>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumn>
</SfGrid>
@code{
private SfGrid<Order> DefaultGrid;
public List<Order> Orders { get; set; }
public async Task
OnToolbarClick(Syncfusion.Blazor.Navigations.ClickEventArgs args)
{
if (args.Item.Id == "Grid_excelexport") //Id is combination of Grid's ID and
itemname
{
ExcelExportProperties ExcelProperties = new ExcelExportProperties();
ExcelProperties.DataSource = Orders;
await this.DefaultGrid.ExcelExport(ExcelProperties);
}
}
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 15).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
ShipCountry = (new string[] { "Germany", "UK", "USA", "Italy", "France"
})[new Random().Next(5)],
Freight = 2.1 * x,
OrderDate = DateTime.Now.AddDays(-x),
```

```

    }).ToList();
}
public class Order
{
    public int? OrderID { get; set; }
    public string CustomerID { get; set; }
    public string ShipCountry { get; set; }
    public DateTime? OrderDate { get; set; }
    public double? Freight { get; set; }
}
}

```

Customizing columns

Excel export provides an option to define the columns dynamically before exporting. You can define the dynamic columns using [Columns](#) property of the [ExcelExportProperties](#) class.

The following sample code demonstrates dynamically adding `ShipCountry` column in the exported excel file,

ASPX-CS

```

@using Syncfusion.Blazor.Grids
<SfGrid ID="Grid" @ref="DefaultGrid" DataSource="@Orders"
AllowExcelExport="true" AllowPaging="true" Toolbar="@ (new List<string>() {
"ExcelExport" })">
<GridEvents OnToolbarClick="OnToolbarClick" TValue="Order"></GridEvents>
<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumns>
</SfGrid>
@code{
private SfGrid<Order> DefaultGrid;
public List<Order> Orders { get; set; }
public async Task
OnToolbarClick(Syncfusion.Blazor.Navigations.ClickEventArgs args)
{
if (args.Item.Id == "Grid_excelexport") //Id is combination of Grid's ID and
itemname
{
ExcelExportProperties ExportProperties = new ExcelExportProperties();
var columns = await DefaultGrid.GetColumns(); //get the columns
available in Grid
columns.Add(new GridColumn() { Field = "ShipCountry" }); //adding
additional column
ExportProperties.Columns = columns;
await DefaultGrid.ExcelExport(ExportProperties);
}
}
}

```

```
protected override void OnInitialized()
{
    Orders = Enumerable.Range(1, 15).Select(x => new Order()
    {
        OrderID = 1000 + x,
        CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
    })[new Random().Next(5)],
        ShipCountry = (new string[] { "Germany", "UK", "USA", "Italy", "France"
    })[new Random().Next(5)],
        Freight = 2.1 * x,
        OrderDate = DateTime.Now.AddDays(-x),
    }).ToList();
}

public class Order
{
    public int? OrderID { get; set; }
    public string CustomerID { get; set; }
    public string ShipCountry { get; set; }
    public DateTime? OrderDate { get; set; }
    public double? Freight { get; set; }
}
}
```

You can refer to our [Blazor DataGrid](#) feature tour page for its groundbreaking feature representations. You can also explore our [Blazor DataGrid example](#) to understand how to present and manipulate data.

<!-- Export the hierarchy datagrid

The datagrid has option to export hierarchy datagrid to the excel document. By default, datagrid exports the master datagrid with expanded child grids alone. The exporting option can be modified by using the [HierarchyExportMode](#) property of the [ExcelExportProperties](#) class.

The available options are,

| Mode | Behavior |
|----------|--|
| Expanded | Exports the master datagrid with expanded child grids. |
| All | Exports the master datagrid with all the child grids. |
| None | Exports the master datagrid alone. |

The following sample code demonstrates modifying the export options for hierarchy datagrid,

ASPX-CS

-->

ASPX-CS

```
@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders">
```

```

<GridSelectionSettings
Type="SelectionType.Multiple"></GridSelectionSettings>
<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumns>
</SfGrid>
@code{
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
OrderDate = DateTime.Now.AddDays(-x),
}).ToList();
}
public class Order {
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
}

```

Copy to clipboard by external buttons

To copy selected rows or cells data into the clipboard with help of toolbar buttons, you need to invoke the **Copy** method.

ASPX-CS

```

@using Syncfusion.Blazor.Buttons
@using Syncfusion.Blazor.Grids
<SfButton OnClick="Copy" Content="Copy"></SfButton>
<SfButton OnClick="CopyHeader" Content="Copy With Header"></SfButton>
<SfGrid @ref="DefaultGrid" DataSource="@Orders">
<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>

```

```

</GridColumns>
</SfGrid>
@code{
private SfGrid<Order> DefaultGrid;
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
OrderDate = DateTime.Now.AddDays(-x),
}).ToList();
}
public void Copy()
{
this.DefaultGrid.Copy();
}
public void CopyHeader()
{
this.DefaultGrid.Copy(true);
}
public class Order
{
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
}

```

You can refer to the [Blazor DataGrid](#) feature tour page for its groundbreaking feature representations. You can also explore [Blazor DataGrid example](#) to understand how to present and manipulate data.

AutoFill

AutoFill Feature allows to copy the data of selected cells and paste it to another cells by just dragging the autofill icon of the selected cells to the required cells. This feature is enabled by defining the [EnableAutoFill](#) property as true.

ASPX-CS

```

@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders" EnableAutoFill="true" AllowSelection="true"
Toolbar="@ (new List<string>() { "Add", "Update", "Cancel" })"
AllowPaging="true">
<GridSelectionSettings CellSelectionMode="CellSelectionMode.Box"
Mode="SelectionMode.Cell"
Type="SelectionType.Multiple"></GridSelectionSettings>
<GridEditSettings AllowAdding="true" AllowDeleting="true"
AllowEditing="true" Mode="EditMode.Batch"></GridEditSettings>
<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
IsPrimaryKey="true" TextAlign="TextAlign.Right" Width="120"></GridColumn>

```

```

<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumn>
</SfGrid>
@code{
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
OrderDate = DateTime.Now.AddDays(-x),
}).ToList();
}
public class Order
{
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
}

```

The following represents Autofill in DataGrid

Order ID	Customer Name	Order Date	Freight
1001	ANANTR	7/21/2019	\$2.10
1002	BLONP	7/20/2019	\$4.20
1003	ALFKI	7/19/2019	\$6.30
1004	ANTON	7/18/2019	\$8.40
1005	ANANTR	7/17/2019	\$10.50
1006	BOLID	7/16/2019	\$12.60
1007	ANANTR	7/15/2019	\$14.70
1008	ANANTR	7/14/2019	\$16.80
1009	ANTON	7/13/2019	\$18.90

K < 1 2 3 4 5 6 7 > X
1 of 7 pages (75 items)

* If [EnableAutoFill](#) is set to true, then the autofill icon will be displayed on cell selection to copy cells.

* It requires the selection [Mode](#) to be **Cell** and [CellSelectionMode](#) to be **Box** and also Batch Editing should be enabled.

Limitations of AutoFill

- Since the string values are not parsed to number and date type, when the selected string type cells are dragged to number type cells then it will display as **NaN**. For date type cells, when the selected string type cells are dragged to date type cells then it will display as an **empty cell**.
- Linear series and the sequential data generations are not supported in this autofill feature.

Paste

You can copy the content of a cell or a group of cells by selecting the cells and pressing Ctrl + C shortcut key, and paste it to another set of cells by selecting the cells and pressing Ctrl + V shortcut key.

ASPX-CS

```
@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders" EnableAutoFill="true" AllowSelection="true"
Toolbar="@ (new List<string>() { "Add", "Update", "Cancel" })">
<GridSelectionSettings CellSelectionMode="CellSelectionMode.Box"
Mode="SelectionMode.Cell"
Type="SelectionMode.Multiple"></GridSelectionSettings>
<GridEditSettings AllowAdding="true" AllowDeleting="true"
AllowEditing="true" Mode="EditMode.Batch"></GridEditSettings>
<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
IsPrimaryKey="true" TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText="Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumns>
</SfGrid>
@code{
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
OrderDate = DateTime.Now.AddDays(-x),
}).ToList();
}
public class Order {
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
```



```
}
}
```

* If [EnableAutoFill](#) is set to true, then the autofill icon will be displayed on cell selection to copy cells.

* To perform paste functionality, it requires the selection [Mode](#) to be **Cell** and [CellSelectionMode](#) to be **Box** and also Batch Editing should be enabled.

Limitation of Paste Functionality

Since the string values are not parsed to number and date type, when the copied string type cells are pasted to number type cells then it will display as **NaN**. For date type cells, when the copied string format cells are pasted to date type cells then it will display as an **empty cell**.

Context Menu in Blazor DataGrid Component

The DataGrid has options to show the context menu when right clicked on it. To enable this feature, you need to define either default or custom item in the [ContextMenuItems](#) property.

The following table lists the default context menu items,

Items | Description

Header | AutoFit

Header | AutoFitAll

Header | Group

Header | Ungroup

Header | SortAscending

Header | SortDescending

Content | Edit

Content | Delete

Content | Save

Content | Cancel

Content | Copy

Content | PdfExport

Content | ExcelExport

Content | CsvExport

Pager | FirstPage

Pager | PrevPage

Pager | LastPage

Pager | NextPage

The following sample code demonstrates enabling context menu with its default items,

ASPX-CS

```
@using Syncfusion.Blazor.Grids
```

```

<SfGrid DataSource="@Orders" AllowSorting="true" AllowPaging="true"
AllowExcelExport="true" AllowPdfExport="true" ContextMenuItems="@ (new
List<object>() { "AutoFit", "AutoFitAll", "SortAscending",
"SortDescending","Copy", "Edit", "Delete", "Save", "Cancel","PdfExport",
"ExcelExport", "CsvExport", "FirstPage", "PrevPage","LastPage",
"NextPage"})">
<GridPageSettings PageSize="8"></GridPageSettings>
<GridEditSettings AllowEditing="true"
AllowDeleting="true"></GridEditSettings>
<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
IsPrimaryKey="true" TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumns>
</SfGrid>
@code{
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
OrderDate = DateTime.Now.AddDays(-x),
}).ToList();
}
public class Order
{
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
}

```

The following GIF represents the DataGrid enabled with default context menu items,

Order ID	Customer Name	Order Date	Freight
1001	ALFKI	9/8/2019	\$2.10
1002	BOLID	9/7/2019	\$4.20
1003	BOLID	9/6/2019	\$6.30
1004	ALFKI	9/5/2019	\$8.40
1005	ANTON	9/4/2019	\$10.50
1006	ANTON	9/3/2019	\$12.60
1007	ALFKI	9/2/2019	\$14.70
1008	BLONP	9/1/2019	\$16.80

K < 1 2 3 4 5 6 7 8 9 10 > X 1 of 10 pages (75 items)

Custom context menu items

The custom context menu items can be added by defining the [ContextMenuItems](#) as a collection of [ContextMenuItemModel](#). Actions for these customized items can be defined in the [ContextMenuItemClicked](#) event.

The following sample code demonstrates defining custom context menu item and its corresponding action in the [ContextMenuItemClicked](#) event,

ASPX-CS

```
@using Syncfusion.Blazor.Grids
<SfGrid @ref="DefaultGrid" DataSource="@Orders" AllowPaging="true"
ContextMenuItems="@ (new List<ContextMenuItemModel>() { new
ContextMenuItemModel { Text = "Copy with headers", Target = ".e-content", Id
= "copywithheader" } })">
<GridEvents ContextMenuItemClicked="OnContextMenuClick"
TValue="Order"></GridEvents>
<GridPageSettings PageSize="8"></GridPageSettings>
<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
TextAlign="TextAlign.Right" Width="120" IsPrimaryKey="true"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumns>
```

```

</SfGrid>
@code{
public List<Order> Orders { get; set; }
private SfGrid<Order> DefaultGrid;
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
OrderDate = DateTime.Now.AddDays(-x),
}).ToList();
}
public class Order
{
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
public void OnContextMenuClick(ContextMenuClickEventArgs<Order> args)
{
if (args.Item.Id == "copywithheader")
{
this.DefaultGrid.Copy(true);
}
}
}

```

The following image represents the DataGrid enabled with custom context menu item,

Order ID	Customer Name	Order Date	Freight
1001	ALFKI	9/8/2019	\$2.10
1002	ALFKI	9/7/2019	\$4.20
1003	BOLID	9/6/2019	\$6.30
1004	ANANTR	9/5/2019	\$8.40
1005	ANANTR	9/4/2019	\$10.50
1006	BOLID	9/3/2019	\$12.60
1007	ANANTR	9/2/2019	\$14.70
1008	BLONP	9/1/2019	\$16.80

K < 1 2 3 4 5 6 7 8 9 10 > X
1 of 10 pages (75 items)

Built-in and Custom context menu items

DataGrid has an option to use both built-in and custom context menu items at same time.

The following sample code demonstrates defining built-in and custom context menu items and custom context menu item corresponding action in the [ContextMenuItemClicked](#) event,

ASPX-CS

```
@using Syncfusion.Blazor.Grids
<SfGrid @ref="DefaultGrid" DataSource="@Orders" AllowPaging="true"
ContextMenuItems="@ (new List<Object>() { "Copy", new ContextMenuItemModel {
Text = "Copy with headers", Target = ".e-content", Id = "copywithheader" }
})">
<GridEvents ContextMenuItemClicked="OnContextMenuClick"
TValue="Order"></GridEvents>
<GridPageSettings PageSize="8"></GridPageSettings>
<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
TextAlign="TextAlign.Right" Width="120" IsPrimaryKey="true"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText="Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumns>
</SfGrid>
@code{
public List<Order> Orders { get; set; }
private SfGrid<Order> DefaultGrid;
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
OrderDate = DateTime.Now.AddDays(-x),
}).ToList();
}
public class Order
{
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
public void OnContextMenuClick(ContextMenuClickEventArgs<Order> args)
{
if (args.Item.Id == "copywithheader")
{
this.DefaultGrid.Copy(true);
}
}
}
```

Sub context menu items in DataGrid

The sub context menu items can be added by defining the collection of **MenuItems** for **Items** Property in [ContextMenuItems](#). Actions for these customized items can be defined in the [ContextMenuItemClicked](#) event.

The following sample code demonstrates defining sub context menu item and its corresponding action in the [ContextMenuItemClicked](#) event,

ASPX-CS

```
@using Syncfusion.Blazor.Grids
@using Syncfusion.Blazor.Navigations
<SfGrid @ref="DefaultGrid" DataSource="@Orders" AllowPaging="true"
ContextMenuItems="@ (new List<ContextMenuItemModel>() { new
ContextMenuItemModel { Text = "Clipboard", Target = ".e-content", Id =
"clipboard", Items = new List<MenuItem>() { new MenuItem { Text = "Copy", Id
= "copy" }, new MenuItem { Text ="CopyWithHeader", Id ="copywithheader" } }
} ) ">
<GridEvents ContextMenuItemClicked="OnContextMenuClick"
TValue="Order"></GridEvents>
<GridPageSettings PageSize="8"></GridPageSettings>
<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
TextAlign="TextAlign.Right" Width="120" IsPrimaryKey="true"></GridColumn>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
TextAlign="TextAlign.Right" Width="120" IsPrimaryKey="true"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumns>
</SfGrid>
@code{
public List<Order> Orders { get; set; }
private SfGrid<Order> DefaultGrid;
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
OrderDate = DateTime.Now.AddDays(-x),
}).ToList();
}
public class Order
{
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
```

```

}
public void OnContextMenuClick(ContextMenuClickEventArgs<Order> args)
{
    if (args.Item.Id == "copywithheader")
    {
        this.DefaultGrid.Copy(true);
    }
    if (args.Item.Id == "copy")
    {
        this.DefaultGrid.Copy(false);
    }
}
}

```

Disable the Context menu for specific columns in DataGrid

Context Menu can be prevented for specific columns using [ContextMenuOpen](#) event of DataGrid. This event will be triggered before opening the ContextMenu. We can prevent the context menu from opening by defining the **Cancel** arguments of [ContextMenuOpen](#) to **false**.

The following sample code demonstrates how to disable the context for specific column using event arguments of [ContextMenuOpen](#) event,

ASPX-CS

```

@using Syncfusion.Blazor.Grids
<SfGrid @ref="DefaultGrid" DataSource="@Orders" AllowPaging="true"
ContextMenuItemModel="@((new List<ContextMenuItemModel>() { new
ContextMenuItemModel { Text = "Copy with headers", Target = ".e-content", Id
= "copywithheader" } })">
<GridEvents ContextMenuItemClicked="OnContextMenuClick"
ContextMenuOpen="OnContextMenuOpen" TValue="Order"></GridEvents>
<GridPageSettings PageSize="8"></GridPageSettings>
<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
TextAlign="TextAlign.Right" Width="120" IsPrimaryKey="true"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText="Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumns>
</SfGrid>
@code{
public List<Order> Orders { get; set; }
private SfGrid<Order> DefaultGrid;
protected override void OnInitialized()
{
    Orders = Enumerable.Range(1, 75).Select(x => new Order()
    {
        OrderID = 1000 + x,
        CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
    })[new Random().Next(5)],
        Freight = 2.1 * x,
        OrderDate = DateTime.Now.AddDays(-x),
    });
}
}

```

```

    }).ToList();
}
public class Order
{
    public int? OrderID { get; set; }
    public string CustomerID { get; set; }
    public DateTime? OrderDate { get; set; }
    public double? Freight { get; set; }
}
public void OnContextMenuOpen(ContextMenuOpenEventArgs<Order> Args)
{
    if (Args.Column.Field == "OrderDate")
    {
        Args.Cancel = true; // To prevent the context menu from opening
    }
}
public void OnContextMenuClick(ContextMenuClickEventArgs<Order> args)
{
    if (args.Item.Id == "copywithheader")
    {
        this.DefaultGrid.Copy(true);
    }
}
}

```

Disable context menu items dynamically in DataGrid

You can enable or disable context menu items using the **Disabled** property. Here, you can enable and disable the **Edit** context menu items in [ContextMenuOpen](#) event of DataGrid. This event will be triggered before opening the ContextMenu. You can disable the context menu item by defining the corresponding context menu items **Disabled** property as **true**.

The following sample code demonstrates how to enable or disable context menu items dynamically in Grid using event arguments of [ContextMenuOpen](#) event,

ASPX-CS

```

@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders" AllowPaging="true" ContextMenuItems="@ (new
List<object>() { "Edit", "Delete", "Save", "Cancel", "PdfExport",
"ExcelExport", "CsvExport", "FirstPage", "PrevPage", "LastPage",
"NextPage"}) ">
<GridEvents ContextMenuOpen="OnContextMenuOpen" TValue="Order"></GridEvents>
<GridPageSettings PageSize="8"></GridPageSettings>
<GridEditSettings AllowEditing="true"
AllowDeleting="true"></GridEditSettings>
<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
IsPrimaryKey="true" TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumns>

```



```
</SfGrid>
@code{
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
OrderDate = DateTime.Now.AddDays(-x),
}).ToList();
}
public void OnContextMenuOpen(ContextMenuOpenEventArgs<Order> Args)
{
#pragma warning disable BL0005
if (Args.Column.Field == "OrderDate") // You can check condition based on
your requirement
{
Args.ContextMenuObj.Items[0].Disabled = true; // To disable edit context
menu item
}
else
{
Args.ContextMenuObj.Items[0].Disabled = false; // To enable edit context
menu item
}
#pragma warning restore BL0005
}
public class Order
{
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
}
```

You can refer to [Blazor DataGrid](#) feature tour page for its groundbreaking feature representations. You can also explore [Blazor DataGrid example](#) to understand how to present and manipulate data.

Accessibility in Blazor DataGrid Component

Accessibility is achieved in the DataGrid component through WAI-ARIA standard and keyboard navigations. The DataGrid features can be effectively accessed through assistive technologies such as screen readers.

WAI-ARIA

WAI-ARIA (Accessibility Initiative – Accessible Rich Internet Applications) defines a way to increase the accessibility of web pages, dynamic content, and user interface components developed with Ajax, HTML, JavaScript, and related technologies. ARIA provides additional semantics to describe the role, state, and functionality of web components.

The following ARIA attributes are used in the DataGrid:

- grid (role)
- row (role)
- gridcell (role)
- aria-selected (attribute)
- aria-expanded (attribute)
- aria-sort (attribute)
- aria-busy (attribute)
- aria-invalid (attribute)
- aria-grabbed (attribute)
- aria-owns (attribute)
- aria-label (attribute)

Keyboard navigation

DataGrid functionalities can be interactive with keyboard shortcuts.

The following keyboard shortcuts are supported by DataGrid.

Interaction Keys | Description

PageDown | Goes to the next page.

PageUp | Goes to the previous page.

Ctrl + Alt + PageDown | Goes to the last page.

Ctrl + Alt + PageUp | Goes to the first page.

Alt + PageDown | Goes to the next page.

Alt + PageUp | Goes to the previous page.

Home | Goes to the first cell.

End | Goes to the last cell.

Ctrl + Home | Goes to the first row.

Ctrl + End | Goes to the last row.

DownArrow | Moves the cell focus downward.

UpArrow | Moves the cell focus upward.

LeftArrow | Moves the cell focus left side.

RightArrow | Moves the cell focus right side.

Shift + DownArrow | Extends the row/cell selection downwards.

Shift + UpArrow | Extends the row/cell selection upwards.

Shift + LeftArrow | Extends the cell selection to the left side.

Shift + RightArrow | Extends the cell selection to the right side.

Enter | Moves the row/cell selection downward. If current cell is in edit state, then completes the editing. If the current cell is a header then performs sorting. If the current cell is an

expand/collapse cell then expands/collapses the current group/detailrow/childgrid. If the current cell is a detailrow, child datagrid or command column then the focus will be moved to the focusable element inside the cell.

Shift + Enter | Moves the row/cell selection upward. If the current cell is a header then clears sorting for the selected column.

Ctrl + Enter | If the current cell is a header then performs multi-sorting.

Tab | Moves the cell selection right side.

Shift + Tab | Moves the cell selection left side.

Esc | Deselects all the rows/cells.

Ctrl + A | Selects all the rows/cells.

UpArrow | Moves up a row/cell selection.

DownArrow | Moves down a row/cell selection.

RightArrow | Moves to the right cell selection.

LeftArrow | Moves to the left cell selection.

Alt + DownArrow | Expands the selected group.

Ctrl + DownArrow | Expands all the visible groups.

Alt + UpArrow | Collapses the selected group.

Ctrl + UpArrow | Collapses all the visible groups.

Ctrl + P | Prints the DataGrid.

Ctrl + C | Copy selected rows or cells data into clipboard

Ctrl + Shift + H | Copy selected rows or cells data with header into clipboard

WebAssembly Performance in Blazor DataGrid Component

This section provides performance guidelines for using Syncfusion data grid component efficiently in Blazor WebAssembly application. The general framework Blazor WebAssembly performance best practice/guidelines can be found [here](#).

You can refer to our Getting Started with [Blazor Server-Side DataGrid](#) and [Blazor WebAssembly DataGrid](#) documentation pages for configuration specifications.

Avoid unnecessary component renders

During Blazor diffing algorithm, every cells of the grid component and its child component will be checked for re-rendering. For instance, having **EventCallback** on the application or grid will check every child component once event callback is completed.

You can have fine-grained control over grid component rendering. **PreventRender** method help you to avoid unnecessary re-rendering of the grid component. This method internally overrides the **ShouldRender** method of the grid to prevent rendering.

In the following example:

- **PreventRender** method is called in the **IncrementCount** method which is a click callback.
- Now grid component will not be a part of the rendering which happens as result of the click event and **currentCount** alone will get updated.

ASPX-CS

```
<h1>Counter</h1>
<p>Current count: @currentCount</p>
<button class="btn btn-primary" @onclick="IncrementCount">Click me</button>
<SfGrid @ref="grid" DataSource="@Orders" AllowPaging="true">
  <GridColumn>
    <GridColumn Type="ColumnType.CheckBox" Width="50"></GridColumn>
    <GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
      TextAlign="TextAlign.Right" Width="120"></GridColumn>
    <GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
      Width="150"></GridColumn>
    <GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
      Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
      Width="130"></GridColumn>
    <GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
      TextAlign="TextAlign.Right" Width="120"></GridColumn>
    <GridColumn Field=@nameof(Order.Verified)" DisplayAsCheckBox="true"
      Width="70"></GridColumn>
  </GridColumn>
</SfGrid>
@code {
  SfGrid<Order> grid { get; set; }
  private int currentCount = 0;
  public List<Order> Orders { get; set; }
  protected override void OnInitialized()
  {
    Orders = Enumerable.Range(1, 75).Select(x => new Order()
    {
      OrderID = 1000 + x,
      CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
    })[new Random().Next(5)],
      Freight = 2.1 * x,
      OrderDate = DateTime.Now.AddDays(-x),
      Verified = (new bool[] { true, false })[new Random().Next(2)]
    }).ToList();
  }
  private void IncrementCount()
  {
    grid.PreventRender();
    currentCount++;
  }
  public class Order
  {
    public int? OrderID { get; set; }
    public string CustomerID { get; set; }
    public DateTime? OrderDate { get; set; }
    public double? Freight { get; set; }
    public bool Verified { get; set; }
  }
}
```

Notes

- **PreventRender** method accepts boolean argument that accepts true or false to disable or enable rendering respectively.
- **PreventRender** method can be used only after grid component completed initial rendering. Calling this method during initial rendering will not have any effect.

Avoid unnecessary component renders after grid events

When a callback method is assigned to the grid events, then the **StateHasChanged** will be called in parent component of the grid automatically once the event is completed.

You can prevent this re-rendering of the grid component by setting **PreventRender** property of the corresponding event argument as true.

In the following example:

- **RowSelected** event is bound with a callback method, so once row selection event is completed the **StateHasChanged** will be invoked for the parent component.
- **RowSelectEventArgs.PreventRender** is set as **true** so now grid will not be part of the **StateHasChanged** invoked as result of the grid.

ASPX-CS

```
<p>Selected OrderID: <span
style="color:red">@SelectedOrder.OrderID</span></p>
<SfGrid @ref="grid" DataSource="@Orders">
  <GridSelectionSettings PersistSelection="true"></GridSelectionSettings>
  <GridColumns>
    <GridColumn Type="ColumnType.CheckBox" Width="50"></GridColumn>
    <GridColumn Field=@nameof(Order.OrderID) IsPrimaryKey="true"
HeaderText="Order ID" TextAlign="TextAlign.Right" Width="120"></GridColumn>
    <GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>
    <GridColumn Field=@nameof(Order.OrderDate) HeaderText="Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
    <GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
    <GridColumn Field="@nameof(Order.Verified)" DisplayAsCheckBox="true"
Width="70"></GridColumn>
  </GridColumns>
  <GridEvents TValue="Order" RowSelected="OnRowSelected"></GridEvents>
</SfGrid>
@code {
  SfGrid<Order> grid { get; set; }
  public List<Order> Orders { get; set; }
  Order SelectedOrder = new Order { };
  protected override void OnInitialized()
  {
    Orders = Enumerable.Range(1, 75).Select(x => new Order()
    {
      OrderID = 1000 + x,
      CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
    })[new Random().Next(5)],
```

```
Freight = 2.1 * x,
OrderDate = DateTime.Now.AddDays(-x),
Verified = (new bool[] { true, false })[new Random().Next(2)]
}).ToList();
}
private void OnRowSelected(RowSelectEventArgs<Order> args)
{
    args.PreventRender = true; //without this, you may see noticable delay in
    selection with 75 rows in grid.
    SelectedOrder = args.Data;
}
public class Order
{
    public int? OrderID { get; set; }
    public string CustomerID { get; set; }
    public DateTime? OrderDate { get; set; }
    public double? Freight { get; set; }
    public bool Verified { get; set; }
}
}
```

Notes

- **PreventRender** property internally overrides the **ShouldRender** method of the grid to prevent rendering.
- It is recommended to set **PreventRender** as true for user interactive events such as RowSelected, RowSelecting etc. for better performance.
- For events without any argument such as **DataBound**, you can use **PreventRender** method of the grid to disable rendering.

Use paging or virtualization to load only visible rows

Grid renders each row and cell as individual component and loading large number of rows and cells in view will have performance impact on both memory consumption and CPU processing.

To use grid without such performance impacts, you can load reduced set of rows in the grid using [paging](#) and [virtualization](#) features.

Even though with paging or virtualization feature enabled, having hundreds of rows in single grid page will again introduce performance lag in the application, so you need to set reasonable page size.

Events in Blazor DataGrid Component

In this section, we have provided the list of events of the datagrid component which will be triggered for appropriate datagrid actions.

The events should be provided to the datagrid using **GridEvents** component. When using events of datagrid, **TValue** must be provided in the **GridEvents** component.

All the events should be provided in a single **GridEvents** component.

OnActionBegin

[OnActionBegin](#) event triggers when DataGrid actions such as sorting, filtering, paging, grouping, [editing](#) etc., starts.

ASPX-CS

```

@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders" AllowPaging="true">
  <GridEvents OnActionBegin="ActionBeginHandler" TValue="Order"></GridEvents>
  <GridColumns>
    <GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
      TextAlign="TextAlign.Right" Width="120"></GridColumn>
    <GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
      Width="150"></GridColumn>
    <GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
      Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
      Width="130"></GridColumn>
    <GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
      TextAlign="TextAlign.Right" Width="120"></GridColumn>
  </GridColumns>
</SfGrid>
@code{
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
  Orders = Enumerable.Range(1, 75).Select(x => new Order()
  {
    OrderID = 1000 + x,
    CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
  })[new Random().Next(5)],
    Freight = 2.1 * x,
    OrderDate = DateTime.Now.AddDays(-x),
  }).ToList();
}
public void ActionBeginHandler(ActionEventArgs<Order> args)
{
  // Here you can customize your code
}
public class Order
{
  public int? OrderID { get; set; }
  public string CustomerID { get; set; }
  public DateTime? OrderDate { get; set; }
  public double? Freight { get; set; }
}
}

```

OnActionComplete

[OnActionComplete](#) event triggers when DataGrid actions such as sorting, filtering, paging, grouping, [editing](#) etc. are completed.

ASPX-CS

```

@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders">
  <GridEvents OnActionComplete="ActionCompletedHandler"
    TValue="Order"></GridEvents>
  <GridColumns>
    <GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
      TextAlign="TextAlign.Right" Width="120"></GridColumn>

```

```

<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumns>
</SfGrid>
@code{
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
OrderDate = DateTime.Now.AddDays(-x),
}).ToList();
}
public class Order {
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
public void ActionCompletedHandler(ActionEventArgs<Order> args)
{
// Here you can customize your code
}
}

```

OnActionFailure

[OnActionFailure](#) event triggers when any DataGrid action failed to achieve the desired results. By using this event you can get the error details and its cause. In the below sample we have provided the wrong url so that it will throw the OnActionFailure event.

ASPX-CS

```

@using Syncfusion.Blazor
@using Syncfusion.Blazor.Data
@using Syncfusion.Blazor.Grids
<SfGrid TValue="Order">
<GridEvents OnActionFailure="ActionFailureHandler"
TValue="Order"></GridEvents>
<SfDataManager Url="https://ej2services.syncfusion.com/production/web-
services/api/Orderss" Adaptor="Adaptors.WebApiAdaptor"></SfDataManager>
<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>

```



```

<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumn>
</SfGrid>
@code{
public void ActionFailureHandler(FailureEventArgs args)
{
// Here you can get the error details in the args
}
public class Order {
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
}

```

BeforeOpenColumnChooser

[BeforeOpenColumnChooser](#) event triggers before ColumnChooser gets opened.

ASPX-CS

```

@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Employees" ShowColumnChooser="true"
Toolbar=@ToolbarItems>
<GridEvents BeforeOpenColumnChooser="BeforeOpenColumnChooserHandler"
TValue="EmployeeData"></GridEvents>
<GridColumn>
<GridColumn Field=@nameof(EmployeeData.EmployeeID)
TextAlign="TextAlign.Center" HeaderText="Employee ID"
Width="120"></GridColumn>
<GridColumn Field=@nameof(EmployeeData.FirstName) HeaderText="First Name"
ShowInColumnChooser="false" Width="130"></GridColumn>
<GridColumn Field=@nameof(EmployeeData.LastName) HeaderText="Last Name"
Width="130"></GridColumn>
<GridColumn Field=@nameof(EmployeeData.Title) HeaderText="Title"
Width="120"></GridColumn>
<GridColumn Field=@nameof(EmployeeData.HireDate) HeaderText="Hire Date"
Format="d" TextAlign="TextAlign.Right" Width="150"></GridColumn>
</GridColumn>
</SfGrid>
@code{
public string[] ToolbarItems = new string[] { "ColumnChooser" };
public List<EmployeeData> Employees { get; set; }
protected override void OnInitialized()
{
Employees = Enumerable.Range(1, 9).Select(x => new EmployeeData()
{
EmployeeID = x,
FirstName = (new string[] { "Nancy", "Andrew", "Janet", "Margaret", "Steven"
})[new Random().Next(5)],
LastName = (new string[] { "Davolio", "Fuller", "Leverling", "Peacock",
"Buchanan" })[new Random().Next(5)],

```

```

Title = (new string[] { "Sales Representative", "Vice President, Sales",
"Sales Manager",
"Inside Sales Coordinator" })[new Random().Next(4)],
HireDate = DateTime.Now.AddDays(-x),
}).ToList();
}
public class EmployeeData
{
public int? EmployeeID { get; set; }
public string FirstName { get; set; }
public string LastName { get; set; }
public string Title { get; set; }
public DateTime? HireDate { get; set; }
}
public void BeforeOpenColumnChooserHandler(ColumnChooserEventArgs Args)
{
//customize your code here
}
}

```

Created

[Created](#) event triggers when the datagrid component is created. You can able to modify the datagrid properties by using this event.

ASPX-CS

```

@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders">
<GridEvents Created="CreatedHandler" TValue="Order"></GridEvents>
<GridColumn>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumn>
</SfGrid>
@code{
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
OrderDate = DateTime.Now.AddDays(-x),
}).ToList();
}
public class Order {
public int? OrderID { get; set; }

```

```

public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
public void CreatedHandler(object args)
{
    // Here you can customize your code
}
}

```

OnLoad

[OnLoad](#) event triggers before the rendering process starts which allows customization of DataGrid properties before the DataGrid rendering.

ASPX-CS

```

@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders">
    <GridEvents OnLoad="LoadHandler" TValue="Order"></GridEvents>
    <GridColumn>
        <GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
            TextAlign="TextAlign.Right" Width="120"></GridColumn>
        <GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
            Width="150"></GridColumn>
        <GridColumn Field=@nameof(Order.OrderDate) HeaderText="Order Date"
            Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
            Width="130"></GridColumn>
        <GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
            TextAlign="TextAlign.Right" Width="120"></GridColumn>
    </GridColumn>
</SfGrid>
@code{
    public List<Order> Orders { get; set; }
    protected override void OnInitialized()
    {
        Orders = Enumerable.Range(1, 75).Select(x => new Order()
        {
            OrderID = 1000 + x,
            CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
            })[new Random().Next(5)],
            Freight = 2.1 * x,
            OrderDate = DateTime.Now.AddDays(-x),
        }).ToList();
    }
    public class Order {
        public int? OrderID { get; set; }
        public string CustomerID { get; set; }
        public DateTime? OrderDate { get; set; }
        public double? Freight { get; set; }
    }
    public void LoadHandler(object args)
    {
        // Here you can customize your code
    }
}

```

Destroyed

[Destroyed](#) event triggers when the datagrid component is destroyed. By using this event you can confirm that the datagrid gets completely destroyed.

ASPX-CS

```
@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders">
  <GridEvents Destroyed="DestroyHandler" TValue="Order"></GridEvents>
  <GridColumns>
    <GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
      TextAlign="TextAlign.Right" Width="120"></GridColumn>
    <GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
      Width="150"></GridColumn>
    <GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
      Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
      Width="130"></GridColumn>
    <GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
      TextAlign="TextAlign.Right" Width="120"></GridColumn>
  </GridColumns>
</SfGrid>
@code{
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
  Orders = Enumerable.Range(1, 75).Select(x => new Order()
  {
    OrderID = 1000 + x,
    CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
  })[new Random().Next(5)],
    Freight = 2.1 * x,
    OrderDate = DateTime.Now.AddDays(-x),
  }).ToList();
}
public class Order {
  public int? OrderID { get; set; }
  public string CustomerID { get; set; }
  public DateTime? OrderDate { get; set; }
  public double? Freight { get; set; }
}
public void DestroyHandler(object args)
{
  // Here you can customize your code
}
}
```

OnDataBound

[OnDataBound](#) event triggers before data is bound to DataGrid.

ASPX-CS

```
@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders">
  <GridEvents OnDataBound="DataBoundHandler" TValue="Order"></GridEvents>
  <GridColumns>
```

```

<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumns>
</SfGrid>
@code{
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
OrderDate = DateTime.Now.AddDays(-x),
}).ToList();
}
public class Order {
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
public void DataBoundHandler(BeforeDataBoundArgs<Order> args)
{
// Here you can customize your code
}
}

```

DataBound

[DataBound](#) event triggers when data source is populated in the DataGrid. You can able to customize your code.

ASPX-CS

```

@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders">
<GridEvents DataBound="DataBoundHandler" TValue="Order"></GridEvents>
<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumns>

```

```

</SfGrid>
@code{
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
OrderDate = DateTime.Now.AddDays(-x),
}).ToList();
}
public class Order {
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
public void DataBoundHandler()
{
// Here you can customize your code
}
}

```

RowDataBound

[RowDataBound](#) event triggers every time a request is made to access row information, element, or data and also before the row element is appended to the DataGrid element.

ASPX-CS

```

@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders">
<GridEvents RowDataBound="RowDataBoundHandler" TValue="Order"></GridEvents>
<GridColumn>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText="Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumn>
</SfGrid>
@code{
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],

```

```

Freight = 2.1 * x,
OrderDate = DateTime.Now.AddDays(-x),
}).ToList();
}
public class Order {
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
public void RowDataBoundHandler(RowDataBoundEventArgs<Order> args)
{
// Here you can customize your code
}
}

```

DetailDataBound

[DetailDataBound](#) event triggers after detail row expands.

ASPX-CS

```

@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Employees">
<GridEvents DetailDataBound="DetailDataBoundHandler"
TValue="EmployeeData"></GridEvents>
<GridTemplates>
<DetailTemplate>
@{
var employee = (context as EmployeeData);
<table class="detailtable" width="100%">
<colgroup>
<col width="35%">
<col width="35%">
<col width="30%">
</colgroup>
<tbody>
<tr>
<td rowspan="4" style="text-align: center;">

</td>
<td>
<span style="font-weight: 500;">Employee ID: </span> @employee.FirstName
</td>
<td>
<span style="font-weight: 500;">Hire Date: </span>
@employee.HireDate.Value.ToShortDateString()
</td>
</tr>
<tr>
<td>
<span style="font-weight: 500;">Last Name: </span> @employee.LastName
</td>
<td>
<span style="font-weight: 500;">City: </span> @employee.City

```

```

</td>
</tr>
<tr>
<td>
<span style="font-weight: 500;">Title: </span> @employee.Title
</td>
<td>
<span style="font-weight: 500;">Country: </span> @employee.Country
</td>
</tr>
</tbody>
</table>
}
</DetailTemplate>
</GridTemplates>
<GridColumns>
<GridColumn Field=@nameof(EmployeeData.FirstName) HeaderText="First Name"
Width="110"> </GridColumn>
<GridColumn Field=@nameof(EmployeeData.LastName) HeaderText="Last Name"
Width="110"></GridColumn>
<GridColumn Field=@nameof(EmployeeData.Title) HeaderText="Title"
Width="110"></GridColumn>
<GridColumn Field=@nameof(EmployeeData.Country) HeaderText="Country"
Width="110"></GridColumn>
</GridColumns>
</SfGrid>
<style type="text/css" class="cssStyles">
.detailtable td {
font-size: 13px;
padding: 4px;
max-width: 0;
overflow: hidden;
text-overflow: ellipsis;
white-space: nowrap;
}
.photo {
width: 100px;
height: 100px;
border-radius: 50px;
box-shadow: inset 0 0 1px #e0e0e0, inset 0 0 14px rgba(0,0,0,0.2);
}
</style>
@code{
public List<EmployeeData> Employees { get; set; }
protected override void OnInitialized()
{
Employees = Enumerable.Range(1, 9).Select(x => new EmployeeData()
{
EmployeeID = x,
FirstName = (new string[] { "Nancy", "Andrew", "Janet", "Margaret", "Steven"
})[new Random().Next(5)],
LastName = (new string[] { "Davolio", "Fuller", "Leverling", "Peacock",
"Buchanan" })[new Random().Next(5)],
Title = (new string[] { "Sales Representative", "Vice President, Sales",
"Sales Manager",
"Inside Sales Coordinator" })[new Random().Next(4)],
HireDate = DateTime.Now.AddDays(-x),

```



```

City = (new string[] { "Seattle", "Tacoma", "Redmond", "Kirkland", "London"
})[new Random().Next(5)],
Country = (new string[] { "USA", "UK" })[new Random().Next(2)],
}).ToList();
}
public class EmployeeData
{
public int? EmployeeID { get; set; }
public string FirstName { get; set; }
public string LastName { get; set; }
public string Title { get; set; }
public DateTime? HireDate { get; set; }
public string City { get; set; }
public string Country { get; set; }
}
public void DetailDataBoundHandler(DetailDataBoundEventArgs<EmployeeData>
args)
{
// Here you can customize your code
}
}

```

HeaderCellInfo

[HeaderCellInfo](#) event triggers during the rendering of every header cells in the DataGrid so that you can able to customize the header cells.

ASPX-CS

```

@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders">
<GridEvents HeaderCellInfo="HeaderCellInfoHandler"
TValue="Order"></GridEvents>
<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumns>
</SfGrid>
@code{
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
OrderDate = DateTime.Now.AddDays(-x),
}).ToList();
}
}

```

```

}
public class Order {
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
public void HeaderCellInfoHandler(HeaderCellInfoEventArgs args)
{
// Here you can customize your code
}
}

```

QueryCellInfo

[QueryCellInfo](#) event triggers every time a request is made to access cell information, element, or data and also before the cell element is appended to the DataGrid element.

ASPX-CS

```

@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders">
<GridEvents QueryCellInfo="QueryCellInfoHandler"
TValue="Order"></GridEvents>
<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumns>
</SfGrid>
@code{
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
OrderDate = DateTime.Now.AddDays(-x),
}).ToList();
}
public class Order {
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
public void QueryCellInfoHandler(QueryCellInfoEventArgs<Order> args)
{

```

```
// Here you can customize your code
}
}
```

OnBeginEdit

[OnBeginEdit](#) event triggers before the record is to be edit.

ASPX-CS

```
@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders">
  <GridEvents OnBeginEdit="BeginEditHandler" TValue="Order"></GridEvents>
  <GridEditSettings AllowAdding="true" AllowEditing="true"
    AllowDeleting="true" Mode="EditMode.Normal"></GridEditSettings>
  <GridColumn>
    <GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
      IsPrimaryKey="true" TextAlign="TextAlign.Right" Width="120"></GridColumn>
    <GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
      Width="150"></GridColumn>
    <GridColumn Field=@nameof(Order.OrderDate) HeaderText="Order Date"
      Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
      Width="130"></GridColumn>
    <GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
      TextAlign="TextAlign.Right" Width="120"></GridColumn>
  </GridColumn>
</SfGrid>
@code{
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
    Orders = Enumerable.Range(1, 75).Select(x => new Order()
    {
        OrderID = 1000 + x,
        CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
    })[new Random().Next(5)],
        Freight = 2.1 * x,
        OrderDate = DateTime.Now.AddDays(-x),
    }).ToList();
}
public class Order {
    public int? OrderID { get; set; }
    public string CustomerID { get; set; }
    public DateTime? OrderDate { get; set; }
    public double? Freight { get; set; }
}
public void BeginEditHandler(BeginEditArgs<Order> args)
{
    // Here you can customize your code
}
}
```

OnBatchAdd

[OnBatchAdd](#) event triggers before records are added in batch mode.

ASPX-CS

```

@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders" Toolbar="@ (new List<string>() { "Add",
"Delete", "Update", "Cancel" })">
<GridEvents OnBatchAdd="BatchAddHandler" TValue="Order"></GridEvents>
<GridEditSettings AllowAdding="true" AllowDeleting="true"
AllowEditing="true" Mode="EditMode.Batch"></GridEditSettings>
<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
IsPrimaryKey="true" TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText="Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumns>
</SfGrid>
@code{
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
OrderDate = DateTime.Now.AddDays(-x),
}).ToList();
}
public class Order {
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
public void BatchAddHandler(BeforeBatchAddArgs<Order> args)
{
// Here you can customize your code
}
}

```

OnBatchSave

[OnBatchSave](#) event triggers before records are saved in batch mode.

ASPX-CS

```

@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders" Toolbar="@ (new List<string>() { "Add",
"Delete", "Update", "Cancel" })">
<GridEvents OnBatchSave="BatchSaveHandler" TValue="Order"></GridEvents>
<GridEditSettings AllowAdding="true" AllowDeleting="true"
AllowEditing="true" Mode="EditMode.Batch"></GridEditSettings>
<GridColumns>

```

```

<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
IsPrimaryKey="true" TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumns>
</SfGrid>
@code{
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
OrderDate = DateTime.Now.AddDays(-x),
}).ToList();
}
public class Order
{
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
public void BatchSaveHandler(BeforeBatchSaveArgs<Order> args)
{
// Here you can customize your code
}
}

```

OnBatchDelete

[OnBatchDelete](#) event triggers before records are deleted in batch mode.

ASPX-CS

```

@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders" Toolbar="@ (new List<string>() { "Add",
"Delete", "Update", "Cancel" })">
<GridEvents OnBatchDelete="BatchDeleteHandler" TValue="Order"></GridEvents>
<GridEditSettings AllowAdding="true" AllowDeleting="true"
AllowEditing="true" Mode="EditMode.Batch"></GridEditSettings>
<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
IsPrimaryKey="true" TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>

```

```

<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumn>
</SfGrid>
@code{
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
OrderDate = DateTime.Now.AddDays(-x),
}).ToList();
}
public class Order {
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
public void BatchDeleteHandler(BeforeBatchDeleteArgs<Order> args)
{
// Here you can customize your code
}
}

```

OnCellEdit

[OnCellEdit](#) event triggers when the cell is being edited.

ASPX-CS

```

@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders">
<GridEvents OnCellEdit="CellEditHandler" TValue="Order"></GridEvents>
<GridEditSettings AllowAdding="true" AllowEditing="true"
AllowDeleting="true" Mode="EditMode.Batch"></GridEditSettings>
<GridSelectionSettings Mode=SelectionMode.Cell></GridSelectionSettings>
<GridColumn>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
IsPrimaryKey="true" TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumn>
</SfGrid>
@code{
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{

```

```

Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
    OrderID = 1000 + x,
    CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
    })[new Random().Next(5)],
    Freight = 2.1 * x,
    OrderDate = DateTime.Now.AddDays(-x),
}).ToList();
}
public class Order {
    public int? OrderID { get; set; }
    public string CustomerID { get; set; }
    public DateTime? OrderDate { get; set; }
    public double? Freight { get; set; }
}
public void CellEditHandler(CellEditArgs<Order> args)
{
    // Here you can customize your code
}
}

```

OnCellSave

[OnCellSave](#) event triggers before saving the cell.

ASPX-CS

```

@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders">
    <GridEvents OnCellSave="CellSaveHandler" TValue="Order"></GridEvents>
    <GridEditSettings AllowAdding="true" AllowEditing="true"
    AllowDeleting="true" Mode="EditMode.Batch"></GridEditSettings>
    <GridSelectionSettings Mode="SelectionMode.Cell"></GridSelectionSettings>
    <GridColumn>
        <GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
        IsPrimaryKey="true" TextAlign="TextAlign.Right" Width="120"></GridColumn>
        <GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
        Width="150"></GridColumn>
        <GridColumn Field=@nameof(Order.OrderDate) HeaderText="Order Date"
        Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
        Width="130"></GridColumn>
        <GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
        TextAlign="TextAlign.Right" Width="120"></GridColumn>
    </GridColumn>
</SfGrid>
@code{
    public List<Order> Orders { get; set; }
    protected override void OnInitialized()
    {
        Orders = Enumerable.Range(1, 75).Select(x => new Order()
        {
            OrderID = 1000 + x,
            CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
            })[new Random().Next(5)],
            Freight = 2.1 * x,
            OrderDate = DateTime.Now.AddDays(-x),
        }).ToList();
    }
}

```

```

}
public class Order {
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
public void CellSaveHandler(CellSaveArgs<Order> args)
{
// Here you can customize your code
}
}

```

CellSaved

[CellSaved](#) event triggers when cell is saved.

ASPX-CS

```

@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders">
  <GridEvents CellSaved="CellSavedHandler" TValue="Order"></GridEvents>
  <GridEditSettings AllowAdding="true" AllowEditing="true"
  AllowDeleting="true" Mode="EditMode.Batch"></GridEditSettings>
  <GridColumns>
    <GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
    IsPrimaryKey="true" TextAlign="TextAlign.Right" Width="120"></GridColumn>
    <GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
    Width="150"></GridColumn>
    <GridColumn Field=@nameof(Order.OrderDate) HeaderText="Order Date"
    Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
    Width="130"></GridColumn>
    <GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
    TextAlign="TextAlign.Right" Width="120"></GridColumn>
  </GridColumns>
</SfGrid>
@code{
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
OrderDate = DateTime.Now.AddDays(-x),
}).ToList();
}
public class Order {
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
public void CellSavedHandler(CellSaveArgs<Order> args)
{

```



```
// Here you can customize your code
}
}
```

RowSelecting

[RowSelecting](#) event triggers before row selection occurs.

ASPX-CS

```
@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders">
  <GridEvents RowSelecting="RowSelectingHandler" TValue="Order"></GridEvents>
  <GridColumn>
    <GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
      TextAlign="TextAlign.Right" Width="120"></GridColumn>
    <GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
      Width="150"></GridColumn>
    <GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
      Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
      Width="130"></GridColumn>
    <GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
      TextAlign="TextAlign.Right" Width="120"></GridColumn>
  </GridColumn>
</SfGrid>
@code{
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
  Orders = Enumerable.Range(1, 75).Select(x => new Order()
  {
    OrderID = 1000 + x,
    CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
  })[new Random().Next(5)],
    Freight = 2.1 * x,
    OrderDate = DateTime.Now.AddDays(-x),
  }).ToList();
}
public class Order {
  public int? OrderID { get; set; }
  public string CustomerID { get; set; }
  public DateTime? OrderDate { get; set; }
  public double? Freight { get; set; }
}
public void RowSelectingHandler(RowSelectingEventArgs<Order> args)
{
  // Here you can customize your code
}
}
```

RowSelected

[RowSelected](#) event triggers when a row is selected.

ASPX-CS

```
@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders">
```

```

<GridEvents RowSelected="RowSelectHandler" TValue="Order"></GridEvents>
<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumns>
</SfGrid>
@code{
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
OrderDate = DateTime.Now.AddDays(-x),
}).ToList();
}
public class Order {
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
public void RowSelectHandler(RowSelectEventArgs<Order> args)
{
// Here you can customize your code
}
}

```

RowDeselecting

[RowDeselecting](#) event triggers before a selected row is being deselected.

ASPX-CS

```

@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders">
<GridEvents RowDeselecting="RowDeselectingHandler"
TValue="Order"></GridEvents>
<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>

```

```

<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumn>
</SfGrid>
@code{
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
OrderDate = DateTime.Now.AddDays(-x),
}).ToList();
}
public class Order
{
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
public void RowDeselectingHandler(RowDeselectEventArgs<Order> args)
{
// Here you can customize your code
}
}

```

RowDeselected

[RowDeselected](#) event triggers when a selected row is deselected.

ASPX-CS

```

@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders">
<GridEvents RowDeselected="RowDeselectHandler" TValue="Order"></GridEvents>
<GridColumn>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumn>
</SfGrid>
@code{
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{

```

```

    OrderID = 1000 + x,
    CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
    })[new Random().Next(5)],
    Freight = 2.1 * x,
    OrderDate = DateTime.Now.AddDays(-x),
    }).ToList();
}

public class Order {
    public int? OrderID { get; set; }
    public string CustomerID { get; set; }
    public DateTime? OrderDate { get; set; }
    public double? Freight { get; set; }
}

public void RowDeselectHandler(RowDeselectEventArgs<Order> args)
{
    // Here you can customize your code
}
}

```

CellSelecting

[CellSelecting](#) event triggers before cell selection occurs.

ASPX-CS

```

@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders">
    <GridEvents CellSelecting="CellSelectingHandler"
    TValue="Order"></GridEvents>
    <GridEditSettings AllowAdding="true" AllowEditing="true"
    AllowDeleting="true" Mode="EditMode.Batch"></GridEditSettings>
    <GridSelectionSettings Mode=SelectionMode.Cell></GridSelectionSettings>
    <GridColumns>
        <GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
        IsPrimaryKey="true" TextAlign="TextAlign.Right" Width="120"></GridColumn>
        <GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
        Width="150"></GridColumn>
        <GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
        Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
        Width="130"></GridColumn>
        <GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
        TextAlign="TextAlign.Right" Width="120"></GridColumn>
    </GridColumns>
</SfGrid>
@code{
    public List<Order> Orders { get; set; }
    protected override void OnInitialized()
    {
        Orders = Enumerable.Range(1, 75).Select(x => new Order()
        {
            OrderID = 1000 + x,
            CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
            })[new Random().Next(5)],
            Freight = 2.1 * x,
            OrderDate = DateTime.Now.AddDays(-x),
            }).ToList();
    }
}

```

```

public class Order {
    public int? OrderID { get; set; }
    public string CustomerID { get; set; }
    public DateTime? OrderDate { get; set; }
    public double? Freight { get; set; }
}

public void CellSelectingHandler(CellSelectingEventArgs<Order> args)
{
    // Here you can customize your code
}
}

```

CellSelected

[CellSelected](#) event triggers after a cell is selected.

ASPX-CS

```

@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders">
    <GridEvents CellSelected="CellSelectedHandler" TValue="Order"></GridEvents>
    <GridSelectionSettings Mode=SelectionMode.Cell></GridSelectionSettings>
    <GridColumns>
        <GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
            IsPrimaryKey="true" TextAlign="TextAlign.Right" Width="120"></GridColumn>
        <GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
            Width="150"></GridColumn>
        <GridColumn Field=@nameof(Order.OrderDate) HeaderText="Order Date"
            Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
            Width="130"></GridColumn>
        <GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
            TextAlign="TextAlign.Right" Width="120"></GridColumn>
    </GridColumns>
</SfGrid>

@code{
    public List<Order> Orders { get; set; }
    protected override void OnInitialized()
    {
        Orders = Enumerable.Range(1, 75).Select(x => new Order()
        {
            OrderID = 1000 + x,
            CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
            })[new Random().Next(5)],
            Freight = 2.1 * x,
            OrderDate = DateTime.Now.AddDays(-x),
        }).ToList();
    }

    public class Order {
        public int? OrderID { get; set; }
        public string CustomerID { get; set; }
        public DateTime? OrderDate { get; set; }
        public double? Freight { get; set; }
    }

    public void CellSelectedHandler(CellSelectEventArgs<Order> args)
    {
        // Here you can customize your code
    }
}

```

```
}

```

CellDeselecting

[CellDeselecting](#) event triggers before cell is deselected.

ASPX-CS

```
@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders">
  <GridEvents CellDeselecting="CellDeselectingHandler"
  TValue="Order"></GridEvents>
  <GridEditSettings AllowAdding="true" AllowEditing="true"
  AllowDeleting="true" Mode="EditMode.Batch"></GridEditSettings>
  <GridSelectionSettings Mode=SelectionMode.Cell></GridSelectionSettings>
  <GridColumn>
    <GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
    IsPrimaryKey="true" TextAlign="TextAlign.Right" Width="120"></GridColumn>
    <GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
    Width="150"></GridColumn>
    <GridColumn Field=@nameof(Order.OrderDate) HeaderText="Order Date"
    Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
    Width="130"></GridColumn>
    <GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
    TextAlign="TextAlign.Right" Width="120"></GridColumn>
  </GridColumn>
</SfGrid>
@code{
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
  Orders = Enumerable.Range(1, 75).Select(x => new Order()
  {
    OrderID = 1000 + x,
    CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
  })[new Random().Next(5)],
    Freight = 2.1 * x,
    OrderDate = DateTime.Now.AddDays(-x),
  }).ToList();
}
public class Order
{
  public int? OrderID { get; set; }
  public string CustomerID { get; set; }
  public DateTime? OrderDate { get; set; }
  public double? Freight { get; set; }
}
public void CellDeselectingHandler(CellDeselectEventArgs<Order> args)
{
  // Here you can customize your code
}
}
```

CellDeselected

[CellDeselected](#) event triggers after a cell is deselected.

ASPX-CS

```

@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders">
<GridEvents CellDeselected="CellDeselectHandler"
TValue="Order"></GridEvents>
<GridEditSettings AllowAdding="true" AllowEditing="true"
AllowDeleting="true" Mode="EditMode.Batch"></GridEditSettings>
<GridSelectionSettings Mode=SelectionMode.Cell></GridSelectionSettings>
<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
IsPrimaryKey="true" TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumns>
</SfGrid>
@code{
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
OrderDate = DateTime.Now.AddDays(-x),
}).ToList();
}
public class Order
{
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
public void CellDeselectHandler(CellDeselectEventArgs<Order> args)
{
// Here you can customize your code
}
}

```

OnRecordClick

[OnRecordClick](#) event triggers when record is clicked.

ASPX-CS

```

@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders">
<GridEditSettings AllowAdding="true" AllowEditing="true"
AllowDeleting="true" Mode="EditMode.Normal"></GridEditSettings>

```

```

<GridEvents OnRecordClick="RecordClickHandler" TValue="Order"></GridEvents>
<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
IsPrimaryKey="true" TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumns>
</SfGrid>
@code{
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
OrderDate = DateTime.Now.AddDays(-x),
}).ToList();
}
public class Order
{
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
public void RecordClickHandler(RecordClickEventArgs<Order> args)
{
// Here you can customize your code
}
}

```

OnRecordDoubleClick

[OnRecordDoubleClick](#) event triggers when record is double clicked.

ASPX-CS

```

@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders">
<GridEditSettings AllowAdding="true" AllowEditing="true"
AllowDeleting="true"
Mode="EditMode.Normal"></GridEditSettings>
<GridEvents OnRecordDoubleClick="RecordDoubleClickHandler"
TValue="Order"></GridEvents>
<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
IsPrimaryKey="true" TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>

```



```

<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumns>
</SfGrid>
@code{
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
OrderDate = DateTime.Now.AddDays(-x),
}).ToList();
}
public class Order {
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
public void RecordDoubleClickHandler(RecordDoubleClickEventArgs<Order> args)
{
// Here you can customize your code
}
}

```

OnToolbarClick

[OnToolbarClick](#) event triggers when toolbar item is clicked.

ASPX-CS

```

@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders" Toolbar="@ (new List<string>() { "Add", "Edit",
"Delete", "Cancel", "Update" })">
<GridEvents OnToolbarClick="ToolbarClickHandler"
TValue="Order"></GridEvents>
<GridEditSettings AllowAdding="true" AllowEditing="true"
AllowDeleting="true" Mode="EditMode.Normal"></GridEditSettings>
<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
IsPrimaryKey="true" TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumns>
</SfGrid>

```

```
@code{
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
OrderDate = DateTime.Now.AddDays(-x),
}).ToList();
}
public class Order {
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
public void ToolbarClickHandler(Syncfusion.Blazor.Navigations.ClickEventArgs
args)
{
// Here you can customize your code
}
}
```

CommandClicked

[CommandClicked](#) event triggers when command button is clicked. It provides the row data of the currently clicked row.

ASPX-CS

```
@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders" AllowPaging="true" Height="315">
<GridEvents CommandClicked="OnCommandClicked" TValue="Order"></GridEvents>
<GridEditSettings AllowAdding="true" AllowEditing="true"
AllowDeleting="true"></GridEditSettings>
<GridColumn>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
IsPrimaryKey="true" TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
EditType="EditType.DatePickerEdit" Format="d" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn HeaderText="Manage Records" Width="150">
<GridCommandColumns>
<GridCommandColumn Type="CommandButtonType.Edit" ButtonOption="@ (new
CommandButtonOptions() { IconCss = "e-icons e-edit", CssClass = "e-flat"
})"></GridCommandColumn>
<GridCommandColumn Type="CommandButtonType.Delete" ButtonOption="@ (new
CommandButtonOptions() { IconCss = "e-icons e-delete", CssClass = "e-flat"
})"></GridCommandColumn>
```

```

<GridCommandColumn Type="CommandButtonType.Save" ButtonOption="@ (new
CommandButtonOptions() { IconCss = "e-icons e-update", CssClass = "e-flat"
}) "></GridCommandColumn>
<GridCommandColumn Type="CommandButtonType.Cancel" ButtonOption="@ (new
CommandButtonOptions() { IconCss = "e-icons e-cancel-icon", CssClass = "e-
flat" }) "></GridCommandColumn>
</GridCommandColumns>
</GridColumn>
</GridColumns>
</SfGrid>
@code{
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
OrderDate = DateTime.Now.AddDays(-x),
ShipCountry = (new string[] { "USA", "UK", "CHINA", "RUSSIA", "INDIA" })[new
Random().Next(5)]
}).ToList();
}
public class Order
{
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
public string ShipCountry { get; set; }
}
public void OnCommandClicked(CommandClickEventArgs<Order> args)
{
// Perform required operations here
}
}

```

ColumnMenuItemClicked

[ColumnMenuItemClicked](#) event triggers when click on column menu.

ASPX-CS

```

@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders" AllowGrouping="true" AllowFiltering="true"
AllowPaging="true" ShowColumnMenu="true">
<GridEvents ColumnMenuItemClicked="ColumnMenuItemClickedHandler"
TValue="Order"></GridEvents>
<GridFilterSettings Type="FilterType.CheckBox"></GridFilterSettings>
<GridGroupSettings ShowGroupedColumn="true"></GridGroupSettings>
<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>

```

```

<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumn>
</SfGrid>
@code{
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
OrderDate = DateTime.Now.AddDays(-x),
}).ToList();
}
public class Order
{
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
public void ColumnMenuItemClickedHandler(ColumnMenuClickEventArgs args)
{
// Here you can customize your code
}
}

```

ContextMenuItemClicked

[ContextMenuItemClicked](#) event triggers when click on context menu.

ASPX-CS

```

@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders" AllowGrouping="true" AllowFiltering="true"
AllowPaging="true" ContextMenuItems="@ (new List<object>() { "AutoFit",
"AutoFitAll", "SortAscending", "SortDescending", "Copy", "Edit", "Delete",
"Save", "Cancel", "PdfExport", "ExcelExport", "CsvExport", "FirstPage",
"PrevPage", "LastPage", "NextPage" }) ">
<GridEvents ContextMenuItemClicked="ContextMenuItemClickedHandler"
TValue="Order"></GridEvents>
<GridFilterSettings
Type="@Syncfusion.Blazor.Grids.FilterType.CheckBox"></GridFilterSettings>
<GridGroupSettings ShowGroupedColumn="true"></GridGroupSettings>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>

```

```

<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumn>
</SfGrid>
@code{
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
OrderDate = DateTime.Now.AddDays(-x),
}).ToList();
}
public class Order
{
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
public void ContextMenuItemClickedHandler(ContextMenuClickEventArgs<Order>
args)
{
// Here you can customize your code
}
}

```

ContextMenuOpen

[ContextMenuOpen](#) event triggers before opening the context menu.

ASPX-CS

```

@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders" AllowGrouping="true" AllowFiltering="true"
AllowPaging="true" ContextMenuItems="@ (new List<object>() { "AutoFit",
"AutoFitAll", "SortAscending", "SortDescending", "Copy", "Edit", "Delete",
"Save", "Cancel", "PdfExport", "ExcelExport", "CsvExport", "FirstPage",
"PrevPage", "LastPage", "NextPage" })">
<GridEvents ContextMenuOpen="ContextMenuOpenHandler"
TValue="Order"></GridEvents>
<GridFilterSettings
Type="@Syncfusion.Blazor.Grids.FilterType.CheckBox"></GridFilterSettings>
<GridGroupSettings ShowGroupedColumn="true"></GridGroupSettings>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>

```

```

<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumns>
</SfGrid>
@code{
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
OrderDate = DateTime.Now.AddDays(-x),
}).ToList();
}
public class Order
{
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
public void ContextMenuOpenHandler(ContextMenuOpenEventArgs<Order> args)
{
// Here you can customize your code
}
}

```

OnPdfExport

[OnPdfExport](#) event triggers before DataGrid data is exported to PDF document.

ASPX-CS

```

@using Syncfusion.Blazor.Grids
<SfGrid @ref="Grid" DataSource="@Orders" AllowPdfExport="true"
Toolbar="@ (new List<string>() { "PdfExport" })">
<GridEvents OnPdfExport="PdfExportHandler"
OnToolbarClick="ToolbarClickHandler" TValue="Order"></GridEvents>
<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumns>
</SfGrid>
@code{

```

```

private SfGrid<Order> Grid;
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
    Orders = Enumerable.Range(1, 75).Select(x => new Order()
    {
        OrderID = 1000 + x,
        CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
    })[new Random().Next(5)],
        Freight = 2.1 * x,
        OrderDate = DateTime.Now.AddDays(-x),
    }).ToList();
}
public class Order
{
    public int? OrderID { get; set; }
    public string CustomerID { get; set; }
    public DateTime? OrderDate { get; set; }
    public double? Freight { get; set; }
}
public async Task
ToolbarClickHandler(Syncfusion.Blazor.Navigations.ClickEventArgs args)
{
    await this.Grid.PdfExport();
}
public void PdfExportHandler(object args)
{
    // Here you can customize your code
}
}

```

PdfHeaderQueryCellInfoEvent

[PdfHeaderQueryCellInfoEvent](#) event triggers before DataGrid data is exported to PDF document. It can be used to customize the header content in pdf document.

ASPX-CS

```

@using Syncfusion.Blazor.Grids
<SfGrid @ref="Grid" DataSource="@Orders" AllowPdfExport="true"
Toolbar="@ (new List<string>() { "PdfExport" })">
<GridEvents PdfHeaderQueryCellInfoEvent="PdfHeaderQueryCellInfoHandler"
OnToolbarClick="ToolbarClickHandler" TValue="Order"></GridEvents>
<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumns>
</SfGrid>
@code{
private SfGrid<Order> Grid;

```

```

public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
    Orders = Enumerable.Range(1, 75).Select(x => new Order()
    {
        OrderID = 1000 + x,
        CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
    })[new Random().Next(5)],
        Freight = 2.1 * x,
        OrderDate = DateTime.Now.AddDays(-x),
    }).ToList();
}
public class Order
{
    public int? OrderID { get; set; }
    public string CustomerID { get; set; }
    public DateTime? OrderDate { get; set; }
    public double? Freight { get; set; }
}
public async Task
ToolBarClickHandler(Syncfusion.Blazor.Navigations.ClickEventArgs args)
{
    await this.Grid.PdfExport();
}
public void PdfHeaderQueryCellInfoHandler(PdfHeaderQueryCellInfoEventArgs
args)
{
    // Here you can customize your code
}
}

```

PdfQueryCellInfoEvent

[PdfQueryCellInfoEvent](#) event triggers before DataGrid data is exported to PDF document. It can be used to customize the DataGrid content in pdf document.

ASPX-CS

```

@using Syncfusion.Blazor.Grids
<SfGrid @ref="Grid" DataSource="@Orders" AllowPdfExport="true"
ToolBar="@ (new List<string>() { "PdfExport" })">
<GridEvents PdfQueryCellInfoEvent="PdfQueryCellInfoHandler"
OnToolBarClick="ToolBarClickHandler" TValue="Order"></GridEvents>
<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumns>
</SfGrid>
@code{
private SfGrid<Order> Grid;

```



```

public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
    Orders = Enumerable.Range(1, 75).Select(x => new Order()
    {
        OrderID = 1000 + x,
        CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
    })[new Random().Next(5)],
        Freight = 2.1 * x,
        OrderDate = DateTime.Now.AddDays(-x),
    }).ToList();
}
public class Order
{
    public int? OrderID { get; set; }
    public string CustomerID { get; set; }
    public DateTime? OrderDate { get; set; }
    public double? Freight { get; set; }
}
public async Task
ToolbarClickHandler(Syncfusion.Blazor.Navigations.ClickEventArgs args)
{
    await this.Grid.PdfExport();
}
public void PdfQueryCellInfoHandler(PdfQueryCellInfoEventArgs<Order> args)
{
    // Here you can customize your code
}
}

```

PdfAggregateTemplateInfo

[PdfAggregateTemplateInfo](#) event triggers before DataGrid data is exported to PDF document. It can be used to customize the DataGrid aggregate content in pdf document.

ASPX-CS

```

@using Syncfusion.Blazor.Grids
<SfGrid @ref="Grid" DataSource="@Orders" AllowPdfExport="true"
Toolbar="@ (new List<string>() { "PdfExport" })">
<GridEvents PdfAggregateTemplateInfo="PdfAggregateTemplateInfoHandler"
OnToolbarClick="ToolbarClickHandler" TValue="Order"></GridEvents>
<GridAggregates>
<GridAggregate>
<GridAggregateColumns>
<GridAggregateColumn Field=@nameof(Order.Freight) Type="AggregateType.Sum"
Format="C2">
<FooterTemplate>
@{
var aggregate = (context as AggregateTemplateContext);
<div>
<p>Sum: @aggregate.Sum</p>
</div>
}
</FooterTemplate>
</GridAggregateColumn>
</GridAggregateColumns>

```

```

</GridAggregate>
<GridAggregate>
<GridAggregateColumns>
<GridAggregateColumn Field=@nameof(Order.Freight)
Type="AggregateType.Average" Format="C2">
<FooterTemplate>
@{
var aggregate = (context as AggregateTemplateContext);
<div>
<p>Average: @aggregate.Average</p>
</div>
}
</FooterTemplate>
</GridAggregateColumn>
</GridAggregateColumns>
</GridAggregate>
</GridAggregates>
<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumns>
</SfGrid>
@code{
private SfGrid<Order> Grid;
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
OrderDate = DateTime.Now.AddDays(-x),
}).ToList();
}
public class Order
{
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
public async Task
ToolbarClickHandler(Syncfusion.Blazor.Navigations.ClickEventArgs args)
{
await this.Grid.PdfExport();
}
public void PdfAggregateTemplateInfoHandler(PdfAggregateEventArgs<Order>
args)

```

```
{
// Here you can customize your code
}
```

OnExcelExport

[OnExcelExport](#) event triggers before DataGrid data is exported to excel file.

ASPX-CS

```
@using Syncfusion.Blazor.Grids
<SfGrid @ref="Grid" DataSource="@Orders" AllowExcelExport="true"
Toolbar="@ (new List<string>() { "ExcelExport" })">
<GridEvents OnExcelExport="ExcelExportHandler"
OnToolbarClick="ToolbarClickHandler" TValue="Order"></GridEvents>
<GridColumn>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText="Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumn>
</SfGrid>
@code{
private SfGrid<Order> Grid;
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
OrderDate = DateTime.Now.AddDays(-x),
}).ToList();
}
public class Order
{
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
public async Task
ToolbarClickHandler(Syncfusion.Blazor.Navigations.ClickEventArgs args)
{
await this.Grid.ExcelExport();
}
public void ExcelExportHandler(object args)
{
// Here you can customize your code
}
```

```
}
}
```

ExcelHeaderQueryCellInfoEvent

[ExcelHeaderQueryCellInfoEvent](#) event triggers before DataGrid data is exported to Excel file. It can be used to customize the header content in Excel file.

ASPX-CS

```
@using Syncfusion.Blazor.Grids
<SfGrid @ref="Grid" DataSource="@Orders" AllowExcelExport="true"
Toolbar="@ (new List<string>() { "ExcelExport" }) ">
<GridEvents ExcelHeaderQueryCellInfoEvent="ExcelHeaderQueryCellInfoHandler"
OnToolbarClick="ToolbarClickHandler" TValue="Order"></GridEvents>
<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumns>
</SfGrid>
@code{
private SfGrid<Order> Grid;
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
OrderDate = DateTime.Now.AddDays(-x),
}).ToList();
}
public class Order
{
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
public async Task
ToolbarClickHandler(Syncfusion.Blazor.Navigations.ClickEventArgs args)
{
await this.Grid.ExcelExport();
}
public void
ExcelHeaderQueryCellInfoHandler(ExcelHeaderQueryCellInfoEventArgs<Order>
args)
{
}
```

```
// Here you can customize your code
}
}
```

ExcelQueryCellInfoEvent

[ExcelQueryCellInfoEvent](#) event triggers before DataGrid data is exported to Excel file. It can be used to customize the DataGrid content in Excel file.

ASPX-CS

```
@using Syncfusion.Blazor.Grids
<SfGrid @ref="Grid" DataSource="@Orders" AllowExcelExport="true"
Toolbar="@ (new List<string>() { "ExcelExport" })">
<GridEvents ExcelQueryCellInfoEvent="ExcelQueryCellInfoHandler"
OnToolbarClick="ToolbarClickHandler" TValue="Order"></GridEvents>
<GridColumn>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumn>
</SfGrid>
@code{
private SfGrid<Order> Grid;
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
OrderDate = DateTime.Now.AddDays(-x),
}).ToList();
}
public class Order
{
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
public async Task
ToolbarClickHandler(Syncfusion.Blazor.Navigations.ClickEventArgs args)
{
await this.Grid.ExcelExport();
}
public void ExcelQueryCellInfoHandler(ExcelQueryCellInfoEventArgs<Order>
args)
{
}
```

```
// Here you can customize your code
}
}
```

ExcelAggregateTemplateInfo

[ExcelAggregateTemplateInfo](#) event triggers before DataGrid data is exported to Excel file. It can be used to customize the DataGrid aggregate content in Excel File.

ASPX-CS

```
@using Syncfusion.Blazor.Grids
<SfGrid @ref="Grid" DataSource="@Orders" AllowExcelExport="true"
Toolbar="@ (new List<string>() { "ExcelExport" })">
<GridEvents ExcelAggregateTemplateInfo="ExcelAggregateTemplateInfoHandler"
OnToolbarClick="ToolbarClickHandler" TValue="Order"></GridEvents>
<GridAggregates>
<GridAggregate>
<GridAggregateColumns>
<GridAggregateColumn Field=@nameof(Order.Freight) Type="AggregateType.Sum"
Format="C2">
<FooterTemplate>
@{
var aggregate = (context as AggregateTemplateContext);
<div>
<p>Sum: @aggregate.Sum</p>
</div>
}
</FooterTemplate>
</GridAggregateColumn>
</GridAggregateColumns>
</GridAggregate>
<GridAggregate>
<GridAggregateColumns>
<GridAggregateColumn Field=@nameof(Order.Freight)
Type="AggregateType.Average" Format="C2">
<FooterTemplate>
@{
var aggregate = (context as AggregateTemplateContext);
<div>
<p>Average: @aggregate.Average</p>
</div>
}
</FooterTemplate>
</GridAggregateColumn>
</GridAggregateColumns>
</GridAggregate>
</GridAggregates>
<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
```

```

<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumn>
</SfGrid>
@code{
private SfGrid<Order> Grid;
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
OrderDate = DateTime.Now.AddDays(-x),
}).ToList();
}
public class Order
{
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
public async Task
ToolbarClickHandler(Syncfusion.Blazor.Navigations.ClickEventArgs args)
{
await this.Grid.ExcelExport();
}
public void ExcelAggregateTemplateInfoHandler(ExcelAggregateEventArgs args)
{
// Here you can customize your code
}
}

```

ExportComplete

[ExportComplete](#) event triggers once DataGrid data is exported to file formats (Pdf, Excel and CSV).

ASPX-CS

```

@using Syncfusion.Blazor.Grids
<SfGrid @ref="Grid" DataSource="@Orders" AllowPdfExport="true"
AllowExcelExport="true" Toolbar="@ (new List<string>() {
"ExcelExport", "CsvExport", "PdfExport" }) ">
<GridEvents ExportComplete="ExportCompleteHandler"
OnToolbarClick="ToolbarClickHandler" TValue="Order"></GridEvents>
<GridColumn>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>

```

```

<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumn>
</SfGrid>
@code{
private SfGrid<Order> Grid;
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
OrderDate = DateTime.Now.AddDays(-x),
}).ToList();
}
public class Order
{
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
public async Task
ToolbarClickHandler(Syncfusion.Blazor.Navigations.ClickEventArgs args)
{
if (args.Item.Text == "Excel Export")
{
await this.Grid.ExcelExport();
}
if (args.Item.Text == "PDF Export")
{
await this.Grid.PdfExport();
}
if (args.Item.Text == "CSV Export")
{
await this.Grid.CsvExport();
}
}
public void ExportCompleteHandler(object args)
{
// Here you can customize your code
}
}

```

OnResizeStart

[OnResizeStart](#) event triggers when column resize starts.

ASPX-CS

```

@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders" AllowResizing="true">
<GridEvents OnResizeStart="OnResizeStartHanlder"
TValue="Order"></GridEvents>

```



```

<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumns>
</SfGrid>
@code{
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
OrderDate = DateTime.Now.AddDays(-x),
}).ToList();
}
public class Order
{
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
public void OnResizeStartHanlder(ResizeArgs args)
{
// Here you can customize your code
}
}

```

ResizeStopped

[ResizeStopped](#) event triggers when column resize ends.

ASPX-CS

```

@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders" AllowResizing="true">
<GridEvents ResizeStopped="ResizeStoppedHanlder"
TValue="Order"></GridEvents>
<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>

```

```

<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumn>
</SfGrid>
@code{
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
OrderDate = DateTime.Now.AddDays(-x),
}).ToList();
}
public class Order
{
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
public void ResizeStoppedHanlder(ResizeArgs args)
{
// Here you can customize your code
}
}

```

OnRowDragStart

[OnRowDragStart](#) event triggers when row drag starts.

ASPX-CS

```

@using Syncfusion.Blazor.Grids
<SfGrid AllowRowDragAndDrop="true" DataSource="@Orders">
<GridEvents OnRowDragStart="RowDragStartHandler"
TValue="Order"></GridEvents>
<GridColumn>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
IsPrimaryKey="true" TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumn>
</SfGrid>
@code{
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()

```

```

{
    OrderID = 1000 + x,
    CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
    })[new Random().Next(5)],
    Freight = 2.1 * x,
    OrderDate = DateTime.Now.AddDays(-x),
}).ToList();
}

public class Order
{
    public int? OrderID { get; set; }
    public string CustomerID { get; set; }
    public DateTime? OrderDate { get; set; }
    public double? Freight { get; set; }
}

public void RowDragStartHandler(RowDragEventArgs<Order> args)
{
    // Here you can customize your code
}
}

```

RowDropped

[RowDropped](#) event triggers when row is dropped.

We are not going to limit datagrid with these events, we will be adding new events in future based on the user requests. If the event, you are looking for is not in the list, then please request [here](#).

ASPX-CS

```

@using Syncfusion.Blazor.Grids
<SfGrid AllowRowDragAndDrop="true" DataSource="@Orders">
    <GridEvents RowDropped="RowDropHandler" TValue="Order"></GridEvents>
    <GridColumn>
        <GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
        IsPrimaryKey="true" TextAlign="TextAlign.Right" Width="120"></GridColumn>
        <GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
        Width="150"></GridColumn>
        <GridColumn Field=@nameof(Order.OrderDate) HeaderText="Order Date"
        Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
        Width="130"></GridColumn>
        <GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
        TextAlign="TextAlign.Right" Width="120"></GridColumn>
    </GridColumn>
</SfGrid>

@code{
    public List<Order> Orders { get; set; }
    protected override void OnInitialized()
    {
        Orders = Enumerable.Range(1, 75).Select(x => new Order()
        {
            OrderID = 1000 + x,
            CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
            })[new Random().Next(5)],
            Freight = 2.1 * x,
            OrderDate = DateTime.Now.AddDays(-x),
        }).ToList();
    }
}

```

```
}  
public class Order  
{  
    public int? OrderID { get; set; }  
    public string CustomerID { get; set; }  
    public DateTime? OrderDate { get; set; }  
    public double? Freight { get; set; }  
}  
public void RowDropHandler(RowDragEventArgs<Order> args)  
{  
    // Here you can customize your code  
}  
}
```

You can refer to our [Blazor DataGrid](#) feature tour page for its groundbreaking feature representations. You can also explore our [Blazor DataGrid example](#) to understand how to present and manipulate data.

How To

<!-- markdownlint-disable MD024 -->

Blazor DataGrid Component in WebAssembly App using CLI

This article provides a step-by-step introduction to configure Syncfusion Blazor setup, build and run a simple Blazor WebAssembly application using [.NET Core CLI](#).

Starting with version 17.4.0.39 (2019 Volume 4), you need to include a valid license key (either paid or trial key) within your applications. Please refer to this [help topic](#) for more information.

Prerequisites

- [.NET Core SDK 3.1.3](#)

Create a Blazor WebAssembly project using .NET Core CLI

1. Install the Blazor project templates by using below command line in the command prompt:

BASH

```
dotnet new -i Microsoft.AspNetCore.Components.WebAssembly.Templates::3.2.0-rc1.20223.4
```

2. Once project templates installed, run the following command line to create a new Blazor WebAssembly application.

BASH

```
dotnet new blazorwasm -o WebApplication1  
cd WebApplication1
```

Importing Syncfusion Blazor component in the application

1. Now, add **Syncfusion.Blazor** NuGet package to the new application using the below command line.

BASH

```
dotnet add package Syncfusion.Blazor -v ':{:nuget-version:}'  
dotnet restore
```

2. The Syncfusion Blazor package will be included in the newly created project after the installation process is completed.
3. Open `~/_Imports.razor` file and import the `Syncfusion.Blazor`.

ASPX-CS

```
@using Syncfusion.Blazor  
@using Syncfusion.Blazor.Grids
```

4. Open the `~/Program.cs` file and register the Syncfusion Blazor Service.

C#

```
using Syncfusion.Blazor;  
namespace WebApplication1  
{  
    public class Program  
    {  
        public static async Task Main(string[] args)  
        {  
            ....  
            ....  
            builder.Services.AddSyncfusionBlazor();  
            await builder.Build().RunAsync();  
        }  
    }  
}
```

5. Add the Syncfusion bootstrap4 theme in the `element` of the `~/wwwroot/index.html` page.

HTML

```
<head>  
    ....  
    ....  
    <link href="_content/Syncfusion.Blazor/styles/bootstrap4.css"  
        rel="stylesheet" />  
</head>
```

The same theme file can be referred through the CDN version by using <https://cdn.syncfusion.com/blazor/{{ site.blazorversion }}/styles/bootstrap4.css>.

To use manual scripts other than the scripts from NuGet package, register the Blazor service in `~/Program.cs` file by using `true` parameter as mentioned below.

CSHARP

```
using Syncfusion.Blazor;  
namespace WebApplication1  
{  
    public class Program  
    {  
        public static async Task Main(string[] args)  
        {  
            ....  
            ....  
            builder.Services.AddSyncfusionBlazor(true);  
            await builder.Build.RunAsync();  
        }  
    }  
}
```

Add DataGrid Component

To initialize the DataGrid component add the below code to your `Index.razor` view page which is present under `~/Pages` folder. For example, the DataGrid component is added in the `~/Pages/Index.razor` page.

ASPX-CS

```
<SfGrid >  
</SfGrid>
```

Defining Row Data

To bind data for the DataGrid component, you can assign a `IEnumerable` object to the `dataSource` property. The list data source can also be provided as an instance of the `DataManager`. You can assign the data source through the `OnInitialized` life cycle of the page.

ASPX-CS

```
<SfGrid DataSource="@gridData">  
</SfGrid>  
@code{  
    public List<OrdersDetails> gridData { get; set; }  
    protected override void OnInitialized()  
    {  
        gridData = OrdersDetails.GetAllRecords();  
    }  
}
```

Defining Columns

The columns are automatically generated when columns declaration is empty or undefined while initializing the datagrid.

The DataGrid has an option to define columns using [GridColumnns](#) component. In `GridColumnns` component we have properties to customize columns.

Let's check the properties used here:

- We have added [Field](#) to map with a property name an array of JavaScript objects.
- We have added [HeaderText](#) to change the title of columns.
- We have used [TextAlign](#) to change the alignment of columns. By default, columns will be left aligned. To change columns to right align, we need to define `TextAlign` as `Right`.
- Also, we have used another useful property, [Format](#). Using this, we can format number and date values to standard or custom formats.

ASPX-CS

```
<SfGrid DataSource="@gridData">
  <GridColumnns>
    <GridColumn Field=@nameof(OrdersDetails.OrderID) HeaderText="Order ID"
      TextAlign="TextAlign.Right" Width="120"></GridColumn>
    <GridColumn Field=@nameof(OrdersDetails.CustomerID) HeaderText="Customer
      Name" Width="150"></GridColumn>
    <GridColumn Field=@nameof(OrdersDetails.OrderDate) HeaderText=" Order Date"
      Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
      Width="130"></GridColumn>
    <GridColumn Field=@nameof(OrdersDetails.Freight) HeaderText="Freight"
      Format="C2" TextAlign="TextAlign.Right" Width="120"></GridColumn>
    <GridColumn Field=@nameof(OrdersDetails.ShipCountry) HeaderText="Ship
      Country" Width="150"></GridColumn>
  </GridColumnns>
</SfGrid>
@code{
public List<OrdersDetails> gridData { get; set; }
protected override void OnInitialized()
{
  gridData = OrdersDetails.GetAllRecords();
}
```

Enable Paging

The paging feature enables users to view the datagrid record in a paged view. It can be enabled by setting the [AllowPaging](#) property to true. Pager can be customized using the [GridPageSettings](#) component.

ASPX-CS

```
<SfGrid DataSource="@gridData" AllowPaging="true">
  <GridPageSettings PageSize="5"></GridPageSettings>
  <GridColumnns>
    <GridColumn Field=@nameof(OrdersDetails.OrderID) HeaderText="Order ID"
      TextAlign="TextAlign.Right" Width="120"></GridColumn>
    <GridColumn Field=@nameof(OrdersDetails.CustomerID) HeaderText="Customer
      Name" Width="150"></GridColumn>
    <GridColumn Field=@nameof(OrdersDetails.OrderDate) HeaderText=" Order Date"
      Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
      Width="130"></GridColumn>
  </GridColumnns>
</SfGrid>
```

```
<GridColumn Field=@nameof(OrdersDetails.Freight) HeaderText="Freight"
Format="C2" TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(OrdersDetails.ShipCountry) HeaderText="Ship
Country" Width="150"></GridColumn>
</GridColumn>
</SfGrid>
@code{
public List<OrdersDetails> gridData { get; set; }
protected override void OnInitialized()
{
gridData = OrdersDetails.GetAllRecords();
}
}
```

Enable Sorting

The sorting feature enables you to order the records. It can be enabled by setting the [AllowSorting](#) property as true. Sorting feature can be customized using the [GridSortSettings](#) component.

ASPX-CS

```
<SfGrid DataSource="@gridData" AllowPaging="true" AllowSorting="true">
<GridPageSettings PageSize="5"></GridPageSettings>
<GridColumn>
<GridColumn Field=@nameof(OrdersDetails.OrderID) HeaderText="Order ID"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(OrdersDetails.CustomerID) HeaderText="Customer
Name" Width="150"></GridColumn>
<GridColumn Field=@nameof(OrdersDetails.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(OrdersDetails.Freight) HeaderText="Freight"
Format="C2" TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(OrdersDetails.ShipCountry) HeaderText="Ship
Country" Width="150"></GridColumn>
</GridColumn>
</SfGrid>
@code{
public List<OrdersDetails> gridData { get; set; }
protected override void OnInitialized()
{
gridData = OrdersDetails.GetAllRecords();
}
}
```

Enable Filtering

The filtering feature enables you to view reduced amount of records based on filter criteria. It can be enabled by setting the [AllowFiltering](#) property as true. Filtering feature can be customized using the [GridFilterSettings](#) component.

ASPX-CS

```
<SfGrid DataSource="@gridData" AllowPaging="true" AllowSorting="true"
AllowFiltering="true">
<GridPageSettings PageSize="5"></GridPageSettings>
<GridColumn>
```



```

<GridColumn Field=@nameof(OrdersDetails.OrderID) HeaderText="Order ID"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(OrdersDetails.CustomerID) HeaderText="Customer
Name" Width="150"></GridColumn>
<GridColumn Field=@nameof(OrdersDetails.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(OrdersDetails.Freight) HeaderText="Freight"
Format="C2" TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(OrdersDetails.ShipCountry) HeaderText="Ship
Country" Width="150"></GridColumn>
</GridColumns>
</SfGrid>
@code{
public List<OrdersDetails> gridData { get; set; }
protected override void OnInitialized()
{
gridData = OrdersDetails.GetAllRecords();
}
}

```

Enable Grouping

The grouping feature enables you to view the datagrid record in a grouped view. It can be enabled by setting the [AllowGrouping](#) property as true. Grouping feature can be customized using the [GridGroupSettings](#) component.

ASPX-CS

```

<SfGrid DataSource="@gridData" AllowPaging="true" AllowSorting="true"
AllowFiltering="true" AllowGrouping="true">
<GridPageSettings PageSize="5"></GridPageSettings>
<GridColumns>
<GridColumn Field=@nameof(OrdersDetails.OrderID) HeaderText="Order ID"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(OrdersDetails.CustomerID) HeaderText="Customer
Name" Width="150"></GridColumn>
<GridColumn Field=@nameof(OrdersDetails.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(OrdersDetails.Freight) HeaderText="Freight"
Format="C2" TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(OrdersDetails.ShipCountry) HeaderText="Ship
Country" Width="150"></GridColumn>
</GridColumns>
</SfGrid>
@code{
public List<OrdersDetails> gridData { get; set; }
protected override void OnInitialized()
{
gridData = OrdersDetails.GetAllRecords();
}
}

```

Output be like the below.

Drag a column header here to group its column				
Order ID	Customer Name	Order Date	Freight	Ship Country
10001	ALFKI	5/15/1991	\$2.30	Denmark
10002	ANATR	4/4/1990	\$3.30	Brazil
10003	ANTON	11/30/1957	\$4.30	Germany
10004	BLONP	10/22/1930	\$5.30	Austria
10005	BOLID	2/18/1953	\$6.30	Switzerland

K < 1 2 3 4 5 6 7 8 ... > X
 1 of 14 pages (70 items)

See Also

- [Getting Started with Syncfusion DataGrid in Blazor WebAssembly using Visual Studio 2019](#)
- [Getting Started with Syncfusion DataGrid in Blazor Server-Side using Visual Studio 2019](#)
- [Getting Started with Syncfusion DataGrid in Blazor Server-Side using .NET Core CLI](#)

<!-- markdownlint-disable MD024 -->

Blazor DataGrid Component in Server Side App using CLI

This article provides a step-by-step instructions to configure Syncfusion Blazor Data Grid in Blazor Server side application using [.NET Core CLI](#).

Starting with version 17.4.0.39 (2019 Volume 4), you need to include a valid license key (either paid or trial key) within your applications. Please refer to this [help topic](#) for more information.

Prerequisites

- [.NET Core SDK 3.1.3](#)

Create a Blazor Server side project using .NET Core CLI

1. Run the following command line to create a new Blazor Server application.

BASH

```
dotnet new blazorserver -o WebApplication1
cd WebApplication1
```

Importing Syncfusion Blazor component in the application

1. Now, add **Syncfusion.Blazor** NuGet package to the new application using the below command line.

BASH

```
dotnet add package Syncfusion.Blazor -v ':{nuget-version:}'
```

```
dotnet restore
```

-
2. The Syncfusion Blazor package will be included in the newly created project after the installation process is completed.
3. Open `~/_Imports.razor` file and import the `Syncfusion.Blazor`.

ASPX-CS

```
@using Syncfusion.Blazor  
@using Syncfusion.Blazor.Grids
```

-
-
-
4. Open the `~/Startup.cs` file and register the Syncfusion Blazor Service.

C#

```
using Syncfusion.Blazor;  
namespace WebApplication1  
{  
    public class Startup  
    {  
        public void ConfigureServices(IServiceCollection services)  
        {  
            ....  
            ....  
            services.AddSyncfusionBlazor();  
        }  
    }  
}
```

-
-
-
-
5. Add the Syncfusion bootstrap4 theme in the `element` of the `~/Pages/_Host.cshtml` page.

HTML

```
<head>  
    ....  
    ....  
    <link href="_content/Syncfusion.Blazor/styles/bootstrap4.css"  
    rel="stylesheet" />  
</head>
```

The same theme file can be referred through the CDN version by using <https://cdn.syncfusion.com/blazor/{site.blazorversion}/styles/bootstrap4.css>.

To use manual scripts other than the scripts from NuGet package, register the Blazor service in `~/Startup.cs` file by using `true` parameter as mentioned below.

CSHARP

```
using Syncfusion.Blazor;  
namespace WebApplication1  
{
```

```
public class Startup
{
    public void ConfigureServices(IServiceCollection services)
    {
        ....
        ....
        services.AddSyncfusionBlazor(true);
    }
}
```

Add DataGrid Component

To initialize the DataGrid component add the below code to your **Index.razor** view page which is present under **~/Pages** folder. For example, the DataGrid component is added in the **~/Pages/Index.razor** page.

CSHARP

```
<SfGrid>
</SfGrid>
```

Defining Row Data

To bind data for the DataGrid component, you can assign a [IEnumerable](#) object to the [DataSource](#) property. The list data source can also be provided as an instance of the [DataManager](#). You can assign the data source through the **OnInitialized** life cycle of the page.

CSHARP

```
<SfGrid DataSource="@gridData">
</SfGrid>
@code{
    public List<OrdersDetails> gridData { get; set; }
    protected override void OnInitialized()
    {
        gridData = OrdersDetails.GetAllRecords();
    }
}
```

Defining Columns

The columns are automatically generated when columns declaration is empty or undefined while initializing the datagrid.

The DataGrid has an option to define columns using [GridColumnns](#) component. In [GridColumn](#) component we have properties to customize columns.

Let's check the properties used here:

- We have added [Field](#) to map with a property name an array of JavaScript objects.
- We have added [HeaderText](#) to change the title of columns.
- We have used [TextAlign](#) to change the alignment of columns. By default, columns will be left aligned. To change columns to right align, we need to define [TextAlign](#) as [Right](#).
- Also, we have used another useful property, [Format](#). Using this, we can format number and date values to standard or custom formats.

CSHARP

```
<SfGrid DataSource="@gridData">
<GridColumns>
<GridColumn Field=@nameof(OrdersDetails.OrderID) HeaderText="Order ID"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(OrdersDetails.CustomerID) HeaderText="Customer
Name" Width="150"></GridColumn>
<GridColumn Field=@nameof(OrdersDetails.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(OrdersDetails.Freight) HeaderText="Freight"
Format="C2" TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(OrdersDetails.ShipCountry) HeaderText="Ship
Country" Width="150"></GridColumn>
</GridColumns>
</SfGrid>
@code{
public List<OrdersDetails> gridData { get; set; }
protected override void OnInitialized()
{
gridData = OrdersDetails.GetAllRecords();
}
}
```

Enable Paging

The paging feature enables users to view the datagrid record in a paged view. It can be enabled by setting the [AllowPaging](#) property to true. Pager can be customized using the [GridPageSettings](#) component.

CSHARP

```
<SfGrid DataSource="@gridData" AllowPaging="true">
<GridPageSettings PageSize="5"></GridPageSettings>
<GridColumns>
<GridColumn Field=@nameof(OrdersDetails.OrderID) HeaderText="Order ID"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(OrdersDetails.CustomerID) HeaderText="Customer
Name" Width="150"></GridColumn>
<GridColumn Field=@nameof(OrdersDetails.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(OrdersDetails.Freight) HeaderText="Freight"
Format="C2" TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(OrdersDetails.ShipCountry) HeaderText="Ship
Country" Width="150"></GridColumn>
</GridColumns>
</SfGrid>
@code{
public List<OrdersDetails> gridData { get; set; }
protected override void OnInitialized()
{
gridData = OrdersDetails.GetAllRecords();
}
}
```

Enable Sorting

The sorting feature enables you to order the records. It can be enabled by setting the [AllowSorting](#) property as true. Sorting feature can be customized using the [GridSortSettings](#) component.

CSHARP

```
<SfGrid DataSource="@gridData" AllowPaging="true" AllowSorting="true">
  <GridPageSettings PageSize="5"></GridPageSettings>
  <GridColumns>
    <GridColumn Field=@nameof(OrdersDetails.OrderID) HeaderText="Order ID"
      TextAlign="TextAlign.Right" Width="120"></GridColumn>
    <GridColumn Field=@nameof(OrdersDetails.CustomerID) HeaderText="Customer
      Name" Width="150"></GridColumn>
    <GridColumn Field=@nameof(OrdersDetails.OrderDate) HeaderText=" Order Date"
      Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
      Width="130"></GridColumn>
    <GridColumn Field=@nameof(OrdersDetails.Freight) HeaderText="Freight"
      Format="C2" TextAlign="TextAlign.Right" Width="120"></GridColumn>
    <GridColumn Field=@nameof(OrdersDetails.ShipCountry) HeaderText="Ship
      Country" Width="150"></GridColumn>
  </GridColumns>
</SfGrid>
@code{
public List<OrdersDetails> gridData { get; set; }
protected override void OnInitialized()
{
  gridData = OrdersDetails.GetAllRecords();
}
}
```

Enable Filtering

The filtering feature enables you to view reduced amount of records based on filter criteria. It can be enabled by setting the [AllowFiltering](#) property as true. Filtering feature can be customized using the [GridFilterSettings](#) component.

CSHARP

```
<SfGrid DataSource="@gridData" AllowPaging="true" AllowSorting="true"
  AllowFiltering="true">
  <GridPageSettings PageSize="5"></GridPageSettings>
  <GridColumns>
    <GridColumn Field=@nameof(OrdersDetails.OrderID) HeaderText="Order ID"
      TextAlign="TextAlign.Right" Width="120"></GridColumn>
    <GridColumn Field=@nameof(OrdersDetails.CustomerID) HeaderText="Customer
      Name" Width="150"></GridColumn>
    <GridColumn Field=@nameof(OrdersDetails.OrderDate) HeaderText=" Order Date"
      Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
      Width="130"></GridColumn>
    <GridColumn Field=@nameof(OrdersDetails.Freight) HeaderText="Freight"
      Format="C2" TextAlign="TextAlign.Right" Width="120"></GridColumn>
    <GridColumn Field=@nameof(OrdersDetails.ShipCountry) HeaderText="Ship
      Country" Width="150"></GridColumn>
  </GridColumns>
</SfGrid>
@code{
public List<OrdersDetails> gridData { get; set; }
}
```

```
protected override void OnInitialized()  
{  
    gridData = OrdersDetails.GetAllRecords();  
}  
}
```

Enable Grouping

The grouping feature enables you to view the datagrid record in a grouped view. It can be enabled by setting the [AllowGrouping](#) property as true. Grouping feature can be customized using the [GridGroupSettings](#) component.

CSHARP

```
<SfGrid DataSource="@gridData" AllowPaging="true" AllowSorting="true"  
AllowFiltering="true" AllowGrouping="true">  
    <GridPageSettings PageSize="5"></GridPageSettings>  
    <GridColumns>  
        <GridColumn Field=@nameof(OrdersDetails.OrderID) HeaderText="Order ID"  
            TextAlign="TextAlign.Right" Width="120"></GridColumn>  
        <GridColumn Field=@nameof(OrdersDetails.CustomerID) HeaderText="Customer  
Name" Width="150"></GridColumn>  
        <GridColumn Field=@nameof(OrdersDetails.OrderDate) HeaderText=" Order Date"  
            Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"  
            Width="130"></GridColumn>  
        <GridColumn Field=@nameof(OrdersDetails.Freight) HeaderText="Freight"  
            Format="C2" TextAlign="TextAlign.Right" Width="120"></GridColumn>  
        <GridColumn Field=@nameof(OrdersDetails.ShipCountry) HeaderText="Ship  
Country" Width="150"></GridColumn>  
    </GridColumns>  
</SfGrid>  
@code{  
    public List<OrdersDetails> gridData { get; set; }  
    protected override void OnInitialized()  
    {  
        gridData = OrdersDetails.GetAllRecords();  
    }  
}
```

Output be like the below.

Drag a column header here to group its column				
Order ID	Customer Name	Order Date	Freight	Ship Country
10001	ALFKI	5/15/1991	\$2.30	Denmark
10002	ANATR	4/4/1990	\$3.30	Brazil
10003	ANTON	11/30/1957	\$4.30	Germany
10004	BLONP	10/22/1930	\$5.30	Austria
10005	BOLID	2/18/1953	\$6.30	Switzerland

K < 1 2 3 4 5 6 7 8 ... > X
 1 of 14 pages (70 items)

See Also

- [Getting Started with Syncfusion DataGrid in Blazor Server-Side using Visual Studio 2019](#)
- [Getting started with Syncfusion Data Grid in Blazor WebAssembly App using .NET Core CLI](#)
- [Getting Started with Syncfusion DataGrid in Blazor WebAssembly using Visual Studio 2019](#)

<!-- markdownlint-disable MD024 -->

Blazor DataGrid Component in WebAssembly App using Visual Studio

This article provides a step-by-step instructions to configure Syncfusion Blazor DataGrid in a simple Blazor WebAssembly application using [Visual Studio 2019](#).

Starting with version 17.4.0.39 (2019 Volume 4), you need to include a valid license key (either paid or trial key) within your applications. Please refer to this [help topic](#) for more information.

Prerequisites

- [Visual Studio 2019](#)
- [.NET Core SDK 3.1.3](#)

.NET Core SDK 3.1.3 requires Visual Studio 2019 16.6 or later.

Syncfusion Blazor components are compatible with .NET Core 5.0 Preview 6 and it requires Visual Studio 16.7 Preview 1 or later.

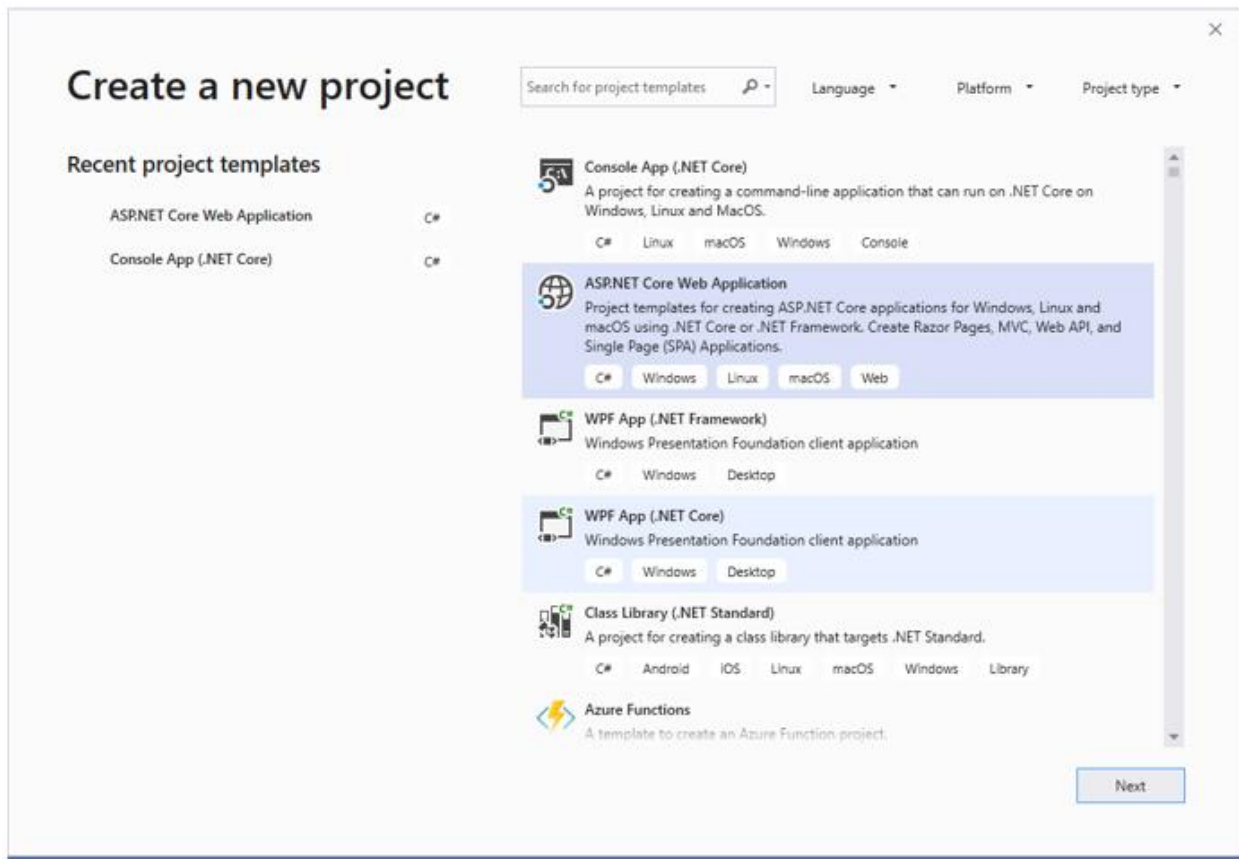
Create a Blazor WebAssembly project in Visual Studio 2019

1. Install the essential project templates in the Visual Studio 2019 by running the below command line in the command prompt.

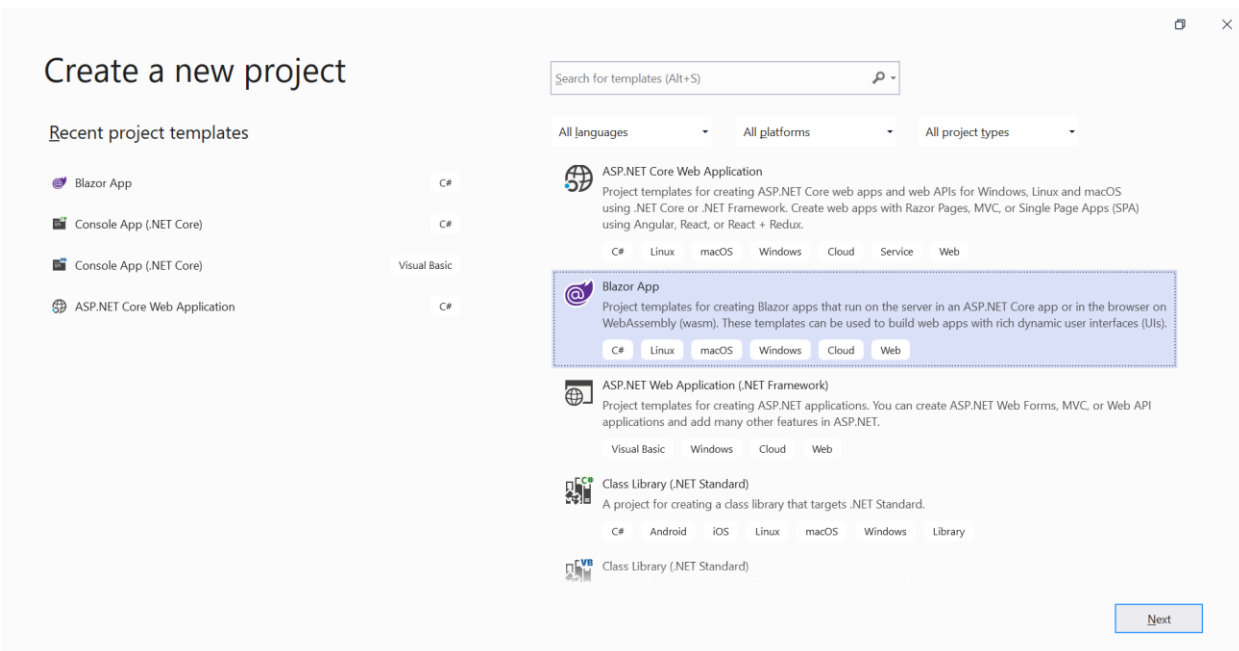
BASH

```
dotnet new -i Microsoft.AspNetCore.Components.WebAssembly.Templates::3.2.0-rc1.20223.4
```


2. Choose **Create a new project** from the Visual Studio dashboard.



3. Select **Blazor App** from the template and click **Next** button.



- Now, the project configuration window will popup. Click **Create** button to create a new project with the default project configuration.

Configure your new project

Blazor App C# Linux macOS Windows Cloud Web

Project name

BlazorApp

Location

C:\Users\DhivyaRajendran\source\repos

Solution

Create new solution

Solution name ⓘ

BlazorApp


☐ Place solution and project in the same directory

Back Create


- Choose **Blazor WebAssembly App** from the dashboard and click **Create** button to create a new Blazor WebAssembly application. Make sure **.NET Core** and **ASP.NET Core 3.1** is selected at the top.

Create a new Blazor app

.NET Core 3.1

 **Blazor Server App**

A project template for creating a Blazor server app that runs server-side inside an ASP.NET Core app and handles user interactions over a SignalR connection. This template can be used for web apps with rich dynamic user interfaces (UIs).

 **Blazor WebAssembly App**

A project template for creating a Blazor app that runs on WebAssembly. This template can be used for web apps with rich dynamic user interfaces (UIs).

Authentication

No Authentication

[Change](#)

Advanced

☒ Configure for HTTPS

☐ Enable Docker Support

(Requires Docker Desktop)

Linux

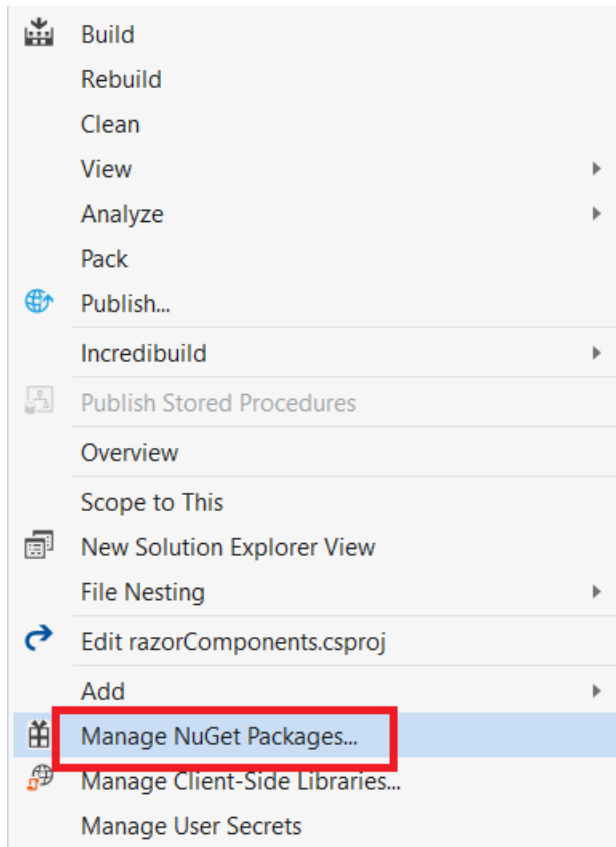
☐ ASP.NET Core hosted

☐ Progressive Web Application

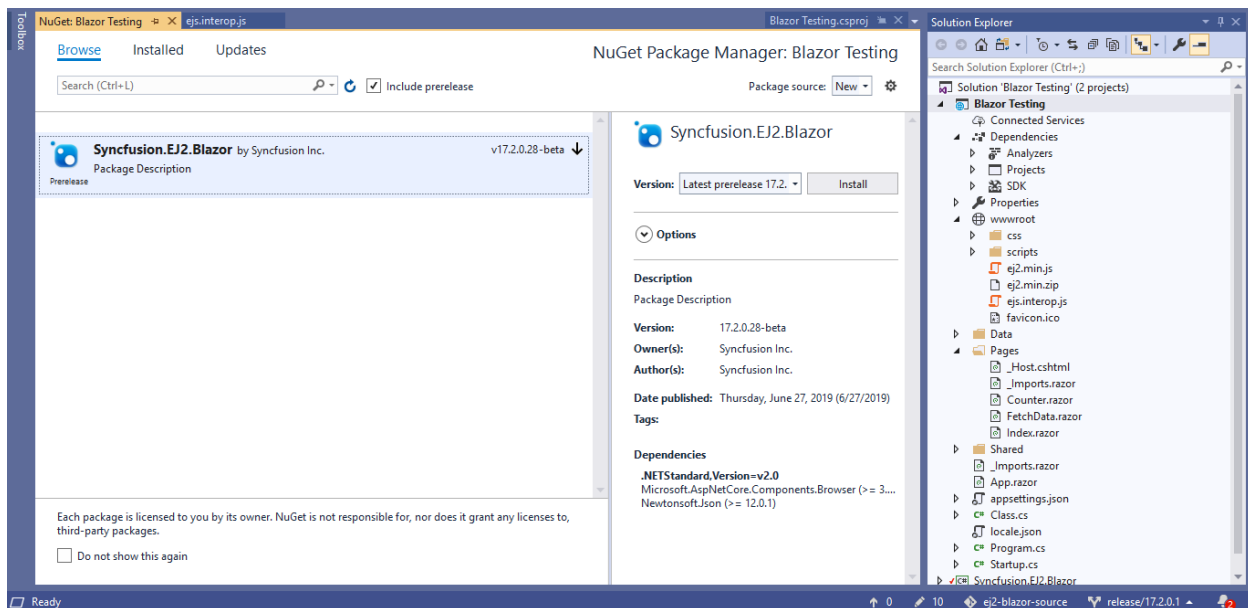
ASP.NET Core 3.1 available in Visual Studio 2019 version.

Importing Syncfusion Blazor component in the application

1. Now, install **Syncfusion.Blazor** NuGet package to the newly created application by using the **NuGet Package Manager**. Right-click the project and select Manage NuGet Packages.



2. Search **Syncfusion.Blazor** keyword in the Browser tab and install **Syncfusion.Blazor** NuGet package in the application.



3. The Syncfusion Blazor package will be installed in the project, once the installation process is completed.

4. Open `~/_Imports.razor` file and import the `Syncfusion.Blazor`.

ASPX-CS

```
@using Syncfusion.Blazor
@using Syncfusion.Blazor.Grids.
```

5. Open the `~/Program.cs` file and register the Syncfusion Blazor Service.

C#

```
using Syncfusion.Blazor;
namespace WebApplication1
{
    public class Program
    {
        public static async Task Main(string[] args)
        {
            ....
            ....
            builder.Services.AddSyncfusionBlazor();
            await builder.Build().RunAsync();
        }
    }
}
```

6. Add the Syncfusion bootstrap4 theme in the `element` of the `~/wwwroot/index.html` page.

HTML

```
<head>
....
....
<link href="_content/Syncfusion.Blazor/styles/bootstrap4.css"
rel="stylesheet" />
</head>
```

The same theme file can be referred through the CDN version by using <https://cdn.syncfusion.com/blazor/{site.blazorversion}/styles/bootstrap4.css>.

To use manual scripts other than the scripts from NuGet package, register the Blazor service in `~/Program.cs` file by using `true` parameter as mentioned below.

CSHARP

```
using Syncfusion.Blazor;
namespace WebApplication1
{
    public class Program
    {
        public static async Task Main(string[] args)
        {
            ....
        }
    }
}
```

```
.....  
builder.Services.AddSyncfusionBlazor(true);  
await builder.Build.RunAsync();  
}  
}  
}
```

Add DataGrid Component

To initialize the DataGrid component add the below code to your **Index.razor** view page which is present under **~/Pages** folder. For example, the DataGrid component is added in the **~/Pages/Index.razor** page.

ASPX-CS

```
<SfGrid >  
</SfGrid>
```

Defining Row Data

To bind data for the DataGrid component, you can assign a [IEnumerable](#) object to the [dataSource](#) property. The list data source can also be provided as an instance of the [DataManager](#). You can assign the data source through the **OnInitialized** life cycle of the page.

ASPX-CS

```
<SfGrid DataSource="@gridData">  
</SfGrid>  
@code{  
public List<OrdersDetails> gridData { get; set; }  
protected override void OnInitialized()  
{  
    gridData = OrdersDetails.GetAllRecords();  
}  
}
```

Defining Columns

The columns are automatically generated when columns declaration is empty or undefined while initializing the datagrid.

The DataGrid has an option to define columns using [GridColumnns](#) component. In [GridColumn](#) component we have properties to customize columns.

Let's check the properties used here:

- We have added [Field](#) to map with a property name an array of JavaScript objects.
- We have added [HeaderText](#) to change the title of columns.
- We have used [TextAlign](#) to change the alignment of columns. By default, columns will be left aligned. To change columns to right align, we need to define [TextAlign](#) as **Right**.
- Also, we have used another useful property, [Format](#). Using this, we can format number and date values to standard or custom formats.

ASPX-CS

```
<SfGrid DataSource="@gridData">  
<GridColumnns>
```

```

<GridColumn Field=@nameof(OrdersDetails.OrderID) HeaderText="Order ID"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(OrdersDetails.CustomerID) HeaderText="Customer
Name" Width="150"></GridColumn>
<GridColumn Field=@nameof(OrdersDetails.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(OrdersDetails.Freight) HeaderText="Freight"
Format="C2" TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(OrdersDetails.ShipCountry) HeaderText="Ship
Country" Width="150"></GridColumn>
</GridColumn>
</SfGrid>
@code{
public List<OrdersDetails> gridData { get; set; }
protected override void OnInitialized()
{
gridData = OrdersDetails.GetAllRecords();
}
}

```

Enable Paging

The paging feature enables users to view the datagrid record in a paged view. It can be enabled by setting the [AllowPaging](#) property to true. Pager can be customized using the [GridPageSettings](#) component.

ASPX-CS

```

<SfGrid DataSource="@gridData" AllowPaging="true">
<GridPageSettings PageSize="5"></GridPageSettings>
<GridColumn>
<GridColumn Field=@nameof(OrdersDetails.OrderID) HeaderText="Order ID"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(OrdersDetails.CustomerID) HeaderText="Customer
Name" Width="150"></GridColumn>
<GridColumn Field=@nameof(OrdersDetails.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(OrdersDetails.Freight) HeaderText="Freight"
Format="C2" TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(OrdersDetails.ShipCountry) HeaderText="Ship
Country" Width="150"></GridColumn>
</GridColumn>
</SfGrid>
@code{
public List<OrdersDetails> gridData { get; set; }
protected override void OnInitialized()
{
gridData = OrdersDetails.GetAllRecords();
}
}

```

Enable Sorting

The sorting feature enables you to order the records. It can be enabled by setting the [AllowSorting](#) property as true. Sorting feature can be customized using the [GridSortSettings](#) component.

ASPX-CS

```

<SfGrid DataSource="@gridData" AllowPaging="true" AllowSorting="true">
  <GridPageSettings PageSize="5"></GridPageSettings>
  <GridColumns>
    <GridColumn Field=@nameof(OrdersDetails.OrderID) HeaderText="Order ID"
      TextAlign="TextAlign.Right" Width="120"></GridColumn>
    <GridColumn Field=@nameof(OrdersDetails.CustomerID) HeaderText="Customer
      Name" Width="150"></GridColumn>
    <GridColumn Field=@nameof(OrdersDetails.OrderDate) HeaderText=" Order Date"
      Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
      Width="130"></GridColumn>
    <GridColumn Field=@nameof(OrdersDetails.Freight) HeaderText="Freight"
      Format="C2" TextAlign="TextAlign.Right" Width="120"></GridColumn>
    <GridColumn Field=@nameof(OrdersDetails.ShipCountry) HeaderText="Ship
      Country" Width="150"></GridColumn>
  </GridColumns>
</SfGrid>
@code{
public List<OrdersDetails> gridData { get; set; }
protected override void OnInitialized()
{
  gridData = OrdersDetails.GetAllRecords();
}
}

```

Enable Filtering

The filtering feature enables you to view reduced amount of records based on filter criteria. It can be enabled by setting the [AllowFiltering](#) property as true. Filtering feature can be customized using the [GridFilterSettings](#) component.

ASPX-CS

```

<SfGrid DataSource="@gridData" AllowPaging="true" AllowSorting="true"
  AllowFiltering="true">
  <GridPageSettings PageSize="5"></GridPageSettings>
  <GridColumns>
    <GridColumn Field=@nameof(OrdersDetails.OrderID) HeaderText="Order ID"
      TextAlign="TextAlign.Right" Width="120"></GridColumn>
    <GridColumn Field=@nameof(OrdersDetails.CustomerID) HeaderText="Customer
      Name" Width="150"></GridColumn>
    <GridColumn Field=@nameof(OrdersDetails.OrderDate) HeaderText=" Order Date"
      Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
      Width="130"></GridColumn>
    <GridColumn Field=@nameof(OrdersDetails.Freight) HeaderText="Freight"
      Format="C2" TextAlign="TextAlign.Right" Width="120"></GridColumn>
    <GridColumn Field=@nameof(OrdersDetails.ShipCountry) HeaderText="Ship
      Country" Width="150"></GridColumn>
  </GridColumns>
</SfGrid>
@code{
public List<OrdersDetails> gridData { get; set; }
protected override void OnInitialized()
{
  gridData = OrdersDetails.GetAllRecords();
}
}

```



```
}

```

Enable Grouping

The grouping feature enables you to view the datagrid record in a grouped view. It can be enabled by setting the [AllowGrouping](#) property as true. Grouping feature can be customized using the [GridGroupSettings](#) component.

ASPX-CS

```
<SfGrid DataSource="@gridData" AllowPaging="true" AllowSorting="true"
AllowFiltering="true" AllowGrouping="true">
  <GridPageSettings PageSize="5"></GridPageSettings>
  <GridColumns>
    <GridColumn Field=@nameof(OrdersDetails.OrderID) HeaderText="Order ID"
    TextAlign="TextAlign.Right" Width="120"></GridColumn>
    <GridColumn Field=@nameof(OrdersDetails.CustomerID) HeaderText="Customer
    Name" Width="150"></GridColumn>
    <GridColumn Field=@nameof(OrdersDetails.OrderDate) HeaderText=" Order Date"
    Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
    Width="130"></GridColumn>
    <GridColumn Field=@nameof(OrdersDetails.Freight) HeaderText="Freight"
    Format="C2" TextAlign="TextAlign.Right" Width="120"></GridColumn>
    <GridColumn Field=@nameof(OrdersDetails.ShipCountry) HeaderText="Ship
    Country" Width="150"></GridColumn>
  </GridColumns>
</SfGrid>
@code{
public List<OrdersDetails> gridData { get; set; }
protected override void OnInitialized()
{
gridData = OrdersDetails.GetAllRecords();
}
}
```

Output be like the below.

Drag a column header here to group its column				
Order ID	Customer Name	Order Date	Freight	Ship Country
10001	ALFKI	5/15/1991	\$2.30	Denmark
10002	ANATR	4/4/1990	\$3.30	Brazil
10003	ANTON	11/30/1957	\$4.30	Germany
10004	BLONP	10/22/1930	\$5.30	Austria
10005	BOLID	2/18/1953	\$6.30	Switzerland

K
<
1
2
3
4
5
6
7
8
...
>
X

1 of 14 pages (70 items)

See Also

- [Getting Started with Syncfusion Data Grid in Blazor WebAssembly using .NET Core CLI](#)
- [Getting Started with Syncfusion DataGrid in Blazor Server-Side using Visual Studio 2019](#)
- [Getting Started with Syncfusion Data Grid in Blazor Server-Side using .NET Core CLI](#)

Show or Hide columns in Dialog editing in Blazor DataGrid Component

You can show hidden columns or hide visible column's editor in the dialog while editing the datagrid record. This can be achieved by **Template**.

In the below example, we have rendered the datagrid columns [OrderDate] as hidden column and [Freight] as visible column. In the edit mode, we have changed the [Freight] column to visible state and [OrderDate] column to hidden state.

ASPX-CS

```
@using Syncfusion.Blazor.Grids
@using Syncfusion.Blazor.DropDowns
@using Syncfusion.Blazor.Inputs
@using Syncfusion.Blazor.Calendars
<SfGrid DataSource="@Orders" AllowPaging="true" @ref="Grid" Toolbar="@ (new
string[] { "Add", "Edit", "Delete", "Cancel", "Update" })" Height="315">
<GridEvents OnActionBegin="ActionBeginHandler" TValue="Order"></GridEvents>
<GridEditSettings AllowAdding="true" AllowEditing="true"
AllowDeleting="true" Mode="EditMode.Dialog">
<Template>
@{
var Order = (context as Order);
<div>
<div class="form-row">
<div class="form-group col-md-6">
<label>Order ID</label>
<SfNumericTextBox TValue="int?" FloatLabelType="FloatLabelType.Always"
@bind-Value="@ (Order.OrderID)" Enabled="@Data"></SfNumericTextBox>
</div>
<div class="form-group col-md-6">
<label>Customer Name</label>
<SfAutoComplete ID="customerID" FloatLabelType="FloatLabelType.Always"
TItem="Order" @bind-Value="@ (Order.CustomerID)" TValue="string"
DataSource="@GridData">
<AutoCompleteFieldSettings Value="CustomerID"></AutoCompleteFieldSettings>
</SfAutoComplete>
</div>
<div class="form-group col-md-6">
<label>Order Date</label>
<SfDatePicker ID="OrderDate" FloatLabelType="FloatLabelType.Always" @bind-
Value="@ (Order.OrderDate)"></SfDatePicker>
</div>
</div>
</div>
</div>
</Template>
</GridEditSettings>
<GridColumns>
```

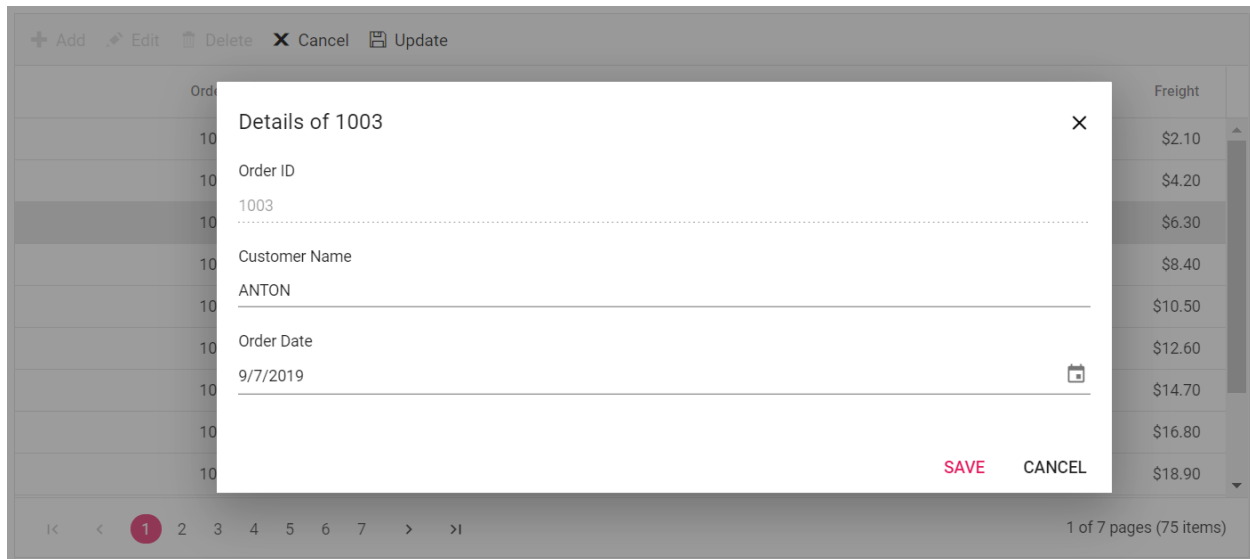
```

<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
IsPrimaryKey="true" ValidationRules="@ (new ValidationRules{ Required=true})"
TextAlign="TextAlign.Center" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
TextAlign="TextAlign.Center" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
EditType="EditType.DatePickerEdit" Visible="false" Format="d"
TextAlign="TextAlign.Center" Width="130"
Type="ColumnType.Date"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumns>
</SfGrid>
@code{
SfGrid<Order> Grid;
public List<Order> Orders { get; set; }
public bool Enabled = true;
public bool Data = false;
public List<Order> GridData = new List<Order>
{
    new Order() { OrderID = 10248, CustomerID = "VINET", Freight = 32.38,
    OrderDate = DateTime.Now.AddDays(-2) },
    new Order() { OrderID = 10249, CustomerID = "TOMSP", Freight = 11.61,
    OrderDate = DateTime.Now.AddDays(-5) },
    new Order() { OrderID = 10250, CustomerID = "HANAR", Freight = 65.83,
    OrderDate = DateTime.Now.AddDays(-12) },
    new Order() { OrderID = 10251, CustomerID = "VICTE", Freight = 41.34,
    OrderDate = DateTime.Now.AddDays(-18) },
    new Order() { OrderID = 10252, CustomerID = "SUPRD", Freight = 51.3,
    OrderDate = DateTime.Now.AddDays(-22) },
    new Order() { OrderID = 10253, CustomerID = "HANAR", Freight = 58.17,
    OrderDate = DateTime.Now.AddDays(-26) },
};
public void ActionBeginHandler(ActionEventArgs<Order> args)
{
    if (args.RequestType == Syncfusion.Blazor.Grids.Action.Add)
    {
        Data = true;
    }
}
protected override void OnInitialized()
{
    Orders = Enumerable.Range(1, 75).Select(x => new Order()
    {
        OrderID = 1000 + x,
        CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
    })[new Random().Next(5)],
        Freight = 2.1 * x,
        OrderDate = DateTime.Now.AddDays(-x),
    }).ToList();
}
public class Order
{
    public int? OrderID { get; set; }
    public string CustomerID { get; set; }
    public DateTime? OrderDate { get; set; }
    public double? Freight { get; set; }
}

```

```
}
}
```

Output be like the below.



Create custom toolbar with drop-down list in Blazor DataGrid Component

You can create your own Toolbar items in the DataGrid. It can be added by defining the [Toolbar](#). Actions for this Toolbar template items are defined in the [ToolbarClick`]

Step 1:

Initialize the template for your custom component. Using the following code add the DropDownList component to the Toolbar.

ASPX-CS

```
<SfToolbar>
<ToolbarItems>
<ToolbarItem Type="ItemType.Input">
<Template>
<SfDropDownList TValue="string" TItem="Select" Placeholder="Enter the value"
DataSource=@LocalData Width="200">
<DropDownListFieldSettings Text="text" Value="text">
</DropDownListFieldSettings>
<DropDownListEvents TValue="string" TItem="Select" ValueChange="OnChange">
</DropDownListEvents>
</SfDropDownList>
</Template>
</ToolbarItem>
</ToolbarItems>
</SfToolbar>
```

Step 2:

To render the DropDownList component, use the [DropDownListEvents](#)

You can select the datagrid row index based on the selected data in the DropDownList. The output will appear as follows.

ASPX-CS

```
@using Syncfusion.Blazor
@using Syncfusion.Blazor.Grids
@using Syncfusion.Blazor.Navigations
@using Syncfusion.Blazor.DropDowns
<SfGrid DataSource="@Orders" AllowPaging="true" Height="200" @ref="Grid">
<GridEvents TValue="Order"></GridEvents>
<SfToolbar>
<ToolbarItems>
<ToolbarItem Type="ItemType.Input">
<Template>
<SfDropDownList TValue="string" TItem="Select" Placeholder="Enter the value"
DataSource=@LocalData Width="200">
<DropDownListFieldSettings Text="text" Value="text">
</DropDownListFieldSettings>
<DropDownListEvents TValue="string" TItem="Select" ValueChange="OnChange">
</DropDownListEvents>
</SfDropDownList>
</Template>
</ToolbarItem>
</ToolbarItems>
</SfToolbar>
<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText="Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumns>
</SfGrid>
@code{
SfGrid<Order> Grid;
public List<Order> Orders { get; set; }
public class Select
{
public string text { get; set; }
}
List<Select> LocalData = new List<Select>
{
new Select() { text = "0"},
new Select() { text = "1"},
new Select() { text = "2"},
new Select() { text = "3"},
new Select() { text = "4"},
new Select() { text = "5"},
new Select() { text = "6"},
new Select() { text = "7"},
new Select() { text = "8"},
new Select() { text = "9"},
}
```

```

};
protected override void OnInitialized()
{
    Orders = Enumerable.Range(1, 10).Select(x => new Order()
    {
        OrderID = 1000 + x,
        CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
    })[new Random().Next(5)],
        Freight = 2.1 * x,
        OrderDate = DateTime.Now.AddDays(-x),
    }).ToList();
}
public async Task
OnChange(Syncfusion.Blazor.DropDowns.ChangeEventArgs<string, Select> args)
{
    await this.Grid.SelectRow(int.Parse(args.Value));
}
public class Order
{
    public int? OrderID { get; set; }
    public string CustomerID { get; set; }
    public DateTime? OrderDate { get; set; }
    public double? Freight { get; set; }
}
}

```

Output be like the below.

Enter the value 

Order ID	Customer Name	Order Date	Freight
1001	ANTON	9/5/2019	\$2.10
1002	ANANTR	9/4/2019	\$4.20
1003	ANTON	9/3/2019	\$6.30
1004	ANTON	9/2/2019	\$8.40
1005	ALFKI	9/1/2019	\$10.50
1006	BLONP	8/31/2019	\$12.60

1

1 of 1 pages (10 items)

Customize Column Styles in Blazor DataGrid Component

You can customize the appearance of the header and content of a particular column using the [CustomAttributes](#) property.

To customize the datagrid column, follow the given steps:

Step 1:

Create a CSS class with custom style to override the default style for row cell and header cell.

CSS

```

.e-attr{
    background: #5DADE2;
}

```

```
font-family: "Bell MT";
color: red;
font-size: 5px;
}
```

Step 2:

Add the custom CSS class to the specified column by using the [CustomAttributes](#) property.

ASPX-CS

```
@using Syncfusion.Blazor
@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders" AllowPaging="true" Height="200" @ref="Grid">
<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150" ></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" CustomAttributes="@ (new Dictionary<string,
object>()) { { "class", "e-attr" } })" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumns>
</SfGrid>
<style>
.e-attr{
background: #5DADE2;
font-family: "Bell MT";
color: red;
font-size: 5px;
}
</style>
@code{
SfGrid<Order> Grid;
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 10).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
OrderDate = DateTime.Now.AddDays(-x),
}).ToList();
}
public class Order
{
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
}
```

Output be like the below.

Order ID	Customer Name	Order Date	Freight
1001	ALFKI	9/3/2019	\$2.10
1002	ANANTR	9/2/2019	\$4.20
1003	ANTON	9/1/2019	\$6.30
1004	ANTON	8/31/2019	\$8.40
1005	BOLID	8/30/2019	\$10.50
1006	BOLID	8/29/2019	\$12.60

1 of 1 pages (10 items)

Access public methods in Blazor DataGrid Component

You can access the public methods available in the DataGrid component by using its reference defined in the component initialization.

This is demonstrated in the below sample code where the [Print](#) method of the datagrid component is invoked on button click using the datagrid reference,

ASPX-CS

```
@using Syncfusion.Blazor.Buttons
@using Syncfusion.Blazor.Grids
<SfButton OnClick="Print" CssClass="e-primary" IsPrimary="true"
Content="Print data"></SfButton>
<SfGrid @ref="DefaultGrid" DataSource="@Employees">
<GridColumns>
<GridColumn Field=@nameof(EmployeeData.EmployeeID)
TextAlign="TextAlign.Center" HeaderText="Employee ID"
Width="120"></GridColumn>
<GridColumn Field=@nameof(EmployeeData.FirstName) HeaderText="First Name"
ShowInColumnChooser="false" Width="130"></GridColumn>
<GridColumn Field=@nameof(EmployeeData.LastName) HeaderText="Last Name"
Width="130"></GridColumn>
<GridColumn Field=@nameof(EmployeeData.Title) HeaderText="Title"
Width="120"></GridColumn>
<GridColumn Field=@nameof(EmployeeData.HireDate) HeaderText="Hire Date"
Format="d" TextAlign="TextAlign.Right" Width="150"></GridColumn>
</GridColumns>
</SfGrid>
@code{
private SfGrid<EmployeeData> DefaultGrid;
public List<EmployeeData> Employees { get; set; }
protected override void OnInitialized()
{
Employees = Enumerable.Range(1, 9).Select(x => new EmployeeData()
{
EmployeeID = x,
FirstName = (new string[] { "Nancy", "Andrew", "Janet", "Margaret", "Steven"
})[new Random().Next(5)],
LastName = (new string[] { "Davolio", "Fuller", "Leverling", "Peacock",
"Buchanan" })[new Random().Next(5)],
```



```

Title = (new string[] { "Sales Representative", "Vice President, Sales",
"Sales Manager",
"Inside Sales Coordinator" })[new Random().Next(4)],
HireDate = DateTime.Now.AddDays(-x),
}).ToList();
}
public async Task Print()
{
await this.DefaultGrid.Print();
}
public class EmployeeData
{
public int? EmployeeID { get; set; }
public string FirstName { get; set; }
public string LastName { get; set; }
public string Title { get; set; }
public DateTime? HireDate { get; set; }
}
}

```

Similarly all the public methods of the DataGrid can be invoked. The available public methods can be found in this [link](#).

Change datasource dynamically in Blazor DataGrid Component

You can change the [DataSource](#) of the datagrid component dynamically through an external button.

This is demonstrated in the below sample code where the [DataSource](#) is dynamically modified using the bounded property,

ASPX-CS

```

@using Syncfusion.Blazor.Grids
@using Syncfusion.Blazor.Buttons
<SfButton OnClick="Change">Change data source dynamically</SfButton>
<SfGrid DataSource="@Orders" AllowPaging="true">
<GridPageSettings PageSize="8"></GridPageSettings>
<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" Type=ColumnType.Date TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumns>
</SfGrid>
@code{
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,

```

```

CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
OrderDate = DateTime.Now.AddDays(-x),
}).ToList();
}
public void Change()
{
// Data source is modified dynamically
this.Orders = Enumerable.Range(1, 45).Select(x => new Order()
{
OrderID = 100 + x,
CustomerID = (new string[] { "CHOPS", "HANAR", "SUPRD", "TOMSP", "VINET"
})[new Random().Next(5)],
Freight = 1.1 * x,
OrderDate = DateTime.Now.AddDays(-x),
}).ToList();
}
public class Order
{
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
}

```

The following GIF represents DataGrid data source modified dynamically on button click,

Change data source dynamically

Order ID	Customer Name	Order Date	Freight
1001	BOLID	9/10/2019	\$2.10
1002	ALFKI	9/9/2019	\$4.20
1003	BOLID	9/8/2019	\$6.30
1004	BLONP	9/7/2019	\$8.40
1005	ANTON	9/6/2019	\$10.50
1006	ALFKI	9/5/2019	\$12.60
1007	BLONP	9/4/2019	\$14.70
1008	ANANTR	9/3/2019	\$16.80

K < 1 2 3 4 5 6 7 8 9 10 > X
1 of 10 pages (75 items)

Custom control in datagrid toolbar in Blazor DataGrid Component

You can render custom controls inside the datagrid's toolbar area. This can be achieved by initializing the custom controls within the Template property of the Toolbar component. This toolbar component is defined inside the datagrid component.

This is demonstrated in the below sample code where Autocomplete component is rendered inside the DataGrid's toolbar and is used for performing search operation on the datagrid,

ASPX-CS

```
@using Syncfusion.Blazor.Grids
@using Syncfusion.Blazor.Navigations
@using Syncfusion.Blazor.DropDowns
<SfGrid DataSource="@Orders" AllowPaging="true" @ref="Grid">
  <GridPageSettings PageSize="8"></GridPageSettings>
  <GridEvents TValue="Order"></GridEvents>
  <SfToolbar>
    <ToolbarItems>
      <ToolbarItem Type="ItemType.Input">
        <Template>
          <SfAutoComplete Placeholder="Search Customer Name" TItem="CustomerDetails"
            TValue="string" DataSource="@Customers">
            <AutoCompleteEvents ValueChange="OnSearch" TValue="string"
              TItem="CustomerDetails"></AutoCompleteEvents>
            <AutoCompleteFieldSettings Value="Name"></AutoCompleteFieldSettings>
          </SfAutoComplete>
        </Template>
      </ToolbarItem>
    </ToolbarItems>
  </SfToolbar>
  <GridColumns>
    <GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
      TextAlign="TextAlign.Right" Width="120"></GridColumn>
    <GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
      Width="150"></GridColumn>
    <GridColumn Field=@nameof(Order.OrderDate) HeaderText="Order Date"
      Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
      Width="130"></GridColumn>
    <GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
      TextAlign="TextAlign.Right" Width="120"></GridColumn>
  </GridColumns>
</SfGrid>
@code{
  public class CustomerDetails
  {
    public string Name { get; set; }
    public int Id { get; set; }
  }
  List<CustomerDetails> Customers = new List<CustomerDetails>
  {
    new CustomerDetails() { Name = "ALFKI", Id = 1 },
    new CustomerDetails() { Name = "ANANTR", Id = 2 },
    new CustomerDetails() { Name = "ANTON", Id = 3 },
    new CustomerDetails() { Name = "BLONP", Id = 4 },
    new CustomerDetails() { Name = "BOLID", Id = 5 }
  };
  private SfGrid<Order> Grid;
```

```

public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
    Orders = Enumerable.Range(1, 75).Select(x => new Order()
    {
        OrderID = 1000 + x,
        CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
    })[new Random().Next(5)],
        Freight = 2.1 * x,
        OrderDate = DateTime.Now.AddDays(-x),
    }).ToList();
}
public async Task
OnSearch(Syncfusion.Blazor.DropDowns.ChangeEventArgs<string, CustomerDetails>
args)
{
    await this.Grid.Search(args.Value);
}
public class Order
{
    public int? OrderID { get; set; }
    public string CustomerID { get; set; }
    public DateTime? OrderDate { get; set; }
    public double? Freight { get; set; }
}
}

```

The following GIF represents the search operation performed on the datagrid using the Autocomplete component rendered in the toolbar,

Search				
Order ID	Customer Name	Order Date	Freight	
1001	BOLID	9/15/2019	\$2.10	
1002	BLONP	9/14/2019	\$4.20	
1003	BOLID	9/13/2019	\$6.30	
1004	BLONP	9/12/2019	\$8.40	
1005	ANANTR	9/11/2019	\$10.50	
1006	ANTON	9/10/2019	\$12.60	
1007	BLONP	9/9/2019	\$14.70	
1008	ALFKI	9/8/2019	\$16.80	
<div> K < 1 2 3 4 5 6 7 8 9 10 > X </div> <div>1 of 10 pages (75 items)</div>				

DataGrid customization in Blazor DataGrid Component

It is possible to customize the default styles of the DataGrid component. This can be achieved by adding class dynamically to the columns using the `AddClass` method of the [QueryCellInfo](#) event. Then the required styles are added to this class.

This is demonstrated in the below sample code,

ASPX-CS

```
@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders" AllowPaging="true">
  <GridEvents QueryCellInfo="QueryCellInfoHandler"
    TValue="Order"></GridEvents>
  <GridPageSettings PageSize="8"></GridPageSettings>
  <GridColumns>
    <GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
      TextAlign="TextAlign.Right" Width="120"></GridColumn>
    <GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
      Width="150"></GridColumn>
    <GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
      Format="d" Type=ColumnType.Date TextAlign="TextAlign.Right"
      Width="130"></GridColumn>
    <GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
      TextAlign="TextAlign.Right" Width="120"></GridColumn>
  </GridColumns>
</SfGrid>
<style>
.e-grid .e-gridcontent .e-rowcell.above-40 {
color: green;
}
.e-grid .e-gridcontent .e-rowcell.above-20 {
color: blue;
}
.e-grid .e-gridcontent .e-rowcell.below-20 {
color: red;
}
</style>
@code{
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 12 * x,
OrderDate = DateTime.Now.AddDays(-x),
}).ToList();
}
public class Order
{
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
```

```

public void QueryCellInfoHandler(QueryCellInfoEventArgs<Order> args) {
    if (args.Data.Freight > 40)
    {
        args.Cell.AddClass(new string[] { "above-40" });
    }
    else if (args.Data.Freight > 20 && args.Data.Freight <= 40)
    {
        args.Cell.AddClass(new string[] { "above-20" });
    }
    else
    {
        args.Cell.AddClass(new string[] { "below-20" });
    }
}

```

<!-- You can also apply style directly to the DataGrid using the `SetAttribute` method in the [QueryCellInfo](#) event. But, this will override the default styles of the grid.

This is demonstrated in the below sample code,

ASPX-CS

-->

ASPX-CS

```

@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders" Toolbar="@ (new List<string>() { "Add",
"Delete", "Edit", "Update", "Cancel" })" AllowPaging="true">
<GridEvents OnActionBegin="OnActionBegin" TValue="Order"></GridEvents>
<GridEditSettings AllowAdding="true" AllowEditing="true"
AllowDeleting="true"></GridEditSettings>
<GridPageSettings PageSize="8"></GridPageSettings>
<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
IsPrimaryKey="true" TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" Type=ColumnType.Date TextAlign="TextAlign.Right"
Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumns>
</SfGrid>
@code{
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
    Orders = Enumerable.Range(1, 75).Select(x => new Order()
    {
        OrderID = 1000 + x,

```

```

CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
    })[new Random().Next(5)],
Freight = 12 * x,
OrderDate = DateTime.Now.AddDays(-x),
}).ToList();
}
public class Order
{
    public int? OrderID { get; set; }
    public string CustomerID { get; set; }
    public DateTime? OrderDate { get; set; }
    public double? Freight { get; set; }
}
public void OnActionBegin(ActionEventArgs<Order> args)
{
    if (args.RequestType == Syncfusion.Blazor.Grids.Action.Add)
    {
        args.Cancel = true;
    }
}
}

```

Get index value of selected rowcell in Blazor DataGrid Component

You can get the index value of a selected rowcell or row by using the [GetSelectedRowCellIndexes](#) method of the DataGrid component.

This is demonstrated in the below sample code where the [GetSelectedRowCellIndexes](#) method is called on button click which returns the selected rowcell indexes,

ASPX-CS

```

@using Syncfusion.Blazor.Buttons
@using Syncfusion.Blazor.Grids
<SfButton OnClick="SelectedRowCellIndex" CssClass="e-primary"
    IsPrimary="true" Content="Get selected rowcell index"></SfButton>
<SfGrid @ref="DefaultGrid" DataSource="@Employees">
    <GridSelectionSettings Mode=SelectionMode.Cell></GridSelectionSettings>
    <GridColumns>
        <GridColumn Field=@nameof(EmployeeData.EmployeeID)
            TextAlign="TextAlign.Center" HeaderText="Employee ID"
            Width="120"></GridColumn>
        <GridColumn Field=@nameof(EmployeeData.FirstName) HeaderText="First Name"
            ShowInColumnChooser="false" Width="130"></GridColumn>
        <GridColumn Field=@nameof(EmployeeData.LastName) HeaderText="Last Name"
            Width="130"></GridColumn>
        <GridColumn Field=@nameof(EmployeeData.Title) HeaderText="Title"
            Width="120"></GridColumn>
        <GridColumn Field=@nameof(EmployeeData.HireDate) HeaderText="Hire Date"
            Format="d" TextAlign="TextAlign.Right" Width="150"></GridColumn>
    </GridColumns>
</SfGrid>
@code{
    private SfGrid<EmployeeData> DefaultGrid;
    public List<EmployeeData> Employees { get; set; }
    protected override void OnInitialized()
    {

```

```

Employees = Enumerable.Range(1, 9).Select(x => new EmployeeData()
{
    EmployeeID = x,
    FirstName = (new string[] { "Nancy", "Andrew", "Janet", "Margaret", "Steven"
    })[new Random().Next(5)],
    LastName = (new string[] { "Davolio", "Fuller", "Leverling", "Peacock",
    "Buchanan" })[new Random().Next(5)],
    Title = (new string[] { "Sales Representative", "Vice President, Sales",
    "Sales Manager",
    "Inside Sales Coordinator" })[new Random().Next(4)],
    HireDate = DateTime.Now.AddDays(-x),
}).ToList();
}
public async Task SelectedRowIndex()
{
    var value = await this.DefaultGrid.GetSelectedRowIndex();
}
public class EmployeeData
{
    public int? EmployeeID { get; set; }
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public string Title { get; set; }
    public DateTime? HireDate { get; set; }
}
}

```

For getting the rowcell index value, the [Mode](#) property of the [GridSelectionSettings](#) component should be set as **Cell**.

Select rows based on certain condition in Blazor DataGrid Component

You can select specific rows in the datagrid based on some conditions by using the [SelectRows](#) method in the [DataBound](#) event of the DataGrid component.

This is demonstrated in the below sample code where the index value of datagrid rows with **Freight** column value greater than 10 are stored in the [RowDataBound](#) event and then selected in the [DataBound](#) event,

ASPX-CS

```

@using Syncfusion.Blazor.Grids
<SfGrid @ref="DefaultGrid" DataSource="@Orders" AllowPaging="true">
  <GridSelectionSettings
    Type="SelectionMode.Multiple"></GridSelectionSettings>
  <GridEvents RowDataBound="OnRowDataBound" DataBound="OnDataBound"
    TValue="Order"></GridEvents>
  <GridPageSettings PageSize="8"></GridPageSettings>
  <GridColumns>
    <GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
      TextAlign="TextAlign.Right" Width="120"></GridColumn>
    <GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
      TextAlign="TextAlign.Right" Width="120"></GridColumn>
    <GridColumn Field=@nameof(Order.OrderDate) HeaderText="Order Date"
      Format="d" Type=ColumnType.Date TextAlign="TextAlign.Right"
      Width="120"></GridColumn>
  </GridColumns>
</SfGrid>

```



```

<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumn>
</SfGrid>
@code{
private SfGrid<Order> DefaultGrid;
public List<Order> Orders { get; set; }
public List<double> SelectedNodeIndex = new List<double>();
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.5 * x,
OrderDate = DateTime.Now.AddDays(-x),
}).ToList();
}
public class Order
{
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
public async Task OnDataBound(object args)
{
// The filtered index values are selected
await this.DefaultGrid.SelectRows(SelectedNodeIndex.ToArray());
}
public void OnRowDataBound(RowDataBoundEventArgs<Order> args)
{
// Freight values greater than 10 are filtered by comparing the primary
column values
if (args.Data.Freight > 10)
{
var dataSource = this.DefaultGrid.DataSource;
var index = 0;
foreach (var data in dataSource)
{
if (data.OrderID == args.Data.OrderID)
{
SelectedNodeIndex.Add(index);
break;
}
index++;
}
}
}
}
}

```

Customize column menu icon in Blazor DataGrid Component

You can customize the column menu icon by overriding the default icon class `.e-icons.e-columnmenu` with the `content` property.

CSS

```
.e-grid .e-columnheader .e-icons.e-columnmenu::before {
content: "\e941";
}
```

This is demonstrated in the below sample code,

ASPX-CS

```
@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders" AllowPaging="true" ShowColumnMenu="true">
<GridPageSettings PageSize="8"></GridPageSettings>
<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
TextAlign="TextAlign.Center" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
TextAlign="TextAlign.Center" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" Type=ColumnType.Date TextAlign="TextAlign.Center"
Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Center" Width="120"></GridColumn>
</GridColumns>
</SfGrid>
@code{
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.5 * x,
OrderDate = DateTime.Now.AddDays(-x),
}).ToList();
}
public class Order
{
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
}
<style>
.e-grid .e-columnheader .e-icons.e-columnmenu::before {
.content: "\e84f";
}
</style>
```

The following image represents datagrid with customized column menu icon

Order ID	Customer Name	Order Date	Freight
1001		9/16/2019	\$2.50
1002		9/15/2019	\$5.00
1003		9/14/2019	\$7.50
1004	BOLID	9/13/2019	\$10.00
1005	ANANTR	9/12/2019	\$12.50
1006	ANANTR	9/11/2019	\$15.00
1007	ALFKI	9/10/2019	\$17.50
1008	ALFKI	9/9/2019	\$20.00

K < 1 2 3 4 5 6 7 8 9 10 > X 1 of 10 pages (75 items)

How to Group the Column Chooser Items

The [GridColumnChooserItemGroup](#) component helps to segregate the column chooser items as group. You can define column's group name by using the [Title](#) property of [GridColumnChooserItemGroup](#) directive.

The following code example demonstrates the default column chooser items as group.

CSHARP

```

@using Syncfusion.Blazor.Grids
@using Syncfusion.Blazor.Buttons
<SfGrid @ref="grid" TValue="OrdersDetails" DataSource="@GridData"
ShowColumnChooser="true" Toolbar="@ ( new List<string>() { "ColumnChooser" })"
AllowPaging="true">
  <GridColumnChooserSettings>
    <Template>
      @{
        var ContextData = context as ColumnChooserTemplateContext;
      }
      @if (ShouldRenderGroup("Order Details", ContextData.Columns))
      {
        <GridColumnChooserItemGroup Title="Order Details">
          @foreach (var column in GetGroupColumns("Order Details",
            ContextData.Columns))
          {
            <GridColumnChooserItem Column="column"></GridColumnChooserItem>
          }
        </GridColumnChooserItemGroup>
      }
      @if (ShouldRenderGroup("Ship Details", ContextData.Columns))
      {
        <GridColumnChooserItemGroup Title="Ship Details">

```

```

@foreach (var column in GetGroupColumns("Ship Details",
ContextData.Columns))
{
<GridColumnChooserItem Column="column"></GridColumnChooserItem>
}
</GridColumnChooserItemGroup>
}
@if (ShouldRenderGroup("Date Details", ContextData.Columns))
{
<GridColumnChooserItemGroup Title="Date Details">
@foreach (var column in GetGroupColumns("Date Details",
ContextData.Columns))
{
<GridColumnChooserItem Column="column"></GridColumnChooserItem>
}
</GridColumnChooserItemGroup>
}
</Template>
<FooterTemplate>
@{
var ContextFooterData = context as ColumnChooserFooterTemplateContext;
var visibles = ContextFooterData.Columns.Where(x => x.Visible).Select(x =>
x.HeaderText).ToArray();
var hiddens = ContextFooterData.Columns.Where(x => !x.Visible).Select(x =>
x.HeaderText).ToArray();
}
<SfButton IsPrimary="true" OnClick="@ (async () => {
await grid.ShowColumnsAsync(visibles);
await grid.HideColumnsAsync(hiddens); }) ">
Submit
</SfButton>
<SfButton @onclick="@ (async () => await
ContextFooterData.CancelAsync()) ">Abort</SfButton>
</FooterTemplate>
</GridColumnChooserSettings>
<GridColumns>
<GridColumn Field=@nameof(OrdersDetails.OrderID) HeaderText="Order ID"
TextAlign="TextAlign.Right" IsPrimaryKey="true" Width="120"></GridColumn>
<GridColumn Field=@nameof(OrdersDetails.CustomerID) HeaderText="Customer
Name" Width="150"></GridColumn>
<GridColumn Field=@nameof(OrdersDetails.OrderDate) HeaderText="Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(OrdersDetails.Freight) HeaderText="Freight"
Format="C2" TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(OrdersDetails.ShippedDate) HeaderText="Shipped
Date" Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="150"></GridColumn>
<GridColumn Field=@nameof(OrdersDetails.ShipCountry) HeaderText="Ship
Country" Visible="false" Width="150"></GridColumn>
<GridColumn Field=@nameof(OrdersDetails.ShipCity) HeaderText="Ship City"
Visible="false" Width="150"></GridColumn>
</GridColumns>
</SfGrid>
@code
{
public List<OrdersDetails> GridData { get; set; }
}

```

```

SfGrid<OrdersDetails> grid { get; set; }
IDictionary<string, string[]> groups = new Dictionary<string, string[]>()
{
    { "Order Details", new string[] { "OrderID", "CustomerID", "Freight" } }, {
    "Ship Details", new string[] { "ShipCountry", "ShipCity" } }, { "Date
    Details", new string[] { "OrderDate", "ShippedDate" } }
};
private GridColumn GetColumn(string field, List<GridColumn> columns)
{
    GridColumn column = null;
    if (columns.Any(x => { column = x; return x.Field == field; }))
    {
        return column;
    }
    return null;
}
private bool ShouldRenderGroup(string title, List<GridColumn> columns)
{
    return groups[title].Any(x => columns.Any(y => y.Field == x));
}
private List<GridColumn> GetGroupColumns(string title, List<GridColumn>
columns)
{
    return columns.Where(x => groups[title].Contains(x.Field)).ToList();
}
protected override void OnInitialized()
{
    GridData = Enumerable.Range(1, 75).Select(x => new OrdersDetails()
    {
        OrderID = 1000 + x,
        CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
        })[new Random().Next(5)],
        Freight = 2.1 * x,
        OrderDate = DateTime.Now.AddDays(-x),
        ShippedDate = DateTime.Now.AddDays(+x),
        ShipCountry = (new string[] { "Denmark", "Brazil", "Germany", "Austria"
        })[new Random().Next(4)],
        ShipCity = (new string[] { "Berlin", "Madrid", "Marseille" })[new
        Random().Next(3)]
    }).ToList();
}
public class OrdersDetails
{
    public int? OrderID { get; set; }
    public string CustomerID { get; set; }
    public DateTime? OrderDate { get; set; }
    public double? Freight { get; set; }
    public DateTime? ShippedDate { get; set; }
    public string ShipCountry { get; set; }
    public string ShipCity { get; set; }
}
}

```

					Columns ▾
Order ID	Customer Name	Order Date	Freight	Shipped Date	
1001	ANANTR	9/30/2021	\$2.10	10/2/2021	
1002	ALFKI	9/29/2021	\$4.20	10/3/2021	
1003	BLONP	9/28/2021	\$6.30	10/4/2021	
1004	ANTON	9/27/2021	\$8.40	10/5/2021	
1005	ALFKI	9/26/2021	\$10.50	10/6/2021	
1006	ALFKI	9/25/2021	\$12.60	10/7/2021	
1007	ANTON	9/24/2021	\$14.70	10/8/2021	
1008	BLONP	9/23/2021	\$16.80	10/9/2021	
1009	ALFKI	9/22/2021	\$18.90	10/10/2021	
1010	ANANTR	9/21/2021	\$21.00	10/11/2021	
1011	BLONP	9/20/2021	\$23.10	10/12/2021	
1012	ANTON	9/19/2021	\$25.20	10/13/2021	

« < 1 2 3 4 5 6 7 > »

1 of 7 pages (75 items)

Calculate column value based on other columns in Blazor DataGrid

You can calculate the values for a datagrid column based on other column values by using the **context** parameter in the [Template](#) property of the [GridColumn](#) component. Inside the [Template](#), you can access the column values using the implicit named parameter **context** and then calculate the values for the new column as required.

This is demonstrated in the below sample code where the value for **FinalCost** column is calculated based on the values of **ManfCost** and **LabCost** columns,

ASPX-CS

```
@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders" AllowPaging="true">
  <GridPageSettings PageSize="8"></GridPageSettings>
  <GridColumn>
    <GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
      TextAlign="TextAlign.Center" Width="120"></GridColumn>
    <GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
      TextAlign="TextAlign.Center" Width="120"></GridColumn>
    <GridColumn Field=@nameof(Order.OrderDate) HeaderText="Order Date"
      Format="d" Type=ColumnType.Date TextAlign="TextAlign.Center"
      Width="120"></GridColumn>
    <GridColumn Field=@nameof(Order.ManfCost) HeaderText="Manufacturing Cost"
      Format="C2" TextAlign="TextAlign.Center" Width="120"></GridColumn>
    <GridColumn Field=@nameof(Order.LabCost) HeaderText="Labor Cost" Format="C2"
      TextAlign="TextAlign.Center" Width="120"></GridColumn>
    <GridColumn Field=@nameof(Order.FinalCost) HeaderText="Final price"
      Format="C2" TextAlign="TextAlign.Center" Width="120">
      <Template>
        @{
          var value = (context as Order);
          var finalAmount = value.ManfCost + value.LabCost;
          <div>${finalAmount}</div>
        }
      </Template>
    </GridColumn>
  </GridColumn>
</SfGrid>
```

```

</GridColumn>
</GridColumn>
</SfGrid>
@code{
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 25).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
ManfCost = 10 * x,
LabCost = 3 * x,
OrderDate = DateTime.Now.AddDays(-x),
}).ToList();
}
public class Order
{
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public int? ManfCost { get; set; }
public int? LabCost { get; set; }
public double? FinalCost { get; set; }
}
}

```

The following image represents the output of the above sample code,

Order ID	Customer Name	Order Date	Manufacturing ...	Labor Cost	Final price
1001	ALFKI	9/17/2019	\$10.00	\$3.00	\$13
1002	ALFKI	9/16/2019	\$20.00	\$6.00	\$26
1003	ANTON	9/15/2019	\$30.00	\$9.00	\$39
1004	BLONP	9/14/2019	\$40.00	\$12.00	\$52
1005	ANTON	9/13/2019	\$50.00	\$15.00	\$65
1006	ANTON	9/12/2019	\$60.00	\$18.00	\$78
1007	BOLID	9/11/2019	\$70.00	\$21.00	\$91
1008	ANTON	9/10/2019	\$80.00	\$24.00	\$104

K < 1 2 3 4 > X
1 of 4 pages (25 items)

Using dictionary values as datasource in Blazor DataGrid Component

You can assign dictionary values in the datagrid's data source by accessing them using **KeyValuePair** data type inside the [Template](#) property of the [GridColumn](#) component

This is demonstrated in the below sample code where **ShipName** is defined as Dictionary value and it is accessed inside the template property of the [GridColumn](#) using **KeyValuePair** data type. The key value is compared with the **OrderID** column value and based on that the value is displayed,

ASPX-CS

```
@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@GridData" AllowPaging="true">
<GridPageSettings PageSize="6"></GridPageSettings>
<GridColumn>
<GridColumn Field=@nameof(OrderDetail.OrderID) HeaderText="Order ID"
TextAlign="@TextAlign.Right" IsPrimaryKey="true" Width="120"></GridColumn>
<GridColumn Field=@nameof(OrderDetail.OrderDate) HeaderText=" Order Date"
Format="d" Type=ColumnType.Date TextAlign="@TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(OrderDetail.Freight) HeaderText="Freight"
Format="C2" AllowEditing="false" TextAlign="@TextAlign.Right"
Width="120"></GridColumn>
<GridColumn Field=@nameof(OrderDetail.ShipCountry) HeaderText="Ship Country"
TextAlign="@TextAlign.Right" Width="150"></GridColumn>
<GridColumn Field=@nameof(OrderDetail.ShipCity) HeaderText="Ship City"
TextAlign="@TextAlign.Right" Width="150"></GridColumn>
<GridColumn Field=@nameof(OrderDetail.ShipName) HeaderText="Ship Name"
TextAlign="@TextAlign.Right" Width="150">
<Template>
@{
var Details = context as OrderDetail;
var address = Details.ShipName.Select(kvp => (kvp.Key == Details.OrderID) ?
kvp.Value.ToString() : "");
<p>@string.Join("", address)</p>
}
</Template>
</GridColumn>
</GridColumn>
</SfGrid>
@code{
private static Dictionary<int, string> ShipDetails = new Dictionary<int,
string>()
{
{ 1001, "White Clover Markets" },
{ 1002, "Vins et alcools Chevalier" },
{ 1003, "Save-a-lot Markets" },
{ 1004, "Vins et alcools Chevalier" },
{ 1005, "Save-a-lot Markets" },
{ 1006, "Victuailles en stock" },
{ 1007, "Rattlesnake Canyon Grocery" },
{ 1008, "Victuailles en stock" },
{ 1009, "Rattlesnake Canyon Grocery" },
{ 1010, "Blondel et fils" }
};
List<OrderDetail> GridData = new List<OrderDetail>
{
new OrderDetail { OrderID = 1001, Freight = 2.3, OrderDate = new
DateTime(1991, 05, 15), ShipCity = "Seattle", ShipName = ShipDetails,
ShipCountry = "United States" },
```



```
new OrderDetail { OrderID = 1002, Freight = 3.3, OrderDate = new
DateTime(1990, 04, 04), ShipCity = "Reims", ShipName = ShipDetails,
ShipCountry = "France" },
new OrderDetail { OrderID = 1003, Freight = 5.3, OrderDate = new
DateTime(1957, 11, 30), ShipCity = "Boise", ShipName = ShipDetails,
ShipCountry = "United States" },
new OrderDetail { OrderID = 1004, Freight = 2.3, OrderDate = new
DateTime(1992, 07, 14), ShipCity = "Reims", ShipName = ShipDetails,
ShipCountry = "France" },
new OrderDetail { OrderID = 1005, Freight = 5.3, OrderDate = new
DateTime(1927, 01, 20), ShipCity = "Boise", ShipName = ShipDetails,
ShipCountry = "United States" },
new OrderDetail { OrderID = 1006, Freight = 9.3, OrderDate = new
DateTime(1920, 02, 15), ShipCity = "Lyon", ShipName = ShipDetails,
ShipCountry = "France" },
new OrderDetail { OrderID = 1007, Freight = 6.3, OrderDate = new
DateTime(1951, 12, 08), ShipCity = "Albuquerque", ShipName = ShipDetails,
ShipCountry = "United States" },
new OrderDetail { OrderID = 1008, Freight = 4.3, OrderDate = new
DateTime(1930, 10, 22), ShipCity = "Lyon", ShipName = ShipDetails,
ShipCountry = "France" },
new OrderDetail { OrderID = 1009, Freight = 6.3, OrderDate = new
DateTime(1953, 02, 18), ShipCity = "Albuquerque", ShipName = ShipDetails,
ShipCountry = "United States" },
new OrderDetail { OrderID = 1010, Freight = 4.3, OrderDate = new
DateTime(1923, 01, 28), ShipCity = "Strasbourg", ShipName = ShipDetails,
ShipCountry = "France" }
};
public class OrderDetail
{
public int? OrderID { get; set; }
public double? Freight { get; set; }
public string ShipCity { get; set; }
public DateTime OrderDate { get; set; }
public string ShipCountry { get; set; }
public Dictionary<int, string> ShipName { get; set; }
}
}
```

The following image represent the datagrid rendered using the above sample code,

Order ID	Order Date	Freight	Ship Country	Ship City	Ship Name
1001	5/15/1991	\$2.30	United States	Seattle	White Clover Markets
1002	4/4/1990	\$3.30	France	Reims	Vins et alcools Chevalier
1003	11/30/1957	\$5.30	United States	Boise	Save-a-lot Markets
1004	7/14/1992	\$2.30	France	Reims	Vins et alcools Chevalier
1005	1/20/1927	\$5.30	United States	Boise	Save-a-lot Markets
1006	2/15/1920	\$9.30	France	Lyon	Victuailles en stock
⏪ < 1 2 > ⏩					1 of 2 pages (10 items)

Upgrade Application To Latest Version in Blazor DataGrid Component

Step 1: Update the latest Syncfusion blazor [NuGet](#) from NuGet package manager in your application.

The screenshot shows the NuGet Package Manager window for 'Syncfusion.EJ2.Blazor'. The 'Updates' tab is active, showing a list of packages with a checkmark next to 'Syncfusion.EJ2.Blazor' and a version update from 'v17.3.0.18-beta' to 'v17.3.0.19-beta'. The right pane shows the package details, including the 'Installed' version, the 'Latest prerelease' version, and a description of the components included: GRIDS (DataGrid, Pivot Table, Tree Grid) and DATA VISUALIZATION (Charts, Circular Gauge, Diagram, Heatmap chart).

Compatible .NET version

Syncfusion Blazor components in the latest version `{:nuget-version:}` are compatible with the latest version of .NET Core 3.1. So, we suggest you to upgrade the .NET Core 3.1 SDK in your machine before upgrading to the latest version.

Client resource file references

Ensure our CSS files have been properly configured in your application.

- If you use the Blazor server app, add the following style file references in `~/Pages/_Host.cshtml`.
- If you use the Blazor WebAssembly app, add the following style file references in `~/wwwroot/index.html`.

HTML

```
<link href="_content/Syncfusion.Blazor/styles/bootstrap4.css"
rel="stylesheet" />
```

For production purpose and minimal requirement, Syncfusion provides an option to generate scripts and styles of selective control by using the Custom Resource Generator (CRG) web tool. Refer to this [link](#) for more details on CRG.

Breaking changes

Some changes have been modified in our Blazor samples for each release. So, we suggest you to ensure the breaking changes. Refer to this [release notes](#) for our Blazor components.

Cache problem

Before restoring the NuGet packages, clean the old version Syncfusion.Blazor NuGet package.

The following steps explain how to clean the cache:

1. Delete/clear the package Syncfusion.Blazor from the installed location `{System-driver}\Users\{user-name}\.nuget\packages\syncfusion.blazor`. In Windows, the installed location of Syncfusion.Blazor package can be found using `%userprofile%\nuget\packages\syncfusion.blazor`.
2. Update the latest version of Syncfusion.Blazor NuGet package.

Custom toolbar items with text name same as default toolbar items

You can create the Custom toolbar items with text name same as default toolbar items (Add,Edit,Delete,etc.). But while creating them, they will be considered as default toolbar items which will cause some issues while clicking on it. To overcome this behavior, we suggest you to define the **Id** property for custom toolbar items.

This is demonstrated in the below sample code where we have custom toolbar items with text same as **Add** and **Delete** buttons. These toolbar buttons will be enabled only when GridEditSettings is defined in DataGrid. So custom toolbar will be disabled state considering it as default toolbar item. We have overcome that behavior by defining the Id property.

ASPX-CS

```
@using Syncfusion.Blazor.Grids
@{
    List<Syncfusion.Blazor.Navigations.ItemModel> Toolbaritems = new
    List<Syncfusion.Blazor.Navigations.ItemModel>();
    Toolbaritems.Add(new Syncfusion.Blazor.Navigations.ItemModel() { Text =
    "Add", Id = "add", TooltipText = "Add Record", PrefixIcon = "add" });
```

```

ToolBarItems.Add(new Syncfusion.Blazor.Navigations.ItemModel() { Text =
"Delete", Id = "delete", TooltipText = "Delete Record", PrefixIcon =
"delete" });
}
<SfGrid DataSource="@Orders" @ref="Grid" AllowGrouping="true"
AllowPaging="true" Height="200" Toolbar="ToolBarItems">
<GridEvents OnToolBarClick="ToolBarClickHandler"
TValue="Order"></GridEvents>
<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="150"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumns>
</SfGrid>
@code{
SfGrid<Order> Grid;
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
OrderDate = DateTime.Now.AddDays(-x),
}).ToList();
}
public class Order
{
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
public void ToolBarClickHandler(Syncfusion.Blazor.Navigations.ClickEventArgs
args)
{
if (args.Item.Text == "Add")
{
//perform your actions here
}
if (args.Item.Text == "Delete")
{
//perform your actions here
}
}
}

```

Blazor DataGrid Component inside the Tab with Specific Height

By default, DataGrid will occupy the entire space of the parent element when DataGrid [Height](#) and [Width](#) property is defined as 100%. But if you render the similar DataGrid inside the Tab control, it will consider the entire page and render the DataGrid without horizontal scroller.

To overcome this behavior we suggest you to render a container element enclosing the DataGrid with specific height and set the DataGrid height as 100%.

ASPX-CS

```
@using Syncfusion.Blazor
@using Syncfusion.Blazor.Navigations
@using Syncfusion.Blazor.Grids
<SfTab ID="Ej2Tab" Width="100%">
  <TabItems>
    <TabItem>
      <ChildContent>
        <TabHeader Text="Grid 1"></TabHeader>
        </ChildContent>
        <ContentTemplate>
          <div style="height:300px">
            <SfGrid DataSource="@Orders" Height="100%" Width="100%">
              <GridColumns>
                <GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
                  TextAlign="TextAlign.Right" Width="120"></GridColumn>
                <GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
                  Width="150"></GridColumn>
                <GridColumn Field=@nameof(Order.OrderDate) HeaderText="Order Date"
                  Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
                  Width="130"></GridColumn>
                <GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
                  TextAlign="TextAlign.Right" Width="120"></GridColumn>
              </GridColumns>
            </SfGrid>
          </div>
        </ContentTemplate>
      </TabItem>
      <TabItem>
        <ChildContent>
          <TabHeader Text="Grid 2"></TabHeader>
          </ChildContent>
          <ContentTemplate>
            <div style="height:300px">
              <SfGrid DataSource="@Employees" Height="100%" Width="100%">
                <GridColumns>
                  <GridColumn Field=@nameof(EmployeeData.EmployeeID) HeaderText="ID"
                    Visible="false" TextAlign="TextAlign.Right" Width="120"></GridColumn>
                  <GridColumn Field=@nameof(EmployeeData.FirstName) HeaderText="First Name"
                    Width="150"></GridColumn>
                  <GridColumn Field=@nameof(EmployeeData.LastName) HeaderText="last Name"
                    Width="150"></GridColumn>
                  <GridColumn Field=@nameof(EmployeeData.HireDate) HeaderText="Hire Date"
                    Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
                    Width="130"></GridColumn>
                  <GridColumn Field=@nameof(EmployeeData.Role) HeaderText="Position"
                    Width="120"></GridColumn>
                </GridColumns>
              </SfGrid>
            </div>
          </ContentTemplate>
        </TabItem>
      </TabItems>
    </SfTab>
```

```

</SfGrid>
</div>
</ContentTemplate>
</TabItem>
</TabItems>
</SfTab>
@code {
public List<Order> Orders { get; set; }
public List<EmployeeData> Employees { get; set; }
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
OrderDate = DateTime.Now.AddDays(-x),
}).ToList();
Employees = Enumerable.Range(1, 75).Select(x => new EmployeeData()
{
EmployeeID = x,
FirstName = (new string[] { "Nancy", "Andrew", "Janet", "Margaret", "Steven"
})[new Random().Next(5)],
LastName = (new string[] { "Davolio", "Fuller", "Leveringg", "peacock",
"Smith" })[new Random().Next(5)],
Role = (new string[] { "Sales Representative", "Sales Representative",
"Sales Manager", "HR Manager", "Inside Sales Coordinator" })[new
Random().Next(5)],
HireDate = DateTime.Now.AddDays(-x),
}).ToList();
}
public class Order
{
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
public class EmployeeData
{
public int? EmployeeID { get; set; }
public string FirstName { get; set; }
public string LastName { get; set; }
public string Role { get; set; }
public DateTime? HireDate { get; set; }
}
}

```

Custom data source & filtering for DropDownList in Blazor DataGrid

You can provide custom data source and enable filter option for DropDownList while performing DataGrid editing by using the [Edit](#) params property.

While setting new data source for DropDownList using [Edit](#) params, you must also specify a new [Query](#) property for DropDownList.

This is demonstrated in the below sample code,

ASPX-CS

```
@using Syncfusion.Blazor.Grids
@using Syncfusion.Blazor.DropDowns
<SfGrid DataSource="@Orders" AllowPaging="true" Toolbar="@ (new
List<string>() {"Add", "Edit", "Delete", "Update", "Cancel"}) ">
<GridEditSettings AllowAdding="true" AllowEditing="true"
AllowDeleting="true"></GridEditSettings>
<GridColumn>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
IsPrimaryKey="true" TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
EditType="EditType.DropDownEdit" EditorSettings="@CustomerIDEditParams"
Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" Type=ColumnType.Date Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumn>
</SfGrid>
@code{
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.5 * x,
OrderDate = DateTime.Now.AddDays(-x),
}).ToList();
}
public IEditorSettings CustomerIDEditParams = new DropDownEditCellParams
{
Params = new DropDownListModel<object, object>() { DataSource = LocalData,
AllowFiltering = true }
};
public class Order
{
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
public class Country
{
public string CustomerID { get; set; }
public int? CountryId { get; set; }
}
public static List<Order> LocalData = new List<Order> {
new Order() { CustomerID= "United States" },
new Order() { CustomerID= "Australia" },
new Order() { CustomerID= "India" }
};
};
```

```
}

```

Single click editing with Batch mode in Blazor DataGrid Component

You can make a cell editable on a single click with a [Batch](#) mode of editing in DataGrid, by using the [EditCell](#) method.

Set the [Mode](#) property of [GridSelectionSettings](#) component to **Both** and bind the [CellSelected](#) event to DataGrid. In the [CellSelected](#) event handler, call the [EditCell](#) method based on the clicked cell.

This is demonstrated in the below sample code,

ASPX-CS

```
@using Syncfusion.Blazor.Grids
<SfGrid @ref="GridInstance" AllowPaging="true" DataSource="@Orders"
Toolbar="@ (new List<string>() { "Cancel", "Update" })">
  <GridSelectionSettings
Mode="Syncfusion.Blazor.Grids.SelectionMode.Both"></GridSelectionSettings>
  <GridEditSettings AllowEditing="true"
Mode="EditMode.Batch"></GridEditSettings>
  <GridEvents CellSelected="CellSelectHandler" TValue="Order"></GridEvents>
  <GridColumns>
    <GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
IsPrimaryKey="true" TextAlign="TextAlign.Right" Width="120"></GridColumn>
    <GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="120"></GridColumn>
    <GridColumn Field=@nameof(Order.OrderDate) HeaderText="Order Date"
EditType="EditType.DatePickerEdit" Format="d" TextAlign="TextAlign.Right"
Width="130" Type="ColumnType.Date"></GridColumn>
    <GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" EditType="EditType.NumericEdit"
Width="120"></GridColumn>
    <GridColumn Field=@nameof(Order.ShipCountry) HeaderText="Ship Country"
EditType="EditType.DropDownEdit" Width="150"></GridColumn>
  </GridColumns>
</SfGrid>
@code{
SfGrid<Order> GridInstance { get; set; }
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
OrderDate = DateTime.Now.AddDays(-x),
ShipCountry = (new string[] { "USA", "UK", "CHINA", "RUSSIA", "INDIA" })[new
Random().Next(5)]
}).ToList();
}
public async Task CellSelectHandler(CellSelectEventArgs<Order> args)
{
//get selected cell index
var CellIndexes = await GridInstance.GetSelectedRowCellIndexes();
//get the row and cell index

```



```

var CurrentEditRowIndex = CellIndexes[0].Item1;
var CurrentEditCellIndex = (int)CellIndexes[0].Item2;
//get the available fields
var fields = await GridInstance.GetColumnFieldNames();
// edit the selected cell using the cell index and column name
await GridInstance.EditCell(CurrentEditRowIndex,
fields[CurrentEditCellIndex]);
}
public List<Order> Orders { get; set; }
public class Order {
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
public string ShipCountry { get; set; }
}
}

```

Cascading DropDownList in Blazor DataGrid Component Editing

You can achieve the Cascading DropDownList with datagrid editing by using the [EditTemplate](#) property of the [GridColumn](#) component.

This is demonstrated in the below sample code where cascading dropdownlist is rendered for the **ShipCountry** and **ShipState** column when perform editing in datagrid.

ASPX-CS

```

@using Syncfusion.Blazor.Grids
@using Syncfusion.Blazor.DropDowns
<SfGrid @ref="GridRef" AllowPaging="true" DataSource="@GridData"
ShowColumnChooser="true" Toolbar="@ (new List<string>() {
"Add", "Edit", "Delete", "Update", "Cancel" }) ">
<GridEvents OnActionBegin="OnActionBegin" TValue="Orders"></GridEvents>
<GridEditSettings AllowEditing="true" AllowDeleting="true"
AllowAdding="true" Mode="@EditMode.Normal"></GridEditSettings>
<GridColumn>
<GridColumn Field=@nameof(Orders.OrderID) HeaderText="Order ID"
IsPrimaryKey="true" TextAlign="@TextAlign.Center" Width="140"></GridColumn>
<GridColumn Field=@nameof(Orders.Freight) HeaderText="Freight"
EditType="EditType.NumericEdit" Format="C2" Width="140"
TextAlign="@TextAlign.Right"></GridColumn>
<GridColumn Field=@nameof(Orders.OrderDate) HeaderText="Order Date"
EditType="EditType.DatePickerEdit" Format="d" Type="ColumnType.Date"
Width="160"></GridColumn>
<GridColumn Field=@nameof(Orders.ShipCountry) HeaderText="Ship Country"
EditType="EditType.DropDownEdit" Width="150">
<EditTemplate>
<SfDropDownList ID="ShipCountry" Placeholder="Select a Country"
TItem="string" TValue="string" @bind-Value="@((context as
Orders).ShipCountry)" DataSource="@Countries">
<DropDownListEvents TValue="string" TItem="string"
ValueChange="ValueChange"></DropDownListEvents>
<DropDownListFieldSettings Text="ShipCountry"
Value="ShipCountry"></DropDownListFieldSettings>
</SfDropDownList>
</EditTemplate>

```

```

</GridColumn>
<GridColumn Field=@nameof(Orders.ShipState) HeaderText=" Ship State"
EditType="EditType.DropDownEdit" Width="150">
<EditTemplate>
<SfDropDownList ID="ShipState" Placeholder="Select a State" TItem="string"
Enabled="@Enabled" TValue="string" @bind-Value="@((context as
Orders).ShipState)" DataSource="@States">
<DropDownListFieldSettings Text="ShipState"
Value="ShipState"></DropDownListFieldSettings>
</SfDropDownList>
</EditTemplate>
</GridColumn>
</GridColumns>
</SfGrid>
@code{
SfGrid<Orders> GridRef;
public List<Orders> GridData { get; set; } = new List<Orders>();
public List<string> Countries = new List<string>() { "United States",
"Australia" };
public List<string> States = new List<string>() { "New York", "Virginia",
"Washington", "Queensland", "Tasmania", "Victoria" };
public bool Enabled = false;
protected override void OnInitialized()
{
if (GridData.Count() == 0)
{
int code = 10000;
for (int i = 1; i < 10; i++)
{
GridData.Add(new Orders(code + 1, "ALFKI", i + 0, 2.3 * i, new
DateTime(1991, 05, 15), "United States", "New York"));
GridData.Add(new Orders(code + 2, "ANATR", i + 2, 3.3 * i, new
DateTime(1990, 04, 04), "Australia", "Queensland"));
GridData.Add(new Orders(code + 3, "ANTON", i + 1, 4.3 * i, new
DateTime(1957, 11, 30), "United States", "Virginia"));
GridData.Add(new Orders(code + 4, "BLONP", i + 3, 5.3 * i, new
DateTime(1930, 10, 22), "United States", "Washington"));
GridData.Add(new Orders(code + 5, "BOLID", i + 4, 6.3 * i, new
DateTime(1953, 02, 18), "Australia", "Victoria"));
code += 5;
}
}
}
}
public class Orders
{
public Orders()
{
}
public Orders(long OrderId, string CustomerId, int EmployeeId, double
Freight, DateTime OrderDate, string ShipCountry, string ShipState)
{
this.OrderID = OrderId;
this.CustomerID = CustomerId;
this.EmployeeID = EmployeeId;
this.Freight = Freight;
this.OrderDate = OrderDate;
this.ShipCountry = ShipCountry;
}
}
}

```

```

this.ShipState = ShipState;
}
public long OrderID { get; set; }
public string CustomerID { get; set; }
public int EmployeeID { get; set; }
public double Freight { get; set; }
public DateTime OrderDate { get; set; }
public string ShipCountry { get; set; }
public string ShipState { get; set; }
}
public void
ValueChanged(@Syncfusion.Blazor.DropDowns.ChangeEventArgs<string, string>
args)
{
if (args.Value == "United States")
{
States = new List<string>() { "New York", "Virginia", "Washington" };
}
else if (args.Value == "Australia")
{
States = new List<string>() { "Queensland", "Tasmania", "Victoria" };
}
Enabled = true;
GridRef.PreventRender(false);
}
public void OnActionBegin(ActionEventArgs<Orders> args)
{
if (args.RequestType == Syncfusion.Blazor.Grids.Action.BeginEdit)
{
Enabled = false;
}
}
}
}

```

Editing with template column in Blazor DataGrid Component

You can edit a template column value by defining the [Field](#) property for that particular [GridColumn](#) component

In the below demo, the **CustomerID** column is rendered with the template.

ASPX-CS

```

@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders" AllowPaging="true" Toolbar="@ (new
List<string>() { "Add", "Edit", "Delete", "Cancel", "Update" })"
Height="315">
<GridEditSettings AllowAdding="true" AllowEditing="true"
AllowDeleting="true"></GridEditSettings>
<GridColumn>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
IsPrimaryKey="true" TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
EditType="EditType.DropDownEdit" Width="120">
<Template>
@{
var data = context as Order;

```

```

<a href="#">@data.CustomerID</a>
}
</Template>
</GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
EditType="EditType.DatePickerEdit" Format="d" TextAlign="TextAlign.Right"
Width="130" Type="ColumnType.Date"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
</GridColumns>
</SfGrid>
@code{
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
OrderDate = DateTime.Now.AddDays(-x),
}).ToList();
}
public class Order
{
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
}

```

Hide DataGrid Header in Blazor DataGrid Component

You can hide the DataGrid header by applying the below styles to DataGrid component.

HTML

```

<style>
.e-grid .e-gridheader .e-columnheader{
display: none;
}
</style>

```

if you want to hide the header for particular DataGrid, then you can apply the above styles to that DataGrid using the ID (#Grid.e-grid .e-gridheader .e-columnheader) property value.

Display Custom Tooltip in Blazor DataGrid Cell

You can display custom tooltip in Grid column using [Column Template](#) feature by rendering the [SfTooltip](#) components inside the template.

This is demonstrated in the below sample code we have render the tooltip for **FirstName** column using [Column Template](#).

ASPX-CS

```

@using Syncfusion.Blazor.Grids
@using Syncfusion.Blazor.Popups
<SfGrid DataSource="@Employees">
<GridColumns>
<GridColumn Field=@nameof(EmployeeData.EmployeeID) HeaderText="Employee ID"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(EmployeeData.FirstName) HeaderText="First Name"
Width="130">
<Template>
@{
var employee = (context as EmployeeData);
Count++;
<SfTooltip Target="#txt" @key="@Count">
<TooltipTemplates>
<Content>
@employee.FirstName
</Content>
</TooltipTemplates>
<span id="txt">@employee.FirstName</span>
</SfTooltip>
}
</Template>
</GridColumn>
<GridColumn Field=@nameof(EmployeeData.Title) HeaderText="Title"
Width="120"></GridColumn>
<GridColumn Field=@nameof(EmployeeData.HireDate) HeaderText="Hire Date"
Format="d" TextAlign="TextAlign.Right" Width="150"></GridColumn>
</GridColumns>
</SfGrid>
@code{
public List<EmployeeData> Employees { get; set; }
int Count { get; set; } = 0;
protected override void OnInitialized()
{
Employees = Enumerable.Range(1, 9).Select(x => new EmployeeData()
{
EmployeeID = x,
FirstName = (new string[] { "Nancy", "Andrew", "Janet", "Margaret", "Steven"
})[new Random().Next(5)],
LastName = (new string[] { "Davolio", "Fuller", "Leverling", "Peacock",
"Buchanan" })[new Random().Next(5)],
Title = (new string[] { "Sales Representative", "Vice President, Sales",
"Sales Manager",
"Inside Sales Coordinator" })[new Random().Next(4)],
HireDate = DateTime.Now.AddDays(-x),
}).ToList();
}
public class EmployeeData
{
public int? EmployeeID { get; set; }
public string FirstName { get; set; }
public string LastName { get; set; }
public string Title { get; set; }
public DateTime? HireDate { get; set; }
}
}

```

Styling and appearance in Blazor DataGrid Component

To modify the Grid appearance, you need to override the default CSS of grid. Please find the list of CSS classes and its corresponding section in grid. Also, you have an option to create your own custom theme for all the Syncfusion Blazor components using our [Theme Studio](#).

Section	CSS class	Purpose of CSS class
	-----	-----
Root	e-grid	This classes are in this root element (div) of the grid control.
Header	e-gridheader	This class is added in the root element of header element. In this class, You can override thin line between header and content of the grid.
	e-table	This class is added at 'table' of the grid header. This CSS class makes table width as 100 %.
	e-columnheader	This class is added at 'tr' of the grid header.
	e-headercell	This class is added in 'th' element of grid header. You can override background color of header and border color.
	e-headercelldiv	This class is add in div which present 'th' element in the header. we recommend you to use the e-headercelldiv to override skeleton of header.
Body	e-gridcontent	This class is added at root of body content. This is to override background color of the body.
	e-table	This class is added to table of content. This CSS class makes table width as 100 %.
	e-altrow	This class is added to alternate rows of grid. This is to override alternate row color of the grid.
	e-rowcell	This class is added to all cells in the grid. This is to override cells appearance and styling.
	e-groupcaption	This class is added to the 'td' of group caption which is to change the background color of caption cell.
	e-selectionbackground	This class is added to rowcell's of the grid. This is override selection.
Pager	e-pager	This class is added to root element of the pager. This to change appearance of the background color and color of font.
	e-pagercontainer	This class is added to numeric items of the pager.
	e-parentmsgbar	This class is added to pager info of the pager.
Summary	e-gridfooter	This class is added to root of the summary div.
	e-summaryrow	This class is added to rows of grid summary.
	e-summarycell	This class is added to cells of summary row. This to override background color of summary.

Icons

CSS class	Purpose of CSS class
e-add	This class is added to icon of Add toolbar button.
e-edit	This class is added to icon of Edit toolbar button.

e-delete| This class is added to icon of Delete toolbar button.

e-cancel| This class is added to icon of Cancel toolbar button.

e-update| This class is added to icon of Update toolbar button.

e-excelexport| This class is added to icon of ExcelExport toolbar button.

e-csvexport| This class is added to icon of CsvExport toolbar button.

e-pdfexport| This class is added to icon of PdfExport toolbar button.

e-search-icon| This class is added to icon of Search toolbar button.

e-icon-ascending| This class is added to Sort ascending notify icon in grid header.

e-icon-descending| This class is added to Sort descending notify icon in grid header.

e-icon-filter| This class is added to Filter icon in grid header.

e-icon-group| This class is added to Group icon in grid header.

e-columnmenu| This class is added to ColumnMenu icon in grid header.

e-columnchooser-btn| This class is added to ColumnChooser button icon in grid toolbar.

e-icon-gdownarrow| This class is added to collapse icon in grouped/detail row.

e-icon-grightarrow| This class is added to expand icon in grouped/detail row.

e-icon-first| This class is added to pager arrow icon which makes navigation to first page.

e-icon-prev| This class is added to pager arrow icon which makes navigation to previous page.

e-icon-last| This class is added to pager arrow icon which makes navigation to last page.

e-icon-next| This class is added to pager arrow icon which makes navigation to next page.

e-ddl-icon| This class is added to icon of pager dropdown arrow.

Customize empty grid display message in Blazor DataGrid Component

You can customize the message shown when rendering an empty grid by using the `EmptyRecordTemplate` feature.

This is demonstrated in the below sample code,

ASPX-CS

```
@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders" AllowPaging="true">
  <GridTemplates>
    <EmptyRecordTemplate>
      <span>Custom no record message</span>
    </EmptyRecordTemplate>
  </GridTemplates>
  <GridColumns>
    <GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
      TextAlign="TextAlign.Center" Width="120"></GridColumn>
    <GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
      TextAlign="TextAlign.Center" Width="120"></GridColumn>
    <GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
      Format="d" Type=ColumnType.Date TextAlign="TextAlign.Center"
      Width="120"></GridColumn>
  </GridColumns>
</SfGrid>
```

```

<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Center" Width="120"></GridColumn>
</GridColumn>
</SfGrid>
@code{
public List<Order> Orders { get; set; }
public class Order
{
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
}

```

Saving a new row at a particular index of the blazor datagrid

By default, a newly added row will be saved at the top of the datagrid. You can change it by setting the `args.Index` in [OnActionBegin](#) handler.

The following sample code demonstrates changing the save index of the new row that gets added in the DataGrid component,

ASPX-CS

```

@using Syncfusion.Blazor.Grids
@using Action = Syncfusion.Blazor.Grids.Action
<SfGrid @ref="GridInstance" AllowPaging="true" DataSource="@Orders"
Toolbar="@ (new List<string>() { "Add", "Edit", "Delete", "Update", "Cancel" })">
<GridEditSettings AllowAdding="true" AllowEditing="true"
AllowDeleting="true"
NewRowPosition="NewRowPosition.Bottom"></GridEditSettings>
<GridEvents OnActionBegin="OnActionBegin" TValue="Order"></GridEvents>
<GridColumn>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
IsPrimaryKey="true" TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText="Order Date"
EditType="EditType.DatePickerEdit" Format="d" TextAlign="TextAlign.Right"
Width="130" Type="ColumnType.Date"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" EditType="EditType.NumericEdit"
Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.ShipCountry) HeaderText="Ship Country"
EditType="EditType.DropDownEdit" Width="150"></GridColumn>
</GridColumn>
</SfGrid>
@code{
SfGrid<Order> GridInstance { get; set; }
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,

```



```

CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
OrderDate = DateTime.Now.AddDays(-x),
ShipCountry = (new string[] { "USA", "UK", "CHINA", "RUSSIA", "INDIA" })[new
Random().Next(5)]
}).ToList();
}
public void OnActionBegin(ActionEventArgs<Order> args)
{
if (args.RequestType.Equals(Action.Save) && args.Action == "Add")
{
//Here you can set the custom index for the saved new row. Below calculation
save the new row as last row of current page.
args.Index = (GridInstance.PageSettings.CurrentPage *
GridInstance.PageSettings.PageSize) - 1;
}
}
public class Order
{
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
public string ShipCountry { get; set; }
}
}

```

Filter choice items count for Excel filter in Blazor DataGrid

For better performance, excel filter will take only the first 1000 records for filter choices item list. So, the distinct values of first thousand records will only be displayed in excel filter dialog as filter choices.

To overcome this default behavior, you can customize the count of filter choice items to be displayed in Excel filter dialog by setting the `FilterChoiceCount` in [OnActionBegin](#) event handler.

This is demonstrated in the below sample code,

ASPX-CS

```

@using Syncfusion.Blazor.Grids
<SfGrid @ref="GridInstance" AllowFiltering="true" AllowPaging="true"
DataSource="@Orders">
<GridFilterSettings Type="FilterType.Excel"></GridFilterSettings>
<GridEvents OnActionBegin="OnActionBegin" TValue="Order"></GridEvents>
<GridColumns>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
IsPrimaryKey="true" TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
EditType="EditType.DatePickerEdit" Format="d" TextAlign="TextAlign.Right"
Width="130" Type="ColumnType.Date"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" EditType="EditType.NumericEdit"
Width="120"></GridColumn>

```

```

<GridColumn Field=@nameof(Order.ShipCountry) HeaderText="Ship Country"
EditType="EditType.DropDownEdit" Width="150"></GridColumn>
</GridColumn>
</SfGrid>
@code{
SfGrid<Order> GridInstance { get; set; }
public List<Order> Orders { get; set; }
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
OrderDate = DateTime.Now.AddDays(-x),
ShipCountry = (new string[] { "USA", "UK", "CHINA", "RUSSIA", "INDIA" })[new
Random().Next(5)]
}).ToList();
}
public void OnActionBegin(ActionEventArgs<Order> args)
{
if (args.RequestType.ToString() == "FilterChoiceRequest")
{
args.FilterChoiceCount = 2;    //here you can override the default take
count
}
}
public class Order
{
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
public string ShipCountry { get; set; }
}
}

```

Custom delete confirmation dialog in Blazor DataGrid Component

You can customize the appearance and contents of delete confirmation dialog by rendering a customized [SfDialog](#) instead of the default grid delete confirmation dialog.

This is demonstrated in the below sample code,

ASPX-CS

```

@using Syncfusion.Blazor.Grids
@using Syncfusion.Blazor.Popups
<SfDialog @ref="Dialog" Width="250px" Visible="false" ShowCloseIcon="true"
IsModal="true">
<DialogEvents Closed="Closed"></DialogEvents>
<DialogTemplates>
@*Here you can customize the Header and Content of delete confirmation
dialog*@
<Header> Delete Record</Header>
<Content> You are about to Delete a Record @SelectedData ?</Content>

```

```

</DialogTemplates>
<DialogButtons>
<DialogButton OnClick="@OkClick">
<DialogButtonModel Content="OK" IsPrimary="true"></DialogButtonModel>
</DialogButton>
<DialogButton OnClick="@CancelClick">
<DialogButtonModel Content="Cancel"></DialogButtonModel>
</DialogButton>
</DialogButtons>
</SfDialog>
<SfGrid @ref="Grid" DataSource="@Orders" AllowPaging="true" Toolbar="@ (new
List<string>() { "Delete" })" Height="315">
<GridEvents OnActionBegin="OnActionBegin" RowSelected="RowSelectHandler"
TValue="Order"></GridEvents>
<GridEditSettings AllowDeleting="true"
Mode="EditMode.Normal"></GridEditSettings>
<GridColumn>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
IsPrimaryKey="true" TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
Format="d" TextAlign="TextAlign.Right" Width="130"
Type="ColumnType.Date"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.ShipCountry) HeaderText="Ship Country"
Width="150"></GridColumn>
</GridColumn>
</SfGrid>
@code{
SfGrid<Order> Grid;
SfDialog Dialog;
public List<Order> Orders { get; set; }
public object SelectedData;
public bool flag = true;
public void Closed()
{
flag = true;
}
public void OnActionBegin(ActionEventArgs<Order> Args)
{
if (Args.RequestType.ToString() == "Delete" && flag)
{
Args.Cancel = true; //cancel default delete action
Dialog.Show();
flag = false;
}
}
public void RowSelectHandler(RowSelectEventArgs<Order> Args)
{
SelectedData = Args.Data.OrderID;
}
private void OkClick()
{
Grid.DeleteRecord(); //delete the record programatically while klikcing OK
button

```

```

Dialog.Hide();
}
private void CancelClick()
{
Dialog.Hide();
}
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
OrderDate = DateTime.Now.AddDays(-x),
ShipCountry = (new string[] { "USA", "UK", "CHINA", "RUSSIA", "INDIA" })[new
Random().Next(5)]
}).ToList();
}
public class Order
{
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
public string ShipCountry { get; set; }
}
}

```

Create custom Grid component in Blazor DataGrid Component

You can create a custom Grid component by rendering the SfGrid as a new razor component. It helps to create your own custom component when you might want to create multiple grids with same configuration or with default configuration through out your application.

This is demonstrated in below example by create a custom Grid component called CustomGrid, where we have rendered SfGrid with some basic default properties such as GridPageSettings etc. which will be reflected in all the Grids rendered using CustomGrid component.

CustomGrid.razor

ASPX-CS

```

@using Syncfusion.Blazor.Grids
@typeparam TValue
@inherits SfGrid<TValue>
<SfGrid TValue="TValue" AllowSorting="AllowSorting"
AllowPaging="AllowPaging" @attributes="props">
@ChildContent
<GridPageSettings PageCount="PAGE_COUNT" PageSize="DEFAULT_PAGE_SIZE"
PageSizes="PageSizes"></GridPageSettings>
</SfGrid>

```

CustomGrid.razor.cs

CSHARP

```

using Microsoft.AspNetCore.Components;
using Syncfusion.Blazor.Grids;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
namespace SF_Grid_Inheritance.Shared
{
    public partial class CustomGrid<TValue> : SfGrid<TValue>
    {
        public const int PAGE_COUNT = 5;
        public const int DEFAULT_PAGE_SIZE = 10;
        public string[] PageSizes = new string[] { "10", "20", "50" };
        IReadOnlyDictionary<string, object> props { get; set; }
        public override Task SetParametersAsync(ParameterView parameters)
        {
            //assign the additional parameters
            props = parameters.ToDictionary();
            return base.SetParametersAsync(parameters);
        }
        protected async override Task OnParametersSetAsync()
        {
            AllowPaging = true;
            AllowSorting = true;
            await base.OnParametersSetAsync();
        }
    }
}

```

Index.razor

CSHARP

```

<CustomGrid DataSource="Orders" TValue="Order"></CustomGrid>
@code{
    public List<Order> Orders { get; set; }
    protected override void OnInitialized()
    {
        Orders = Enumerable.Range(1, 75).Select(x => new Order()
        {
            OrderID = 1000 + x,
            CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
        })[new Random().Next(5)],
            Freight = 2.1 * x,
            OrderDate = DateTime.Now.AddDays(-x),
        }).ToList();
    }
    public class Order
    {
        public int? OrderID { get; set; }
        public string CustomerID { get; set; }
        public DateTime? OrderDate { get; set; }
        public double? Freight { get; set; }
    }
}

```

Add a range of items into ObservableCollection in Blazor DataGrid

By default, you can use `Add` method to add a single item into `ObservableCollection`. To add a range of items you can call the `Add` method multiple times using `foreach` statement. For every single add action into `ObservableCollection`, Grid will be refreshed to display the `DataSource` changes. So calling `Add` repeatedly inside `foreach` might have performance impact in Grid.

So, to effectively add a range of items into `ObservableCollection` bind to Grid, you can extend `ObservableCollection<T>` class and define an `AddRange` method. You can use this `AddRange` method to add a range of items and handle the `OnCollectionChanged` call occur one time for the multiple add actions.

This is demonstrated in the below sample code,

ASPX-CS

```
@using Syncfusion.Blazor.Grids
@using Syncfusion.Blazor.Buttons
@using System.Collections.ObjectModel
@using System.Collections.Specialized
<SfButton OnClick="AddRangeItems">Add Range of Items</SfButton>
<SfGrid @ref="Grid" DataSource="@GridData" AllowPaging="true">
  <GridColumns>
    <GridColumn Field=@nameof(OrdersDetailsObserveData.OrderID)
    HeaderText="Order ID" TextAlign="TextAlign.Right" Width="120"></GridColumn>
    <GridColumn Field=@nameof(OrdersDetailsObserveData.CustomerID)
    HeaderText="Customer Name" Width="150"></GridColumn>
    <GridColumn Field=@nameof(OrdersDetailsObserveData.OrderDate) HeaderText="
    Order Date" Format="d" Type="ColumnType.Date" TextAlign="TextAlign.Right"
    Width="130"></GridColumn>
    <GridColumn Field=@nameof(OrdersDetailsObserveData.Freight)
    HeaderText="Freight" Format="C2" TextAlign="TextAlign.Right"
    Width="120"></GridColumn>
  </GridColumns>
</SfGrid>
@code{
  SfGrid<OrdersDetailsObserveData> Grid;
  public SmartObservableCollection<OrdersDetailsObserveData> GridData = new
  SmartObservableCollection<OrdersDetailsObserveData>();
  public void AddRangeItems()
  {
    GridData.AddRange(Orders);
  }
  public class SmartObservableCollection<T> : ObservableCollection<T>
  {
    private bool _preventNotification = false;
    protected override void OnCollectionChanged(NotifyCollectionChangedEventArgs e)
    {
      {
        if (!_preventNotification)
          base.OnCollectionChanged(e);
      }
    }
    public void AddRange(IEnumerable<T> list)
    {
      _preventNotification = true;
      foreach (T item in list)
        Add(item);
    }
  }
}
```

```

_preventNotification = false;
OnCollectionChanged(new
NotifyCollectionChangedEventArgs(NotifyCollectionChangedAction.Reset));
}
}
IEnumerable<OrdersDetailsObserveData> Orders = Enumerable.Range(1,
10000).Select(x => new OrdersDetailsObserveData()
{
    OrderID = 1000 + x,
    CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
    Freight = 2.1 * x,
    OrderDate = DateTime.Now.AddDays(-x),
}).ToList();
public class OrdersDetailsObserveData
{
    public int? OrderID { get; set; }
    public string CustomerID { get; set; }
    public DateTime? OrderDate { get; set; }
    public double? Freight { get; set; }
}
}

```

Hide the command column button in a specific record

Command Columns can be used perform CRUD operation in Grid records. For some specific records, these editing actions can be prevented by hiding the command buttons. (i.e.) Some records can be prevented from edited, some records can be prevented from deleted. This can be achieved by hiding the command buttons in [RowDataBound](#) event of the DataGrid component.

This is demonstrated in the below sample code where the RowDataBound event is triggered when record is created. Based on record details, we can add specific class name to that row and hide the command buttons using CSS styles.

ASPX-CS

```

@using Syncfusion.Blazor.Grids
<SfGrid DataSource="@Orders" AllowPaging="true" Height="315">
<GridEvents RowDataBound="RowBound" TValue="Order"></GridEvents>
<GridEditSettings AllowAdding="true" AllowEditing="true"
AllowDeleting="true"></GridEditSettings>
<GridColumn>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="Order ID"
IsPrimaryKey="true" TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.CustomerID) HeaderText="Customer Name"
Width="120"></GridColumn>
<GridColumn Field=@nameof(Order.OrderDate) HeaderText=" Order Date"
EditType="EditType.DatePickerEdit" Format="d" TextAlign="TextAlign.Right"
Width="130"></GridColumn>
<GridColumn Field=@nameof(Order.Freight) HeaderText="Freight" Format="C2"
TextAlign="TextAlign.Right" Width="120"></GridColumn>
<GridColumn HeaderText="Manage Records" Width="150">
<GridCommandColumns>
<GridCommandColumn Type="CommandButtonType.Edit" ButtonOption="@ (new
CommandButtonOptions() { IconCss = "e-icons e-edit", CssClass = "e-flat"
})"></GridCommandColumn>

```

```

<GridCommandColumn Type="CommandButtonType.Delete" ButtonOption="@ (new
CommandButtonOptions() { IconCss = "e-icons e-delete", CssClass = "e-flat"
}) "></GridCommandColumn>
<GridCommandColumn Type="CommandButtonType.Save" ButtonOption="@ (new
CommandButtonOptions() { IconCss = "e-icons e-update", CssClass = "e-flat"
}) "></GridCommandColumn>
<GridCommandColumn Type="CommandButtonType.Cancel" ButtonOption="@ (new
CommandButtonOptions() { IconCss = "e-icons e-cancel-icon", CssClass = "e-
flat" }) "></GridCommandColumn>
</GridCommandColumns>
</GridColumn>
</GridColumns>
</SfGrid>
<style>
/*to remove the edit button alone*/
.e-removeEditcommand .e-unboundcell .e-unboundcelldiv button.e-Editbutton {
display: none;
}
/*to remove the delete button alone*/
.e-removeDeletecommand .e-unboundcell .e-unboundcelldiv button.e-
Deletebutton {
display: none;
}
</style>
@code{
public List<Order> Orders { get; set; }
public void RowBound(RowDataBoundEventArgs<Order> Args)
{
if (Args.Data.Verified)
{
Args.Row.AddClass(new string[] { "e-removeEditcommand" });
}
else
{
Args.Row.AddClass(new string[] { "e-removeDeletecommand" });
}
}
protected override void OnInitialized()
{
Orders = Enumerable.Range(1, 75).Select(x => new Order()
{
OrderID = 1000 + x,
CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID"
})[new Random().Next(5)],
Freight = 2.1 * x,
OrderDate = DateTime.Now.AddDays(-x),
Verified = (new bool[] { true, false })[new Random().Next(2)],
ShipCountry = (new string[] { "USA", "UK", "CHINA", "RUSSIA", "INDIA" })[new
Random().Next(5)]
}).ToList();
}
public class Order
{
public int? OrderID { get; set; }
public string CustomerID { get; set; }
public DateTime? OrderDate { get; set; }
public double? Freight { get; set; }
}
}

```



```
public bool Verified { get; set; }
public string ShipCountry { get; set; }
}
}
```

Hide expand icon in hierarchical Grid when it has no child records

Hierarchical structure in Grid can be achieved using Detail Template feature of Grid. But in some scenario, child grid may not have records. So expanding it will show empty Grid. So we can hide the expand icon when child grid has no records.

We can achieve this requirement using [RowDataBound](#) event of the DataGrid component. This event will be triggered when a row is created in Grid.

In this event, we have checked whether the Child grid has any records using the current record details and child grid datasource. Based on the results, we have added specific class name to row using AddClass method. We have traversed that specific element to hide the icon and pointer events.

This is demonstrated in the below sample code where expand icon is hidden for some specific records.

ASPX-CS

```
@using Syncfusion.Blazor.Grids
@using Syncfusion.Blazor.Data
<SfGrid DataSource="@Employees" Height="315px">
<GridEvents RowDataBound="RowDataBound" TValue="EmployeeData"></GridEvents>
<GridTemplates>
<DetailTemplate>
@{
var employee = (context as EmployeeData);
<SfGrid DataSource="@Orders" Query="@ (new Query().Where("EmployeeID",
"equal", employee.EmployeeID)) ">
<GridColumn>
<GridColumn Field=@nameof(Order.OrderID) HeaderText="First Name"
Width="110"> </GridColumn>
<GridColumn Field=@nameof(Order.CustomerName) HeaderText="Last Name"
Width="110"></GridColumn>
<GridColumn Field=@nameof(Order.ShipCountry) HeaderText="Title"
Width="110"></GridColumn>
</GridColumn>
</SfGrid>
}
</DetailTemplate>
</GridTemplates>
<GridColumn>
<GridColumn Field=@nameof(EmployeeData.FirstName) HeaderText="First Name"
Width="110"> </GridColumn>
<GridColumn Field=@nameof(EmployeeData.LastName) HeaderText="Last Name"
Width="110"></GridColumn>
<GridColumn Field=@nameof(EmployeeData.Title) HeaderText="Title"
Width="110"></GridColumn>
<GridColumn Field=@nameof(EmployeeData.Country) HeaderText="Country"
Width="110"></GridColumn>
</GridColumn>
</SfGrid>
<style>
/*to disable the mouse actions*/
```

```

.e-detail-disable .e-detailrowcollapse {
pointer-events: none;
}
/*if required hide the icons*/
.e-detail-disable .e-detailrowcollapse .e-icon-grightarrow {
visibility: hidden;
}
</style>
@code{
public void RowDataBound(RowDataBoundEventArgs<EmployeeData> Args) // will
be triggered when row is created
{
if (Orders.Where(x => x.EmployeeID == Args.Data.EmployeeID).ToList().Count
== 0) // check condition here whether the detail grid has records
{
Args.Row.AddClass(new string[] { "e-detail-disable" });
}
}
}
List<EmployeeData> Employees = new List<EmployeeData>
{
new EmployeeData() {EmployeeID = 1001, FirstName="Nancy", LastName="Fuller",
Title="Sales Representative", Country="USA"},
new EmployeeData() {EmployeeID = 1002, FirstName="Andrew",
LastName="Davolio", Title="Vice President", Country="UK"},
new EmployeeData() {EmployeeID = 1003, FirstName="Janet",
LastName="Leverling", Title="Sales", Country="UK"},
new EmployeeData() {EmployeeID = 1004, FirstName="Margaret",
LastName="Peacock", Title="Sales Manager", Country="UAE"},
new EmployeeData() {EmployeeID = 1005, FirstName="Steven",
LastName="Buchanan", Title="Inside Sales Coordinator", Country="USA"},
new EmployeeData() {EmployeeID = 1006, FirstName="Smith", LastName="Steven",
Title="HR Manager", Country="UAE"},
new EmployeeData() {EmployeeID = 1007, FirstName="Nancy",
LastName="Davolio", Title="HR Manager", Country="UAE"},
new EmployeeData() {EmployeeID = 1008, FirstName="Steven", LastName="Smith",
Title="HR Manager", Country="UAE"},
new EmployeeData() {EmployeeID = 1009, FirstName="Fuller", LastName="Janet",
Title="HR Manager", Country="UAE"},
};
List<Order> Orders = new List<Order> {
new Order() {EmployeeID = 1001, OrderID=001, CustomerName="Nancy",
ShipCountry="USA"},
new Order() {EmployeeID = 1001, OrderID=002, CustomerName="Steven",
ShipCountry="UR"},
new Order() {EmployeeID = 1003, OrderID=005, CustomerName="Nancy",
ShipCountry="ITA"},
new Order() {EmployeeID = 1003, OrderID=006, CustomerName="Nancy",
ShipCountry="UK"},
new Order() {EmployeeID = 1003, OrderID=007, CustomerName="Steven",
ShipCountry="GER"},
new Order() {EmployeeID = 1005, OrderID=009, CustomerName="Fuller",
ShipCountry="USA"},
new Order() {EmployeeID = 1006, OrderID=010, CustomerName="Leverling",
ShipCountry="UAE"},
new Order() {EmployeeID = 1006, OrderID=011, CustomerName="Margaret",
ShipCountry="KEN"},
};

```

```
new Order() {EmployeeID = 1007, OrderID=012, CustomerName="Buchanan",
ShipCountry="GER"},
new Order() {EmployeeID = 1006, OrderID=014, CustomerName="Andrew",
ShipCountry="UAE"},
};
public class EmployeeData
{
public int? EmployeeID { get; set; }
public string FirstName { get; set; }
public string LastName { get; set; }
public string Title { get; set; }
public string Country { get; set; }
}
public class Order
{
public int? EmployeeID { get; set; }
public int? OrderID { get; set; }
public string CustomerName { get; set; }
public string ShipCountry { get; set; }
}
}
```

DatePicker

Getting Started with Blazor DatePicker Component

This section briefly explains about how to include a [Blazor DatePicker](#) Component in your Blazor Server-Side and Client-Side application. You can refer to our Getting Started with [Blazor Server-Side DatePicker](#) and [Blazor WebAssembly DatePicker](#) documentation pages for configuration specifications.

To get start quickly with Blazor DatePicker component, you can check on this video.

{% youtube

"youtube:https://www.youtube.com/watch?v=ZN0zSIU59nY"%}

Importing Syncfusion Blazor component in the application

- Install `Syncfusion.Blazor.Calendars` NuGet package to the application by using the `NuGet Package Manager`.
- You can add the client-side resources through [CDN](#) or from [NuGet](#) package in the **HEAD** element of the `~/wwwroot/index.html` page.

HTML

```
<head>
<link href="_content/Syncfusion.Blazor.Themes/bootstrap4.css"
rel="stylesheet" />
<!-- <link
href="https://cdn.syncfusion.com/blazor/{{version}}/styles/{{theme}}.css"
rel="stylesheet" /> -->
</head>
```

For Internet Explorer 11 kindly refer the polyfills. Refer the [documentation](#) for more information.

HTML

```
<head>
<link href="_content/Syncfusion.Blazor.Themes/bootstrap4.css"
rel="stylesheet" />
<script
src="https://github.com/Daddoon/Blazor.Polyfill/releases/download/3.0.1/blaz
or.polyfill.min.js"></script>
</head>
```

Adding component package to the application

Open `~/Imports.razor` file and import the `Syncfusion.Blazor.Calendars` package.

ASPX-CS

```
@using Syncfusion.Blazor.Calendars
```

Add `SyncfusionBlazor` service in `Program.cs`

Open the `Program.cs` file and add services required by Syncfusion components using `builder.Services.AddSyncfusionBlazor()` method.

CSHARP

```
using Syncfusion.Blazor;
namespace BlazorApplication
{
    public class Program
    {
        public static async Task Main(string[] args)
        {
            ....
            ....
            builder.Services.AddSyncfusionBlazor();
            await builder.Build().RunAsync();
        }
    }
}
```

To enable custom client side resource loading from CRG or CDN. You need to disable resource loading by `AddSyncfusionBlazor(true)` and load the scripts in the **HEAD** element of the `~/wwwroot/index.html` page.

HTML

```
<head>
<script src="https://cdn.syncfusion.com/blazor/{{ site.blazorversion
}}/syncfusion-blazor.min.js"></script>
</head>
```

Adding `DatePicker` component to the application

To initialize the `DatePicker` component add the below code to your `Index.razor` view page which is present under `~/Pages` folder.

The following code shows a basic DatePicker component.

ASPX-CS

```
<SfDatePicker TValue="DateTime?" Placeholder='Choose a Date'></SfDatePicker>
```

Run the application

After the successful compilation of your application, press **F5** to run the application.

The output will be as follows.



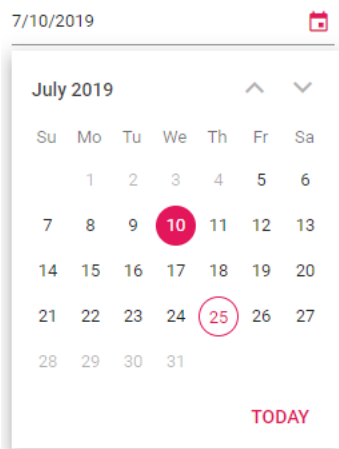
Setting the Value and Min and Max dates

The following example demonstrates how to set the [Value](#) and [Min](#) and [Max](#) dates on initializing the DatePicker. Here, you can select a date within the range from 5th to 27th of this month. [TValue](#) specifies the type of the DatePicker component.

ASPX-CS

```
<SfDatePicker TValue="DateTime?" Value='@DateValue' Min='@MinDate'
Max='@MaxDate'></SfDatePicker>
@code {
public DateTime? DateValue {get;set;} = new
DateTime(DateTime.Now.Year,DateTime.Now.Month,10);
public DateTime MinDate {get;set;} = new
DateTime(DateTime.Now.Year,DateTime.Now.Month,05);
public DateTime MaxDate {get;set;} = new DateTime(DateTime.Now.Year,
DateTime.Now.Month, 27);
}
```

The output will be as follows.



See Also

- [Getting Started with Syncfusion Blazor for Client-Side in .NET Core CLI](#)
- [Getting Started with Syncfusion Blazor for Server-side in Visual Studio 2019](#)

- [Getting Started with Syncfusion Blazor for Server-Side in .NET Core CLI](#)

Data Binding in Blazor DatePicker Component

This section briefly explains how to bind the value to the DatePicker component in the below different ways.

- One-Way Data Binding
- Two-Way Data Binding
- Dynamic Value Binding

One-Way Binding

We can bind the value to the DatePicker component directly for **Value** property as mentioned in the following code example. In one-way binding, we need to pass property or variable name along with **@** (For Ex: "@DateValue").

ASPX-CS

```
@using Syncfusion.Blazor.Calendars
<SfDatePicker TValue="DateTime?" Value="@DateValue"></SfDatePicker>
<button @onclick="@UpdateValue">Update Value</button>
@code {
    public DateTime? DateValue {get;set;} = new DateTime(DateTime.Now.Year,
    DateTime.Now.Month, 28);
    public void UpdateValue()
    {
        DateValue = DateTime.Now;
    }
}
```

Two-Way Data Binding

Two-way binding can be achieved by using **bind-Value** attribute and its supports string, int, Enum, DateTime, bool types. If component value has been changed, it will affect the all places where we bind the variable for the **bind-value** attribute.

ASPX-CS

```
@using Syncfusion.Blazor.Calendars
<p>DatePicker value is: @DateValue</p>
<SfDatePicker TValue="DateTime?" @bind-Value="@DateValue"></SfDatePicker>
@code {
    public DateTime? DateValue { get; set; } = DateTime.Now;
}
```

Dynamic Value Binding

We can change the property value dynamically by manually calling the **StateHasChanged()** method inside public event of **Blazor DatePicker component** only. This method notifies the component that its state has changed and queues a re-render.

There is no need to call this method for native events since it's called after any lifecycle method has been called and can also be invoked manually to trigger a re-render. Please refer the below mentioned code example.

ASPX-CS

```
@using Syncfusion.Blazor.Calendars
<p>DatePicker value is: @DateValue</p>
<SfDatePicker TValue="DateTime?" Value="@DateValue">
  <DatePickerEvents TValue="DateTime?"
  ValueChange="@onChange"></DatePickerEvents>
</SfDatePicker>
@code {
public DateTime? DateValue { get; set; } = DateTime.Now;
private void
onChange(Syncfusion.Blazor.Calendars.ChangedEventArgs<DateTime?> args)
{
DateValue = args.Value;
StateHasChanged();
}
}
```

Date Range in Blazor DatePicker Component

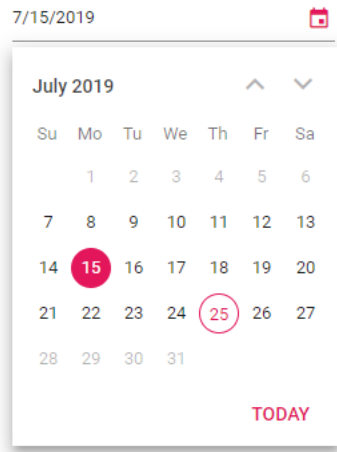
The DatePicker provides an option to select a date value within a specified range by using the [Min](#) and [Max](#) properties. Always the Min value has to be lesser than the Max value. The [Value](#) property depends on the Min/Max with respect to the [StrictMode](#) property. For more information about StrictMode, refer to the [Strict Mode](#) section from the documentation.

The following code allows selecting a date within the range from 7th to 27th in a month.

ASPX-CS

```
@using Syncfusion.Blazor.Calendars
<SfDatePicker TValue="DateTime?" Min='@MinDate' Max='@MaxDate'
Value='@DateValue'></SfDatePicker>
@code {
public DateTime MinDate {get;set;} = new
DateTime(DateTime.Now.Year,DateTime.Now.Month,07);
public DateTime MaxDate {get;set;} = new DateTime(DateTime.Now.Year,
DateTime.Now.Month, 27);
public DateTime? DateValue {get;set;} = new DateTime(DateTime.Now.Year,
DateTime.Now.Month, 15);
}
```

The output will be as follows.



When the Min and Max properties are configured and the selected date value is out-of-range or invalid, then the model value will be set to **out of range** date value or **null** respectively with highlighted **error** class to indicate the date is out of range or invalid.

ASPX-CS

```
@using Syncfusion.Blazor.Calendars
<SfDatePicker TValue="DateTime?" Min='@MinDate' Max='@MaxDate'
Value='@DateValue'></SfDatePicker>
@code {
public DateTime MinDate {get;set;} = new
DateTime(DateTime.Now.Year,DateTime.Now.Month,07);
public DateTime MaxDate {get;set;} = new DateTime(DateTime.Now.Year,
DateTime.Now.Month, 27);
public DateTime? DateValue {get;set;} = new DateTime(DateTime.Now.Year,
DateTime.Now.Month, 28);
}
```

The output will be as follows.



If the value of **Min** or **Max** properties changed through code behind, you have to update the **Value** property to set within the range.

Date Format in Blazor DatePicker Component

Date format is a way of representing the date value in different string formats in text box.

By default, the DatePicker's format is based on the culture. You can also set the own custom format by using the [Format](#) property.

Once the date format property has been defined, it will be common for all the cultures.

The following code demonstrates the DatePicker with the custom format (**yyyy-MM-dd**).

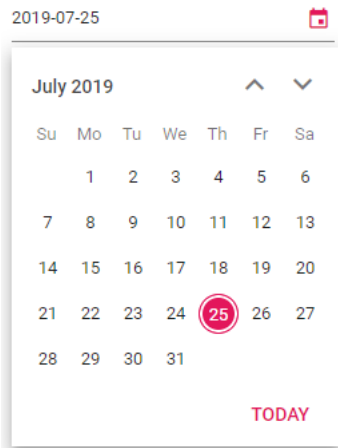
ASPX-CS

```
@using Syncfusion.Blazor.Calendars
<SfDatePicker TValue="DateTime?" Value='@DateValue' Format='yyyy-MM-
dd'></SfDatePicker>
```



```
@code {
    public DateTime? DateValue {get;set;} = DateTime.Now;
}
```

The output will be as follows.



Start and Depth View in Blazor DatePicker Component

The DatePicker has the following predefined views that provides a flexible way to navigate back and forth to select the date:

| **View** | **Description** |

| --- | --- |

| Month (default) | Displays the days in a month. |

| Year | Displays the months in a year. |

| Decade | Displays the years in a decade. |

Start view

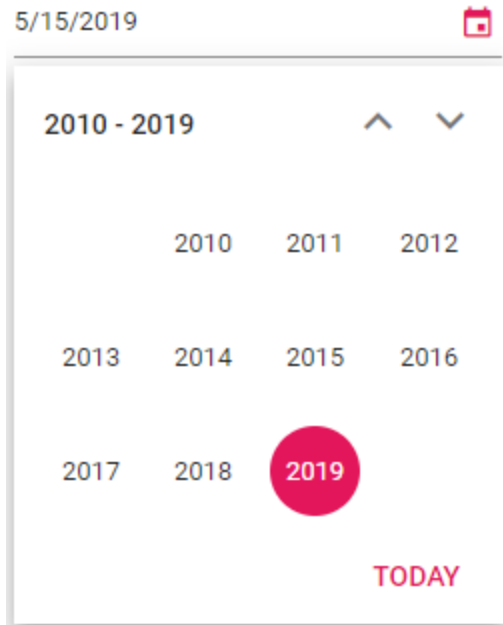
You can use the [Start](#) property to define the initial rendering view.

The following example demonstrates how to create a DatePicker with **Decade** as initial rendering view.

ASPX-CS

```
@using Syncfusion.Blazor.Calendar
<SfDatePicker TValue="DateTime?" Value="@DateValue" Placeholder="Select a date" Start='CalendarView.Decade'></SfDatePicker>
@code {
    public DateTime? DateValue {get;set;} = DateTime.Now;
}
```

The output will be as follows.



Depth view

Define the [Depth](#) property to control the view navigation.

Always the Depth view has to be smaller than the Start view, otherwise the view restriction will be not restricted.

The following example demonstrates how to create a DatePicker that allows users to select a month.

ASPX-CS

```
@using Syncfusion.Blazor.Calendar
<SfDatePicker TValue="DateTime?" Value="@DateValue" Placeholder='choose a date' Start='CalendarView.Decade' Depth='CalendarView.Year'></SfDatePicker>
@code {
    public DateTime? DateValue { get; set; } = DateTime.Now;
}
```

To learn more about Calendar views, refer to the Calendar's [Calendar Views](#) section.

Globalization in Blazor DatePicker Component

Globalization is the combination of adapting the control to various languages by means of parsing and formatting the date or number **Internationalization** and also by adding cultural specific customizations and translating the text **localization**.

Blazor server side

Add **UseRequestLocalization** middle-ware in Configure method in **Startup.cs** file to get browser Culture Info.

Refer the following code to add configuration in Startup.cs file

CSHARP

```
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Localization;
namespace BlazorApplication
```

```
{
public class Startup
{
    ....
    ....
    public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
    {
        app.UseRequestLocalization();
        ....
        ....
    }
}
}
```

The **Localization** library allows you to localize default text content. The DatePicker component has static text that can be changed to other cultures (Arabic, Deutsch, French, etc.).

In the following examples, demonstrate how to enable **Localization** for DatePicker in server side Blazor samples. Here, we have used Resource file to translate the static text.

The Resource file is an XML file which contains the strings(key and value pairs) that you want to translate into different language. You can also refer Localization [link](#) to know more about how to configure and use localization in the ASP.NET Core application framework.

- Open the **Startup.cs** file and add the below configuration in the **ConfigureServices** function as follows.

CSHARP

```
using Syncfusion.Blazor;
using System.Globalization;
using Microsoft.AspNetCore.Localization;
namespace BlazorApplication
{
    public class Startup
    {
        ....
        ....
        public void ConfigureServices(IServiceCollection services)
        {
            ....
            ....
            services.AddSyncfusionBlazor();
            services.AddLocalization(options => options.ResourcesPath = "Resources");
            services.Configure<RequestLocalizationOptions>(options =>
            {
                // define the list of cultures your app will support
                var supportedCultures = new List<CultureInfo>()
                {
                    new CultureInfo("de")
                };
                // set the default culture
                options.DefaultRequestCulture = new RequestCulture("de");
                options.SupportedCultures = supportedCultures;
                options.SupportedUICultures = supportedCultures;
            });
        }
    }
}
```

```
options.RequestCultureProviders = new List<IRequestCultureProvider>() {
    new QueryStringRequestCultureProvider() // Here, You can also use other
    localization provider
};
});
services.AddSingleton(typeof(ISyncfusionStringLocalizer),
    typeof(SampleLocalizer));
}
}
}
```

- Then, write a **class** by inheriting **ISyncfusionStringLocalizer** interface and override the Manager property to get the resource file details from the application end.

CSHARP

```
using Syncfusion.Blazor;
namespace blazorCalendars
{
    public class SampleLocalizer : ISyncfusionStringLocalizer
    {
        public string Get(string key)
        {
            return this.Manager.GetString(key);
        }
        public System.Resources.ResourceManager Manager
        {
            get
            {
                return blazorCalendars.Resources.SyncfusionBlazorLocale.ResourceManager;
            }
        }
    }
}
```

- Add **.resx** file to Resource folder and enter the key value (Locale Keywords) in the **Name** column and the translated string in the Value column as follows.

Name	Value (in Deutsch culture)
---	---
DatePicker_Today	Heute
DatePicker_Placeholder	Wählen Sie ein Datum

- Finally, Specify the culture for DatePicker using **locale** property.

ASPX-CS

```
@using Syncfusion.Blazor.Calendars
<SfDatePicker TValue="DateTime?" Locale="de"></SfDatePicker>
```

Blazor WebAssembly

By default, the DatePicker week and month names are specific to the **American English** culture. It utilizes the **Blazor Internationalization** package to parse and format the date object based on the culture by using the official [UNICODE CLDR](#) JSON data.

The following steps explain how to render the DatePicker in German culture ('de-DE') in Blazor Web Assembly application.

- Open the **program.cs** file and add the below configuration in the **Builder ConfigureServices** function as follows.

CSHARP

```
using Syncfusion.Blazor;
using Microsoft.AspNetCore.Builder;
namespace WebAssemblyLocale
{
    public class Program
    {
        public static async Task Main(string[] args)
        {
            ....
            ....
            builder.Services.Configure<RequestLocalizationOptions>(options =>
            {
                // Define the list of cultures your app will support
                var supportedCultures = new List<System.Globalization.CultureInfo>()
                {
                    new System.Globalization.CultureInfo("en-US"),
                    new System.Globalization.CultureInfo("de"),
                };
                // Set the default culture
                options.DefaultRequestCulture = new
                Microsoft.AspNetCore.Localization.RequestCulture("de");
                options.SupportedCultures = supportedCultures;
                options.SupportedUICultures = supportedCultures;
                options.RequestCultureProviders = new
                List<Microsoft.AspNetCore.Localization.IRequestCultureProvider>() {
                    new Microsoft.AspNetCore.Localization.QueryStringRequestCultureProvider()
                };
            });
            ....
            ....
        }
    }
}
```

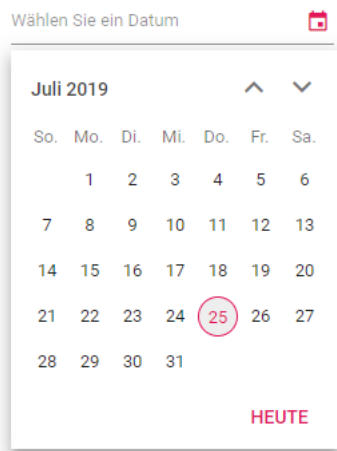
- Download the required locale packages to render the Blazor DatePicker component with specified locale.
- To download the locale definition of Blazor components, use this [link](#).
- After downloading the **blazor-locale** package, copy the **blazor-locale** folder with required local definition file into **wwwroot** folder.

- By default, the `blazor-locale` package contains the localized text for static text present in components like button text, placeholder, tooltip, and more.
- Set the culture by using the `SetCulture` method.

ASPX-CS

```
@using Syncfusion.Blazor.Calendars
@inject HttpClient Http;
<SfDatePicker TValue="DateTime?" Locale="de"></SfDatePicker>
@code {
    [Inject]
    protected IJSRuntime JsRuntime { get; set; }
    protected override async Task OnInitializedAsync()
    {
        this.JsRuntime.Sf().LoadLocaleData(await Http.GetJsonAsync<object>("blazor-locale/src/de.json")).SetCulture("de");
    }
}
```

The output will be as follows.



Customize the localized text

- You can change the localized text of particular component by editing the `wwwroot/blazor-locale/src/{{locale name}}.json` file.
- In the following code, modified the localized text of `today button` and `placeholder` in `de` culture.

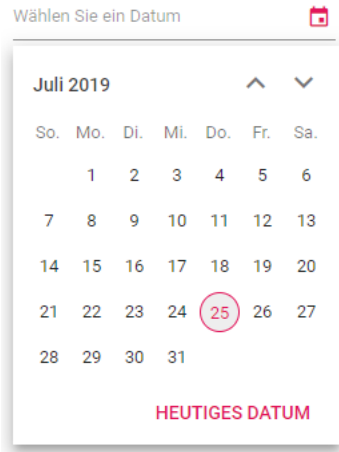
[`wwwroot/blazor-locale/src/de.json`]

CSHARP

```
{
  "de": {
    "datepicker": {
      "today": "Heutiges Datum",
      "placeholder": "Wählen Sie ein Datum"
    }
  }
}
```

```
}
}
```

The output will be as follows.



Right-To-Left

The DatePicker supports RTL (right-to-left) functionality for languages like Arabic and Hebrew to display the text in the right-to-left direction. Use the [EnableRtl](#) property to set the RTL direction.

The following code example initializes the DatePicker component in **Arabic** culture.

ASPX-CS

```
@using Syncfusion.Blazor.Calendars
@inject HttpClient Http;
<SfDatePicker TValue="DateTime?" EnableRtl=true Locale="ar"></SfDatePicker>
@code {
    [Inject]
    protected IJSRuntime JsRuntime { get; set; }
    protected override async Task OnInitializedAsync()
    {
        this.JsRuntime.Sf().LoadLocaleData(await Http.GetJsonAsync<object>("blazor-
        locale/src/ar.json")).SetCulture("ar");
    }
}
```

The output will be as follows.



Strict Mode in Blazor DatePicker Component

The [StrictMode](#) is an act that allows the users to enter only the valid date within the specified **Min/Max** range in text box. If the date is invalid, then the component will stay with the previous value. Else, if the date is out of range, then the component will set the date to the Min/Max date.

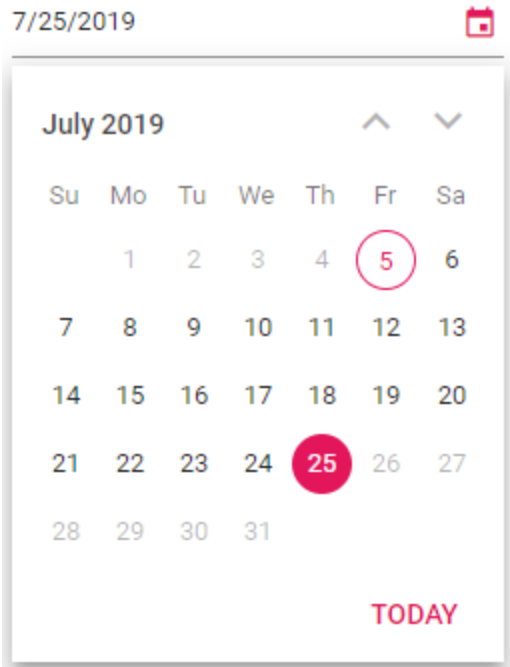
The following example demonstrates the DatePicker in **StrictMode** with Min/Max range of 5th to 25th in a month of May. Here, it allows the users to enter only the valid date within the specified range.

- If you are trying to enter the out-of-range value like 28th of May, then the Value will be set to the Max date of 25th May since the value 28th is greater than Max value of 25th.
- If you are trying to enter the invalid date, then the Value will stay with the previous value.

ASPX-CS

```
@using Syncfusion.Blazor.Calendars
<SfDatePicker TValue="DateTime?" Min="@MinDate" Max="@MaxDate"
StrictMode=true Value="@DateValue"></SfDatePicker>
@code {
    public DateTime MinDate {get;set;} = new
    DateTime(DateTime.Now.Year,DateTime.Now.Month,05);
    public DateTime MaxDate {get;set;} = new DateTime(DateTime.Now.Year,
    DateTime.Now.Month, 25);
    public DateTime? DateValue {get;set;} = new DateTime(DateTime.Now.Year,
    DateTime.Now.Month, 28);
}
```

The output will be as follows.



By default, the DatePicker act in **StrictMode** false state allows you to enter the invalid or out-of-range date in text box.

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

The following code demonstrates the **StrictMode** as false. Here, it allows you to enter the valid or invalid value in text box.

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

ASPX-CS

```
@using Syncfusion.Blazor.Calendars
<SfDatePicker TValue="DateTime?" Min="@MinDate" Max="@MaxDate"
StrictMode=false Value="@DateValue"></SfDatePicker>
@code {
public DateTime MinDate {get;set;} = new
DateTime(DateTime.Now.Year,DateTime.Now.Month,05);
public DateTime MaxDate {get;set;} = new DateTime(DateTime.Now.Year,
DateTime.Now.Month, 25);
public DateTime? DateValue {get;set;} = new DateTime(DateTime.Now.Year,
DateTime.Now.Month, 28);
}
```

The output will be as follows.



If the value of **Min** or **Max** properties changed through code behind, you have to update the **Value** property to set within the range.

Native Events in Blazor DatePicker Component

This section explains the steps to include native events and pass data to event handler in the DatePicker component.

Bind native events to DatePicker

You can access any native event by using on `<event>` attribute with a component. The attribute's value is treated as an event handler.

In the following example, The KeyPressed method is called every time the key is pressed on input.

ASPX-CS

```
@using Syncfusion.Blazor.Calendars
<SfDatePicker TValue="DateTime?" @onkeypress='@KeyPressed'></SfDatePicker>
@code {
    public void KeyPressed() {
        Console.WriteLine("Key Pressed!");
    }
}
```

Also, you can rewrite the previous code example as follows using the Lambda expressions.

ASPX-CS

```
@using Syncfusion.Blazor.Calendars
<SfDatePicker TValue="DateTime?" @onkeypress="@(() => Console.WriteLine("Key Pressed!"))"></SfDatePicker>
```

Pass event data to event handler

Blazor provides a set of argument types to map to native events. The list of event types and event arguments are:

- Focus Events - FocusEventArgs
- Mouse Events - MouseEventArgs
- Keyboard Events - KeyboardEventArgs
- Input Events - ChangeEventArgs/EventArgs
- Touch Events – TouchEventArgs
- Pointer Events – PointerEventArgs

In the following example, the KeyPressed method is called every time any key is pressed inside the input. But, the message will be printed when you press the "5" key.

ASPX-CS

```
@using Syncfusion.Blazor.Calendars
<SfDatePicker TValue="DateTime?" @onkeypress='@(e => KeyPressed(e))'></SfDatePicker>
@code {
    public void KeyPressed(KeyboardEventArgs args)
    {
        if (args.Key == "5")
        {
            Console.WriteLine("5 was pressed");
        }
    }
}
```

```
}
}
```

Using Lambda expression, you can pass the event data to the event handler.

List of Native events supported

| List of Native events | | |

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

| onclick | onblur | onfocus | onfocusout |

| onmousemove | onmouseover | onmouseout | onmousedown | onmouseup |

| ondblclick | onkeydown | onkeyup | onkeypress |

| ontouchend | onfocusin | onmouseup | ontouchstart |

Accessibility in Blazor DatePicker Component

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

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

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

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

- **aria-expanded:** Attribute indicates the state of a collapsible element.
- **aria-disabled:** Attribute indicates the disabled state of this DatePicker component.
- **aria-activedescendent:** Attribute helps in managing the current active child of the DatePicker component.

Keyboard interaction

You can use the following keys to interact with the DatePicker. The component implements the keyboard navigation support by following the [WAI-ARIA practices](#).

It supports the following list of shortcut keys:

Input navigation

Before opening the pop-up, use the following list of keys to control the pop-up element:

| **Keys** | **Description** |

| --- | --- |

| Alt + Down Arrow | Opens the popup. |

| Alt + Upper Arrow | Closes the popup. |

| Esc | Closes the popup. |

Calendar Navigation

Use the following list of keys to navigate the Calendar after the pop-up has been opened:

keys	Description
--- ---	
Upper Arrow	Focuses the previous week date.
Down Arrow	Focuses the next week date.
Left Arrow	Focuses the previous date.
Right Arrow	Focuses the next date.
Home	Focuses the first date in the month.
End	Focuses the last date in the month.
Page Up	Focuses the same date in the previous month.
Page Down	Focuses the same date in the next month.
Enter	Selects the currently focused date.
Shift + Page Up	Focuses the same date in the previous year.
Shift + Page Down	Focuses the same date in the previous year.
Control + Upper Arrow	Moves into the inner level of view like month-year and year-decade
Control + Down Arrow	Moves out from the depth level view like decade-year and year-month
Control +Home	Focuses the starting date in the current year.
Control +End	Focuses the ending date in the current year.

To focusout the DatePicker component, use the **t** keys. For additional information about native event, [click here](#).

ASPX-CS

```
@using Syncfusion.Blazor.Calendars
<SfDatePicker TValue="DateTime?" @onkeypress="@ (e => KeyPressed(e))"
@ref="DateObj"></SfDatePicker>
@code {
public SfDatePicker<DateTime?> DateObj;
public void KeyPressed(KeyboardEventArgs args)
{
if (args.Key == "t")
{
this.DateObj.FocusOutAsync();
}
}
}
```

Events in Blazor DatePicker Component

This section explains the list of events of the DatePicker component which will be triggered for appropriate DatePicker actions.

From v17.2. *added only the limited number of events for the DatePicker component. The event names are different from the previous releases and also removed several events. The following are the event name changes from v17.1. to v17.2.**

Event Name(v17.1.) / Event Name(v17.2.)

change | [ValueChange](#)

close | [OnClose](#)

open | [OnOpen](#)

renderDayCell | [OnRenderDayCell](#)

Blur

Blur event triggers when the input loses the focus.

ASPX-CS

```
@using Syncfusion.Blazor.Calendar
<SfDatePicker TValue="DateTime?">
  <DatePickerEvents TValue="DateTime?" Blur="BlurHandler"></DatePickerEvents>
</SfDatePicker>
@code{
public void BlurHandler(Syncfusion.Blazor.Calendar.BlurEventArgs args)
{
  // Here you can customize your code
}
}
```

ValueChange

ValueChange event triggers when the Calendar value is changed.

ASPX-CS

```
@using Syncfusion.Blazor.Calendar
<SfDatePicker TValue="DateTime?">
  <DatePickerEvents TValue="DateTime?"
  ValueChange="ValueChangeHandler"></DatePickerEvents>
</SfDatePicker>
@code{
public void ValueChangeHandler(ChangedEventArgs<DateTime?> args)
{
  // Here you can customize your code
}
}
```

OnClose

OnClose event triggers when the popup is closed.

ASPX-CS

```
@using Syncfusion.Blazor.Calendars
<SfDatePicker TValue="DateTime?">
  <DatePickerEvents TValue="DateTime?"
    OnClose="OnCloseHandler"></DatePickerEvents>
</SfDatePicker>
@code{
public void OnCloseHandler(PopupObjectArgs args)
{
    // Here you can customize your code
}
}
```

Created

Created event triggers when the component is created.

ASPX-CS

```
@using Syncfusion.Blazor.Calendars
<SfDatePicker TValue="DateTime?">
  <DatePickerEvents TValue="DateTime?"
    Created="CreatedHandler"></DatePickerEvents>
</SfDatePicker>
@code{
public void CreatedHandler(object args)
{
    // Here you can customize your code
}
}
```

Destroyed

Destroyed event triggers when the component is destroyed.

ASPX-CS

```
@using Syncfusion.Blazor.Calendars
<SfDatePicker TValue="DateTime?">
  <DatePickerEvents TValue="DateTime?"
    Destroyed="DestroyHandler"></DatePickerEvents>
</SfDatePicker>
@code{
public void DestroyHandler(object args)
{
    // Here you can customize your code
}
}
```

Focus

Focus event triggers when the input gets focus.

ASPX-CS

```
@using Syncfusion.Blazor.Calendars
<SfDatePicker TValue="DateTime?">
  <DatePickerEvents TValue="DateTime?"
    Focus="FocusHandler"></DatePickerEvents>
```

```
</SfDatePicker>
@code{
public void FocusHandler(FocusEventArgs args)
{
// Here you can customize your code
}
}
```

Navigated

Navigated event triggers when the Calendar is navigated to another level or within the same level of view

ASPX-CS

```
@using Syncfusion.Blazor.Calendars
<SfDatePicker TValue="DateTime?">
<DatePickerEvents TValue="DateTime?"
Navigated="NavigateHandler"></DatePickerEvents>
</SfDatePicker>
@code{
public void NavigateHandler(NavigatedEventArgs args)
{
// Here you can customize your code
}
}
```

OnOpen

OnOpen event triggers when the popup is opened

ASPX-CS

```
@using Syncfusion.Blazor.Calendars
<SfDatePicker TValue="DateTime?">
<DatePickerEvents TValue="DateTime?"
OnOpen="OpenHandler"></DatePickerEvents>
</SfDatePicker>
@code{
public void OpenHandler(PopupObjectArgs args)
{
// Here you can customize your code
}
}
```

OnRenderDayCell

OnRenderDayCell event triggers when each day cell of the Calendar is rendered.

ASPX-CS

```
@using Syncfusion.Blazor.Calendars
<SfDatePicker TValue="DateTime?">
<DatePickerEvents TValue="DateTime?"
OnRenderDayCell="onRenderDayCellHandler"></DatePickerEvents>
</SfDatePicker>
@code{
public void onRenderDayCellHandler(RenderDayCellEventArgs args)
```

```
{
// Here you can customize your code
}
```

DatePicker will be limited with these events and new events will be added in future based on the user requests. If the event you are looking for is not in the list, then please request [here](#).

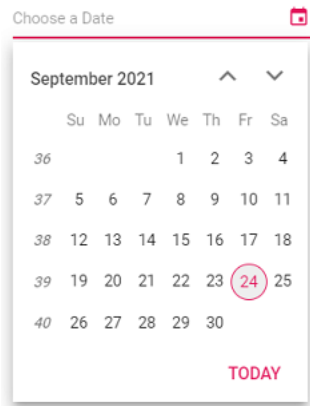
Week Number in Blazor DatePicker Component

You can enable WeekNumber in the DatePicker by using the [WeekNumber](#) property.

ASPX-CS

```
@using Syncfusion.Blazor.Calendars
<SfDatePicker TValue="DateTime?" Width="250px"
WeekNumber="true"></SfDatePicker>
```

The output will be as follows.



Week Rule

You can enable **WeekRule** in the DatePicker by using the [WeekRule](#) property. This property provide an option to specify the rule for defining the first week of the year. Please find the possible values of **WeekRule** property.

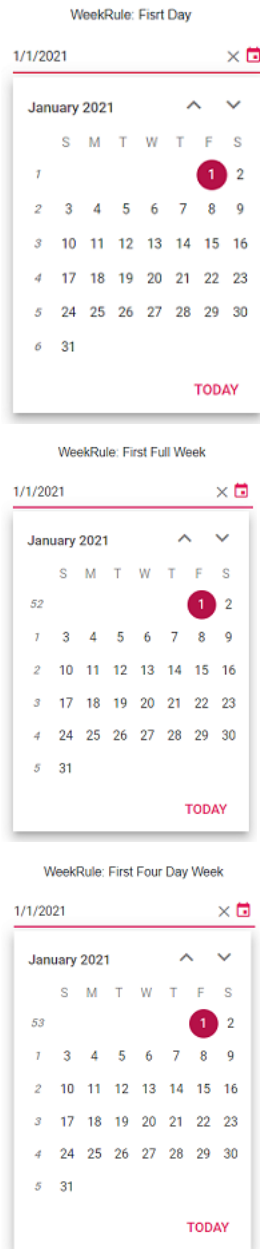
Types | Description

FirstDay | Set the first week of the year's week number to be started from 1. Then it followed as 1, 2, 3 ...

FirstFullWeek | Set the first week of the year's week number to be started from 52 or 53 (i.e December last week's week Number). Then it followed as 53, 1, 2 ...

FirstFourDayWeek | Set the week number based on the majority of dates present in the week for the respected months. If January dates are presented in the week more than December, the first week of the year's week number will be started from 1. If December dates are presented in the week more than January, the first week of the year's week number will be started from 52 or 53.

The output will be as follows.



Special Dates in Blazor DatePicker Component

You can customize specific dates in a DatePicker by using the [OnRenderDayCell](#) event. This event gets triggered on each day cell element creation that allows you to customize or disable the specific dates in the DatePicker. Here, list of dates in the current month are customized with custom styles by adding the personal-appointment and official-appointment class.

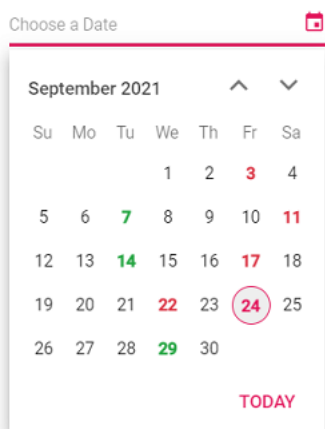
ASPX-CS

```
@using Syncfusion.Blazor.Calendars
<div class="control-wrapper">
<SfDatePicker TValue="DateTime?" Placeholder="Choose a Date"
ShowClearButton="true" @bind-Value="@SelectedDate">
<DatePickerEvents TValue="DateTime?" OnRenderDayCell="CustomDates"
ValueChange="OnChange"></DatePickerEvents>
```

```
</SfDatePicker>
</div>
<div id="display-date">
<span>Selected Day : @SelectedValue</span>
</div>
@code {
public DateTime? SelectedDate { get; set; }
public string SelectedValue { get; set; } =
DateTime.Now.ToString("M/d/yyyy");
public DateTime? CurrentDate { get; set; } = DateTime.Now;
public void CustomDates(RenderDayCellEventArgs args)
{
var CurrentMonth = CurrentDate.Value.Month;
if (args.Date.Month == CurrentMonth && (args.Date.Day == 7 || args.Date.Day
== 14 || args.Date.Day == 24 || args.Date.Day == 29)) {
args.CellData.ClassList += " personal-appointment";
}
if (args.Date.Month == CurrentMonth && (args.Date.Day == 3 || args.Date.Day
== 11 || args.Date.Day == 17 || args.Date.Day == 22))
{
args.CellData.ClassList += " official-appointment";
}
}
public void OnChange(ChangedEventArgs<DateTime?> args)
{
var Count = 0;
var CurrentMonth = CurrentDate.Value.Month;
if (args.Value.Value.Month == CurrentMonth && (args.Value.Value.Day == 7 ||
args.Value.Value.Day == 14 || args.Value.Value.Day == 24 ||
args.Value.Value.Day == 29))
{
this.SelectedValue = this.SelectedDate?.ToString("M/d/yyyy") + " (Personal
appointment)";
Count++;
}
if (args.Value.Value.Month == CurrentMonth && (args.Value.Value.Day == 3 ||
args.Value.Value.Day == 11 || args.Value.Value.Day == 17 ||
args.Value.Value.Day == 22))
{
this.SelectedValue = this.SelectedDate?.ToString("M/d/yyyy") + " (Official
appointment)";
Count++;
}
if (Count == 0)
{
this.SelectedValue = this.SelectedDate?.ToString("M/d/yyyy");
}
}
}
<style>
#display-date {
max-width: 270px;
padding: 15px 0;
font-size: 13px;
}
.control-wrapper {
width: 300px;
```

```
margin: 0 auto;
padding-top: 50px;
}
.e-calendar .e-content .e-cell.personal-appointment span.e-day,
.e-calendar .e-content td:hover.e-cell.personal-appointment span.e-day,
.e-calendar .e-content td.e-selected.e-focused-date.e-cell.personal-
appointment span.e-day {
color: #28a745;
font-weight: 800;
}
.e-calendar .e-content .e-cell.official-appointment span.e-day,
.e-calendar .e-content td:hover.e-cell.official-appointment span.e-day,
.e-calendar .e-content td.e-selected.e-focused-date.e-cell.official-
appointment span.e-day {
color: #dc3545;
font-weight: 800;
}
.e-calendar .e-content td.e-selected.e-focused-date.e-cell.personal-
appointment span.e-day,
.e-calendar .e-content td.e-selected.e-focused-date.e-cell.official-
appointment span.e-day {
background-color: #b511485e;
}
</style>
```

The output will be as follows.



How To

Disabled the Blazor DatePicker Component

To disable the DatePicker, use its [Enabled](#) property.

The following code demonstrates the DatePicker in disabled state.

ASPX-CS

```
@using Syncfusion.Blazor.Calendars
<SfDatePicker TValue="DateTime?" Enabled=false
Value="@DateValue"></SfDatePicker>
@code {
public DateTime? DateValue {get;set;} = DateTime.Now;
}
```

The output will be as follows.



Set the Placeholder in Blazor DatePicker Component

The following example demonstrates how to set **Placeholder** in the DatePicker component.

Using [Placeholder](#), you can display a short hint in the input element.

ASPX-CS

```
@using Syncfusion.Blazor.Calendar  
<SfDatePicker TValue="DateTime?" Placeholder="Select a date"></SfDatePicker>
```

The output will be as follows.



Set the Readonly in Blazor DatePicker Component

Enabled

By default, the Enabled property is true, it specifies the input can be focused, editable, and allows you to select date from the popup. But when enabled property is false, the input is not focusable, non-editable and cannot open the popup.

AllowEdit

By default, the AllowEdit property is true, it allows the textbox input to be changed as well as the user can select the value from the popup and false state defines the input is not editable but allows to select the value from the popup.

Readonly

By default, the Readonly property is false, it allows the input to be editable, and also allows value selection from the popup, and the true state does not allow user input, nor does it open popup, but the input can be focused. If you want to use the property Readonly, then you must disable the AllowEdit API.

The following code demonstrates how to set **Readonly** in DatePicker component. You can achieve this by using the [Readonly](#) and [AllowEdit](#) property.

ASPX-CS

```
@using Syncfusion.Blazor.Calendar  
<SfDatePicker TValue="DateTime?" AllowEdit="false" Readonly=true  
Value="@DateValue"></SfDatePicker>  
@code {  
    public DateTime? DateValue { get; set; } = DateTime.Now;  
}
```

The output will be as follows.



Open the Blazor DatePicker popup on Focus

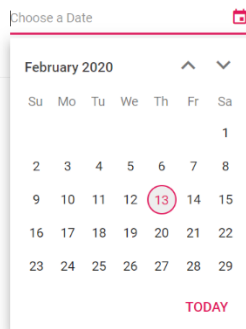
You can open the DatePicker popup on input focus by calling the `show` method in the input `focus` event.

The following example demonstrates how to open the DatePicker popup when the input is focused.

ASPX-CS

```
@using Syncfusion.Blazor.Calendars
<SfDatePicker TValue="DateTime?" @ref="@DateObj">
  <DatePickerEvents TValue="DateTime?"
    Focus="FocusHandler"></DatePickerEvents>
</SfDatePicker>
@code{
  SfDatePicker<DateTime?> DateObj;
  public void FocusHandler(Syncfusion.Blazor.Calendars.FocusEventArgs args)
  {
    this.DateObj.ShowAsync();
  }
}
```

The output will be as follows.



DateRangePicker

Getting Started with Blazor DateRangePicker Component

This section briefly explains about how to include a [Blazor DateRangePicker](#) Component in your Blazor Server-Side and Client-Side application. You can refer to our Getting Started with [Blazor Server-Side DateRangePicker](#) and [Blazor WebAssembly DateRangePicker](#) documentation pages for configuration specifications.

To get start quickly with Blazor DateRangePicker component, you can check on this video.

{% youtube

"youtube:https://www.youtube.com/watch?v=1xB_h1Zixl0"%}

Importing Syncfusion Blazor component in the application

- Install Syncfusion.Blazor.Calendars NuGet package to the application by using the NuGet Package Manager.
- You can add the client-side resources through [CDN](#) or from [NuGet](#) package in the **HEAD** element of the ~/wwwroot/index.html page.

HTML

```
<head>
<link href="_content/Syncfusion.Blazor.Themes/bootstrap4.css"
rel="stylesheet" />
<!-- <link
href="https://cdn.syncfusion.com/blazor/{{version}}/styles/{{theme}}.css"
rel="stylesheet" /> -->
</head>
```

For Internet Explorer 11 kindly refer the polyfills. Refer the [documentation](#) for more information.

HTML

```
<head>
<link href="_content/Syncfusion.Blazor.Themes/bootstrap4.css"
rel="stylesheet" />
<script
src="https://github.com/Daddoon/Blazor.Polyfill/releases/download/3.0.1/blaz
or.polyfill.min.js"></script>
</head>
```

Adding component package to the application

Open ~/_Imports.razor file and import the Syncfusion.Blazor.Calendars package.

ASPX-CS

```
@using Syncfusion.Blazor.Calendars
```

Add SyncfusionBlazor service in Program.cs

Open the **Program.cs** file and add services required by Syncfusion components using **services.AddSyncfusionBlazor()** method.

C#

```
using Syncfusion.Blazor;
namespace BlazorApplication
{
    public class Program
    {
        public static async Task Main(string[] args)
        {
            ....
            ....
            builder.Services.AddSyncfusionBlazor();
            await builder.Build().RunAsync();
        }
    }
}
```

```
}  
}
```

To enable custom client side resource loading from CRG or CDN. You need to disable resource loading by `AddSyncfusionBlazor(true)` and load the scripts in the **HEAD** element of the `~/wwwroot/index.html` page.

HTML

```
<head>  
<script src="https://cdn.syncfusion.com/blazor/{{ site.blazorversion  
}}/syncfusion-blazor.min.js"></script>  
</head>
```

Adding DateRangePicker component to the application

To initialize the DateRangePicker component add the below code to your `Index.razor` view page which is present under `~/Pages` folder.

The following code shows a basic DateRangePicker component.

ASPX-CS

```
<SfDateRangePicker TValue="DateTime?" Placeholder="Choose a  
Range"></SfDateRangePicker>
```

Run the application

After successful compilation of your application, press **F5** to run the application.

The output will be as follows.

Choose a Range 

Setting the Min and Max

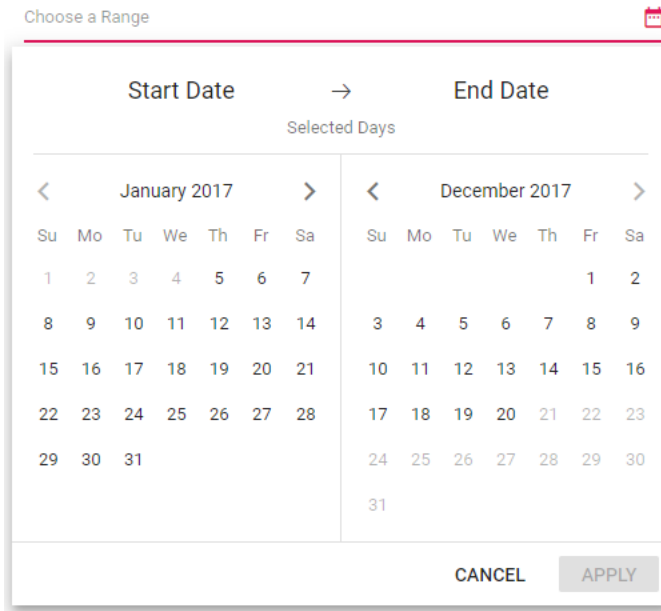
The minimum and maximum date range can be defined with the help of [Min](#) and [Max](#) properties.

The following code demonstrates how to set the `Min` and `Max` on initializing the DateRangePicker.

ASPX-CS

```
<SfDateRangePicker TValue="DateTime?" Placeholder="Choose a Range"  
Min="@MinDate" Max="@MaxDate"></SfDateRangePicker>  
@code {  
    public DateTime MinDate {get;set;} = new DateTime(2017, 01, 05);  
    public DateTime MaxDate {get;set;} = new DateTime(2017, 12, 20);  
}
```

The output will be as follows.



You can refer to our [Blazor Date Range Picker](#) feature tour page for its groundbreaking feature representations. You can also explore our [Blazor Date Range Picker example](#) to understand how to present and manipulate data.

See Also

- [Getting Started with Syncfusion Blazor for Client-Side in .NET Core CLI](#)
- [Getting Started with Syncfusion Blazor for Server-side in Visual Studio](#)
- [Getting Started with Syncfusion Blazor for Server-Side in .NET Core CLI](#)

Data Binding in Blazor DateRangePicker Component

This section briefly explains how to bind the value to the DateRangePicker component in the below different ways.

- One-Way Data Binding
- Two-Way Data Binding
- Dynamic Value Binding

One-Way Binding

We can bind the value to the DateRangePicker component directly for **StartDate** and **EndDate** property as mentioned in the following code example. In one-way binding, we need to pass property or variable name along with **@** (For Ex: "@StartValue").

ASPX-CS

```
@using Syncfusion.Blazor.Calendars
<SfDateRangePicker TValue="DateTime?" StartDate="@StartValue"
EndDate="@EndValue"></SfDateRangePicker>
<button @onclick="@UpdateValue">Update Value</button>
@code {
public DateTime? StartValue { get; set; } = new DateTime(DateTime.Now.Year,
DateTime.Now.Month, 28);
```



```

public DateTime? EndValue { get; set; } = new DateTime(DateTime.Now.Year,
DateTime.Now.Month + 1, 28);
public void UpdateValue()
{
    StartValue = new DateTime(DateTime.Now.Year + 1, DateTime.Now.Month, 28);
    EndValue = new DateTime(DateTime.Now.Year + 1, DateTime.Now.Month + 1, 28);
}
}

```

Two-Way Data Binding

Two-way binding can be achieved by using `bind-StartDate` and `bind-EndDate` attribute and its supports string, int, Enum, DateTime, bool types. If component value has been changed, it will affect the all places where we bind the variable for the `bind-StartDate` and `bind-EndDate` attribute.

ASPX-CS

```

@using Syncfusion.Blazor.Calendars
<p>DateRangePickers StartDate and EndDate is: <strong>@StartValue</strong>
and <strong>@EndValue</strong></p>
<SfDateRangePicker TValue="DateTime?" @bind-StartDate="@StartValue"
EndDate="@EndValue" ></SfDateRangePicker>
@code {
    public DateTime? StartValue { get; set; } = DateTime.Now;
    public DateTime? EndValue { get; set; } = DateTime.Now;
}

```

Dynamic Value Binding

We can change the property value dynamically by manually calling the `StateHasChanged()` method inside public event of **Blazor DateRangePicker component** only. This method notifies the component that its state has changed and queues a re-render.

There is no need to call this method for native events since it's called after any lifecycle method has been called and can also be invoked manually to trigger a re-render. Please refer the below mentioned code example.

ASPX-CS

```

@using Syncfusion.Blazor.Calendars
<p>DateRangePicker StartDate and EndDate is: <strong> @StartValue </strong>
and <strong> @EndValue </strong></p>
<SfDateRangePicker TValue="DateTime?" StartDate="@StartValue"
EndDate="@EndValue">
    <DateRangePickerEvents TValue="DateTime?"
    ValueChange="@onChange"></DateRangePickerEvents>
</SfDateRangePicker>
@code {
    public DateTime? StartValue { get; set; } = DateTime.Now;
    public DateTime? EndValue { get; set; } = DateTime.Now;
    private void onChange(RangepickerEventArgs<DateTime?> args)
    {
        StartValue = args.StartDate;
        EndValue = args.EndDate;
        StateHasChanged();
    }
}

```

You can refer to our [Blazor Date Range Picker](#) feature tour page for its groundbreaking feature representations. You can also explore our [Blazor Date Range Picker example](#) to understand how to present and manipulate data.

Range Restriction in Blazor DateRangePicker Component

Range selection in a DateRangePicker can be made-to-order with desired restrictions based on the application needs.

Restrict the range within a range

You can restrict the minimum and maximum date that can be allowed as Start and End date in a range selection with the help of [Min](#) and [Max](#) properties.

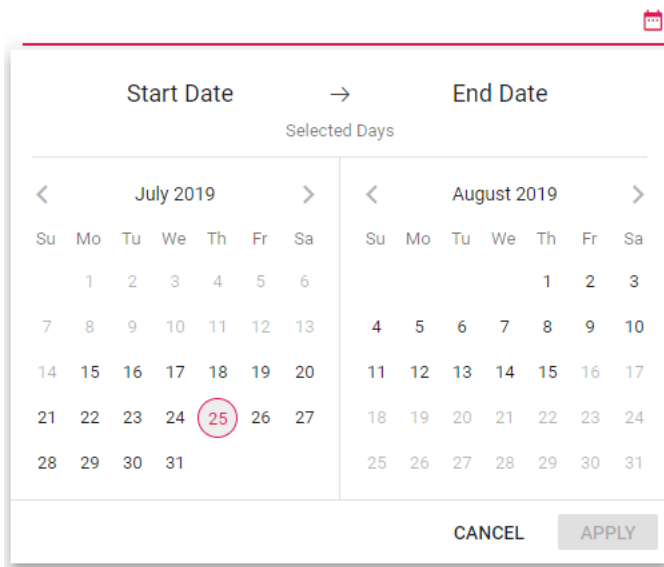
- **Min**: Sets the minimum date that can be selected as StartDate.
- **Max**: Sets the maximum date that can be selected as EndDate.

In the following sample, you can select a range from 15th day of this month to 15th day of next month.

ASPX-CS

```
@using Syncfusion.Blazor.Calendars
<SfDateRangePicker TValue="DateTime?" Min="@MinDate"
Max="@MaxDate"></SfDateRangePicker>
@code {
public DateTime MinDate {get;set;} = new
DateTime(DateTime.Now.Year,DateTime.Now.Month,15);
public DateTime MaxDate {get;set;} = new DateTime(DateTime.Now.Year,
DateTime.Now.Month+1, 15);
}
```

The output will be as follows.



Range span

Span between ranges can be limited to avoid excess or less days selection towards the required days in a range. In this, minimum and maximum span allowed within the date range can be customized by the [MinDays](#) and [MaxDays](#) properties.

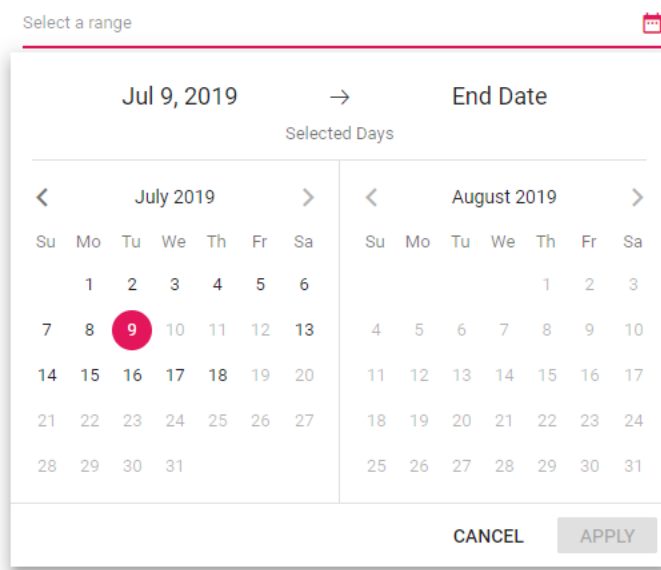
- **MinDays**: Sets the minimum number of days between Start and EndDate.
- **MaxDays**: Sets the maximum number of days between Start and EndDate.

In the following sample, the range selection should be greater than 5 days and less than 10 days, else it will not set.

ASPX-CS

```
@using Syncfusion.Blazor.Calendars
<SfDateRangePicker TValue="DateTime?" MinDays=5 MaxDays=10
Placeholder='Select a range'>
</SfDateRangePicker>
```

The output will be as follows.



Strict mode

DateRangePicker provides an option to limit the user towards entering the valid date. With **StrictMode**, you can set only the valid date. If any invalid range is specified, the date range value resets to previous value. This restriction can be availed by setting the [StrictMode](#) property to true.

ASPX-CS

```
@using Syncfusion.Blazor.Calendars
<SfDateRangePicker TValue="DateTime?" StartDate='@Start' EndDate='@End'
StrictMode=true Min='@MinDate' Max='@MaxDate'></SfDateRangePicker>
@code {
public DateTime MinDate {get;set;} = new
DateTime(DateTime.Now.Year,DateTime.Now.Month,15);
public DateTime MaxDate {get;set;} = new DateTime(DateTime.Now.Year,
DateTime.Now.Month+1, 15);
```

```
public DateTime? Start {get;set;} = new
DateTime(DateTime.Now.Year,DateTime.Now.Month,20);
public DateTime? End {get;set;} = new DateTime(DateTime.Now.Year,
DateTime.Now.Month+1, 25);
}
```

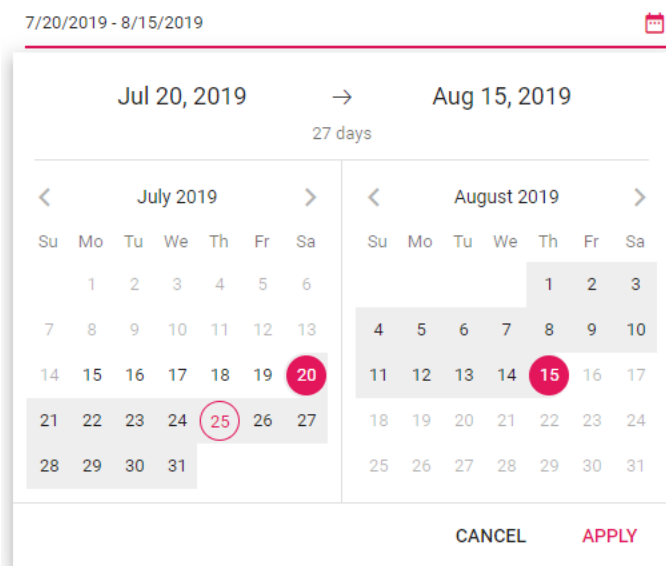
If the value of **Min** or **Max** property is changed through code behind, update the **StartDate** and **EndDate** properties to set within the range.

If the Start and End date is out of specified date range, a validation error class will be appended to the input element. If **StrictMode** is enabled and both the Start and End date is lesser than the Min date, then the Start and End date will be updated with Min date.

If both the Start and End date is higher than the Max date, then Start and End date will be updated with Max date.

If StartDate is less than Min date, it will be updated with Min date or if EndDate is greater than Max date, will be updated with the Max date.

The output will be as follows.



By default, the DatePicker acts in **StrictMode** false state that allows you to enter the invalid or out-of-range date in text box.

If the Start and End date is out of specified date range or invalid, then the model value will be set to **out of range** value or **null** respectively with highlighted **error** class to indicate the value is out of range or invalid.

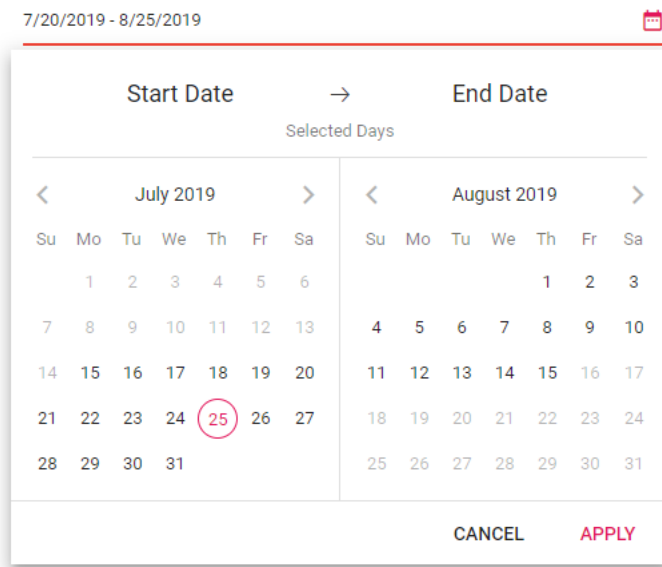
The following code demonstrates the **StrictMode** as false. Here, it allows you to enter the valid or invalid value in text box.

ASPX-CS

```
@using Syncfusion.Blazor.Calendars
<SfDateRangePicker TValue="DateTime?" StartDate="@Start" EndDate="@End"
StrictMode=false Min="@MinDate" Max="@MaxDate"></SfDateRangePicker>
```

```
@code {
    public DateTime MinDate {get;set;} = new
    DateTime(DateTime.Now.Year,DateTime.Now.Month,15);
    public DateTime MaxDate {get;set;} = new DateTime(DateTime.Now.Year,
    DateTime.Now.Month+1, 15);
    public DateTime? Start {get;set;} = new
    DateTime(DateTime.Now.Year,DateTime.Now.Month,20);
    public DateTime? End {get;set;} = new DateTime(DateTime.Now.Year,
    DateTime.Now.Month+1, 25);
}
```

The output will be as follows.



You can refer to our [Blazor Date Range Picker](#) feature tour page for its groundbreaking feature representations. You can also explore our [Blazor Date Range Picker example](#) to understand how to present and manipulate data.

Globalization in Blazor DateRangePicker Component

Globalization is the combination of adapting the control to various languages by means of parsing and formatting the date or number **Internationalization** and also by adding cultural specific customizations and translating the text **localization**.

Blazor server side

Add **UseRequestLocalization** middle-ware in Configure method in **Startup.cs** file to get browser Culture Info.

Refer the following code to add configuration in Startup.cs file

CSHARP

```
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Localization;
namespace BlazorApplication
{
    public class Startup
```

```
{
    ....
    ....
    public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
    {
        app.UseRequestLocalization();
        ....
        ....
    }
}
```

The **Localization** library allows you to localize default text content. The DateRangePicker component has static text that can be changed to other cultures (Arabic, Deutsch, French, etc.).

In the following examples, demonstrate how to enable **Localization** for DateRangePicker in server side Blazor samples. Here, we have used Resource file to translate the static text.

The Resource file is an XML file which contains the strings(key and value pairs) that you want to translate into different language. You can also refer Localization [link](#) to know more about how to configure and use localization in the ASP.NET Core application framework.

- Open the **Startup.cs** file and add the below configuration in the **ConfigureServices** function as follows.

CSHARP

```
using Syncfusion.Blazor;
using System.Globalization;
using Microsoft.AspNetCore.Localization;
namespace BlazorApplication
{
    public class Startup
    {
        ....
        ....
        public void ConfigureServices(IServiceCollection services)
        {
            ....
            ....
            services.AddSyncfusionBlazor();
            services.AddLocalization(options => options.ResourcesPath = "Resources");
            services.Configure<RequestLocalizationOptions>(options =>
            {
                // define the list of cultures your app will support
                var supportedCultures = new List<CultureInfo>()
                {
                    new CultureInfo("de")
                };
                // set the default culture
                options.DefaultRequestCulture = new RequestCulture("de");
                options.SupportedCultures = supportedCultures;
                options.SupportedUICultures = supportedCultures;
                options.RequestCultureProviders = new List<IRequestCultureProvider>() {
```

```
new QueryStringRequestCultureProvider() // Here, You can also use other
    localization provider
};
});
services.AddSingleton(typeof(ISyncfusionStringLocalizer),
    typeof(SampleLocalizer));
}
}
}
```

- Then, write a **class** by inheriting **ISyncfusionStringLocalizer** interface and override the **Manager** property to get the resource file details from the application end.

CSHARP

```
using Syncfusion.Blazor;
namespace blazorCalendars
{
    public class SampleLocalizer : ISyncfusionStringLocalizer
    {
        public string Get(string key)
        {
            return this.Manager.GetString(key);
        }
        public System.Resources.ResourceManager Manager
        {
            get
            {
                return blazorCalendars.Resources.SyncfusionBlazorLocale.ResourceManager;
            }
        }
    }
}
```

- Add **.resx** file to Resource folder and enter the key value (Locale Keywords) in the **Name** column and the translated string in the Value column as follows.

Name	Value (in Deutsch culture)
---	---
DateRangePicker_ApplyText	Anwenden
DateRangePicker_CancelText	Stornieren
DateRangePicker_CustomRange	Benutzerdefinierten Bereich
DateRangePicker_Days	Tage
DateRangePicker_EndLabel	Endtermin
DateRangePicker_Placeholder	Wählen Sie einen Datumsbereich
DateRangePicker_SelectedDays	Ausgewählte Tage

| DateRangePicker_StartLabel | Anfangsdatum |

- Finally, Specify the culture for DateRangePicker using `locale` property.

ASPX-CS

```
@using Syncfusion.Blazor.Calendars
<SfDateRangePicker TValue="DateTime?" Locale="de"></SfDateRangePicker>
```

Blazor WebAssembly

By default, the DateRangePicker week and month names are specific to the `American English` culture. It utilizes the `Blazor Internationalization` package to parse and format the date object based on the culture by using the official [UNICODE CLDR](#) JSON data.

The following steps explain how to render the DateRangePicker in German culture ('de-DE') in Blazor Web Assembly application.

- Open the `program.cs` file and add the below configuration in the `Builder ConfigureServices` function as follows.

C#

```
using Syncfusion.Blazor;
using Microsoft.AspNetCore.Builder;
namespace WebAssemblyLocale
{
    public class Program
    {
        public static async Task Main(string[] args)
        {
            ....
            ....
            builder.Services.Configure<RequestLocalizationOptions>(options =>
            {
                // Define the list of cultures your app will support
                var supportedCultures = new List<System.Globalization.CultureInfo>()
                {
                    new System.Globalization.CultureInfo("en-US"),
                    new System.Globalization.CultureInfo("de"),
                };
                // Set the default culture
                options.DefaultRequestCulture = new
                Microsoft.AspNetCore.Localization.RequestCulture("de");
                options.SupportedCultures = supportedCultures;
                options.SupportedUICultures = supportedCultures;
                options.RequestCultureProviders = new
                List<Microsoft.AspNetCore.Localization.IRequestCultureProvider>() {
                    new Microsoft.AspNetCore.Localization.QueryStringRequestCultureProvider()
                };
            });
            ....
        }
    }
}
```



```
}

```

- Download the required locale packages to render the Blazor DateRangePicker component with specified locale.
- To download the locale definition of Blazor components, use this [link](#).
- After downloading the `blazor-locale` package, copy the `blazor-locale` folder with required local definition file into `wwwroot` folder.
- By default, the `blazor-locale` package contains the localized text for static text present in components like button text, placeholder, tooltip, and more.
- Set the culture by using the `SetCulture` method.

ASPX-CS

```
@using Syncfusion.Blazor.Calendars
@inject HttpClient Http;
<SfDateRangePicker TValue="DateTime?" Locale="de"></SfDateRangePicker>
@code {
    [Inject]
    protected IJSRuntime JsRuntime { get; set; }
    protected override async Task OnInitializedAsync()
    {
        this.JsRuntime.Sf().LoadLocaleData(await Http.GetJsonAsync<object>("blazor-locale/src/de.json")).SetCulture("de");
    }
}
```

The output will be as follows.

Customize the localized text

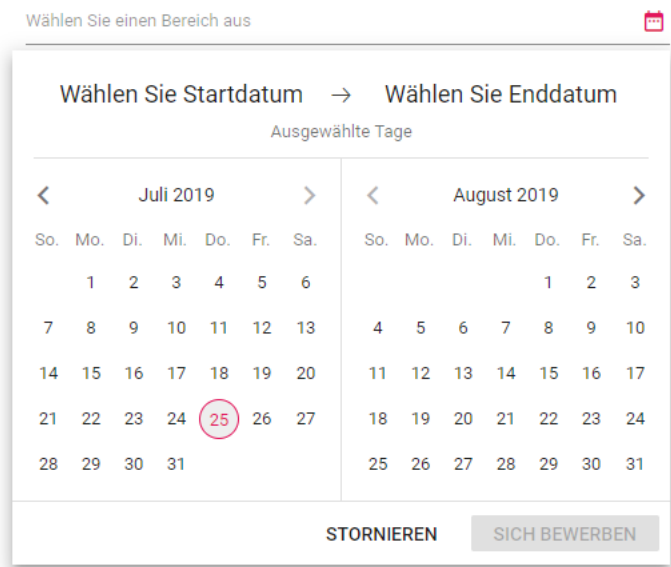
You can change the localized text of particular component by editing the `wwwroot/blazor-locale/src/{{locale name}}.json` file.

[`wwwroot/blazor-locale/src/de.json`]

CSHARP

```
{
  "de": {
    "daterangepicker": {
      "placeholder": "Wählen Sie einen Bereich aus",
      "startLabel": "Wählen Sie Startdatum",
      "endLabel": "Wählen Sie Enddatum",
      "applyText": "Sich bewerben",
      "cancelText": "Stornieren",
      "selectedDays": "Ausgewählte Tage",
      "days": "Tage",
      "customRange": "benutzerdefinierten Bereich"
    }
  }
}
```

The output will be as follows.

**Right-To-Left**

The DateRangePicker supports RTL (right-to-left) functionality for languages like Hebrew and Hebrew to displays the text in the right-to-left direction. Use [EnableRtl](#) property to set the RTL direction.

The following code example initialize the DateRangePicker component in **Hebrew** culture.

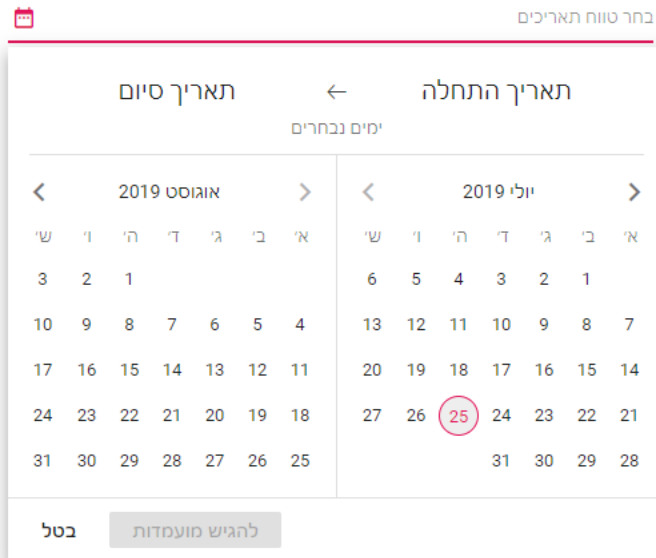
ASPX-CS

```
@using Syncfusion.Blazor.Calendars
@inject HttpClient Http;
<SfDateRangePicker TValue="DateTime?" Locale="ar"
EnableRtl=true></SfDateRangePicker>
@code {
    [Inject]
    protected IJSRuntime JsRuntime { get; set; }
    protected override async Task OnInitializedAsync()
    {

```

```
this.JsRuntime.Sf().LoadLocaleData(await Http.GetJsonAsync<object>("blazor-locale/src/ar.json")).SetCulture("ar");
}
```

The output will be as follows.



You can refer to our [Blazor Date Range Picker](#) feature tour page for its groundbreaking feature representations. You can also explore our [Blazor Date Range Picker example](#) to understand how to present and manipulate data.

Native Events in Blazor DateRangePicker Component

The following section explains steps to include native events and pass data to event handler in the DateRangePicker component.

Bind native events to DateRangePicker

You can access any native event by using on `<event>` attribute with a component. The attribute's value is treated as an event handler.

In the following example, the KeyPressed method is called every time the key is pressed on input.

ASPX-CS

```
@using Syncfusion.Blazor.Calendars
<SfDateRangePicker TValue="DateTime?"
@onkeypress='@KeyPressed'></SfDateRangePicker>
@code {
public void KeyPressed() {
Console.WriteLine("Key Pressed!");
}
}
```

Also, you can rewrite the above example code as follows using Lambda expressions.

ASPX-CS

```
@using Syncfusion.Blazor.Calendars
```

```
<SfDateRangePicker TValue="DateTime?" @onkeypress="@(() =>
Console.WriteLine("Key Pressed!"))"></SfDateRangePicker>
```

Pass event data to event handler

Blazor provides set of argument types to map to native events. the list of event types and event arguments are:

- Focus Events - FocusEventArgs
- Mouse Events - MouseEventArgs
- Keyboard Events - KeyboardEventArgs
- Input Events - ChangeEventArgs/EventArgs
- Touch Events – TouchEventArgs
- Pointer Events – PointerEventArgs

In the following example, the KeyPressed method is called every time any key is pressed inside input. But the message will be printed when you press "5" key.

ASPX-CS

```
@using Syncfusion.Blazor.Calendars
<SfDateRangePicker TValue="DateTime?" @onkeypress='@(e =>
KeyPressed(e))'></SfDateRangePicker>
@code {
public void KeyPressed(KeyboardEventArgs args)
{
if (args.Key == "5")
{
Console.WriteLine("5 was pressed");
}
}
}
```

Using Lambda expression also, you can pass the event data to the event handler.

List of Native events supported

| List of Native events | | |

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

| onclick | onblur | onfocus | onfocusout |

| onmousemove | onmouseover | onmouseout | onmousedown | onmouseup |

| ondblclick | onkeydown | onkeyup | onkeypress |

| ontouchend | onfocusin | onmouseup | ontouchstart |

You can refer to our [Blazor Date Range Picker](#) feature tour page for its groundbreaking feature representations. You can also explore our [Blazor Date Range Picker example](#) to understand how to present and manipulate data.

Customization in Blazor DateRangePicker Component

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

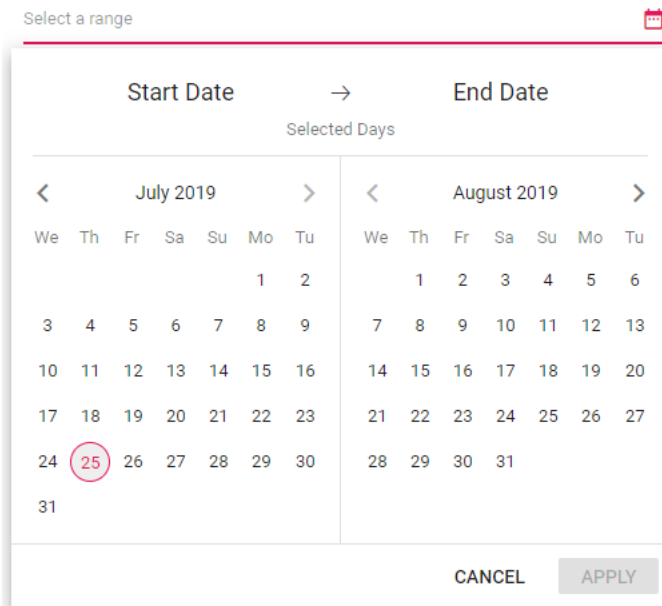
First day of week

Start day in a week will differ based on the culture, but you can also customize this based on the application needs. For this, use the [FirstDayOfWeek](#) property. By default, first day of a week in en-US is Sunday. In the following example, it is customized to Wednesday with the help of this property.

ASPX-CS

```
@using Syncfusion.Blazor.Calendars
<SfDateRangePicker TValue="DateTime?" Placeholder="Select a range"
FirstDayOfWeek=3></SfDateRangePicker>
```

The output will be as follows.



You can refer to our [Blazor Date Range Picker](#) feature tour page for its groundbreaking feature representations. You can also explore our [Blazor Date Range Picker example](#) to understand how to present and manipulate data.

Accessibility in Blazor DateRangePicker Component

The web accessibility makes web content and web applications more accessible for people with disabilities. It especially helps in dynamic content change and development of advanced user interface controls with AJAX, HTML, JavaScript, and related technologies. DateRangePicker provides built-in compliance with [WAI-ARIA](#) specifications. WAI-ARIA

support is achieved through the attributes like `aria-expanded`, `aria-disabled`, and `aria-activedescendant` applied as an input element.

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

It helps people with disabilities by providing information about the widget for assistive technology in the screen readers. DateRangePicker component contains grid role and grid cell for each day cell.

- **aria-expanded:** Indicates the currently selected date of the DateRangePicker component.
- **aria-disabled:** Indicates the disabled state of the DateRangePicker component.

Keyboard interaction

Use the following keys to interact with the DateRangePicker. This component implements the keyboard navigation support by following the [WAI-ARIA practices](#).

It supports the following list of shortcut keys:

Input navigation

Before opening the popup, use the following list of keys to control the popup element.

Keys	Explanation
--- ---	
Alt + Down Arrow	Opens the popup.
Alt + Upper Arrow	Closes the popup.
Esc	Closes the popup.

Calendar navigation

Use the following list of keys to navigate the currently focused Calendar after the popup has opened:

Keys	Explanation
--- ---	
Upper Arrow	Focuses the same day of the previous week.
Down Arrow	Focuses the same day of the next week.
Left Arrow	Focuses the day before.
Right Arrow	Focuses the next day.
Home	Focuses the first day of the month.
End	Focuses the last day of the month.
Page Up	Focuses the same date of the previous month.
Page Down	Focuses the same date of the next month.
Enter	Selects the currently focused date.
Shift + Page Up	Focuses the same date for the previous year.
Shift + Page Down	Focuses the same date for the next year.
Control + Home	Focuses the first date of the current year.
Control + End	Focuses the last date of the current year.
Alt + Right	Focuses through out the pop-up container in forward direction.
Alt + Left	Focuses through out the pop-up container in backward direction.

To focus out the DateRangePicker component use the `Tab` keys. For additional information about native event, [click](#) here.

ASPX-CS

```
@using Syncfusion.Blazor.Calendars
<SfDateRangePicker TValue="DateTime?" @onkeypress="@ (e => KeyPressed(e)) "
@ref="RangeObj"></SfDateRangePicker>
@code {
public SfDateRangePicker RangeObj;
public void KeyPressed(KeyboardEventArgs args)
{
if (args.Key == "t")
{
this.RangeObj.FocusOutAsync();
}
}
}
```

You can refer to our [Blazor Date Range Picker](#) feature tour page for its groundbreaking feature representations. You can also explore our [Blazor Date Range Picker example](#) to understand how to present and manipulate data.

Events in Blazor DateRangePicker Component

This section explains the list of events of the DateRangePicker component which will be triggered for appropriate DateRangePicker actions.

The events should be provided to the DateRangePicker using **DateRangePickerEvents** component.

From v17.2. *added only the limited number of events for the DateRangePicker component. The event names are different from the previous releases and also removed several events. The following are the event name changes from v17.1. to v17.2.**

Event Name(v17.1.) | Event Name(v17.2.)

change | [ValueChange](#)

close | [OnClose](#)

open | [OnOpen](#)

renderDayCell | [OnRenderDayCell](#)

select | [RangeSelected](#)

Blur

Blur event triggers when the input loses the focus.

ASPX-CS

```
@using Syncfusion.Blazor.Calendars
<SfDateRangePicker TValue="DateTime?">
<DateRangePickerEvents TValue="DateTime?"
Blur="BlurHandler"></DateRangePickerEvents>
</SfDateRangePicker>
@code{
public void BlurHandler(Syncfusion.Blazor.Calendars.BlurEventArgs args)
{
// Here you can customize your code
}
}
```

ValueChanged

ValueChanged event triggers when the Calendar value is changed.

ASPX-CS

```
@using Syncfusion.Blazor.Calendars
<SfDateRangePicker TValue="DateTime?">
  <DateRangePickerEvents TValue="DateTime?"
  ValueChange="ValueChangeHandler"></DateRangePickerEvents>
</SfDateRangePicker>
@code{
public void ValueChangeHandler(RangePickerEventArgs<DateTime?> args)
{
  // Here you can customize your code
}
```

OnClose

OnClose event triggers when the DateRangePicker is closed.

ASPX-CS

```
@using Syncfusion.Blazor.Calendars
<SfDateRangePicker TValue="DateTime?">
  <DateRangePickerEvents TValue="DateTime?"
  OnClose="OnCloseHandler"></DateRangePickerEvents>
</SfDateRangePicker>
@code{
public void OnCloseHandler(RangePopupEventArgs args)
{
  // Here you can customize your code
}
```

Created

Created event triggers when the component is created.

ASPX-CS

```
@using Syncfusion.Blazor.Calendars
<SfDateRangePicker TValue="DateTime?">
  <DateRangePickerEvents TValue="DateTime?"
  Created="CreatedHandler"></DateRangePickerEvents>
</SfDateRangePicker>
@code{
public void CreatedHandler(object args)
{
  // Here you can customize your code
}
```

Destroyed

Destroyed event triggers when the component is destroyed.

ASPX-CS

```
@using Syncfusion.Blazor.Calendar
<SfDateRangePicker TValue="DateTime?">
  <DateRangePickerEvents TValue="DateTime?"
    Destroyed="DestroyHandler"></DateRangePickerEvents>
</SfDateRangePicker>
@code{
public void DestroyHandler(object args)
{
  // Here you can customize your code
}
}
```

Focus

Focus event triggers when the input gets focus.

ASPX-CS

```
@using Syncfusion.Blazor.Calendar
<SfDateRangePicker TValue="DateTime?">
  <DateRangePickerEvents TValue="DateTime?"
    Focus="FocusHandler"></DateRangePickerEvents>
</SfDateRangePicker>
@code{
public void FocusHandler(Syncfusion.Blazor.Calendar.FocusEventArgs args)
{
  // Here you can customize your code
}
}
```

Navigated

Navigated event triggers when the Calendar is navigated to another level or within the same level of view

ASPX-CS

```
@using Syncfusion.Blazor.Calendar
<SfDateRangePicker TValue="DateTime?">
  <DateRangePickerEvents TValue="DateTime?"
    Navigated="NavigateHandler"></DateRangePickerEvents>
</SfDateRangePicker>
@code{
public void NavigateHandler(NavigatedEventArgs args)
{
  // Here you can customize your code
}
}
```

OnOpen

OnOpen event triggers when the DateRangePicker is opened.

ASPX-CS

```
@using Syncfusion.Blazor.Calendar
```

```
<SfDateRangePicker TValue="DateTime?">
  <DateRangePickerEvents TValue="DateTime?"
    OnOpen="OpenHandler"></DateRangePickerEvents>
</SfDateRangePicker>
@code{
public void OpenHandler(RangePopupEventArgs args)
{
  // Here you can customize your code
}
}
```

OnRenderDayCell

OnRenderDayCell event triggers when each day cell of the Calendar is rendered.

ASPX-CS

```
@using Syncfusion.Blazor.Calendars
<SfDateRangePicker TValue="DateTime?">
  <DateRangePickerEvents TValue="DateTime?"
    OnRenderDayCell="onRenderDayCellHandler"></DateRangePickerEvents>
</SfDateRangePicker>
@code{
public void onRenderDayCellHandler(RenderDayCellEventArgs args)
{
  // Here you can customize your code
}
}
```

RangeSelected

RangeSelected event triggers on selecting the start and end date.

ASPX-CS

```
@using Syncfusion.Blazor.Calendars
<SfDateRangePicker TValue="DateTime?">
  <DateRangePickerEvents TValue="DateTime?"
    RangeSelected="RangeSelectHandler"></DateRangePickerEvents>
</SfDateRangePicker>
@code{
public void RangeSelectHandler(RangePickerEventArgs<DateTime?> args)
{
  // Here you can customize your code
}
}
```

Datepicker will be limited with these events and new events will be added in future based on the user requests. If the event you are looking for is not in the list, then please request [here](#).

Week Number in Blazor DateRangePicker Component

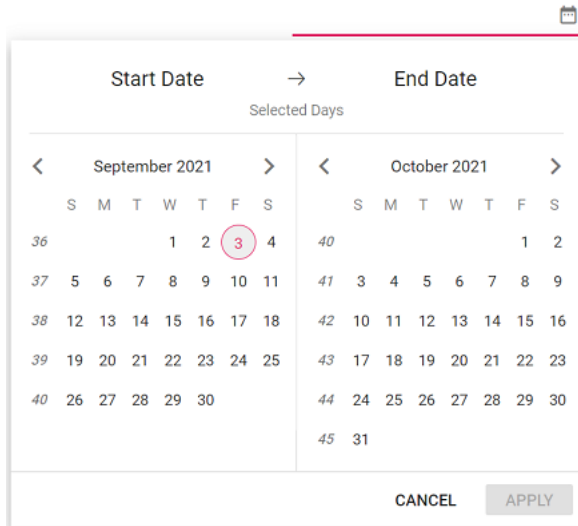
You can enable WeekNumber in the DateRangePicker by using the [WeekNumber](#) property.

ASPX-CS

```
@using Syncfusion.Blazor.Calendars
```

```
<SfDateRangePicker TValue="DateTime?" Width="250px"
WeekNumber="true"></SfDateRangePicker>
```

The output will be as follows.



Week Rule

You can enable **WeekRule** in the DateRangePicker by using the [WeekRule](#) property. This property provide an option to specify the rule for defining the first week of the year. Please find the possible values of **WeekRule** property.

Types | Description

FirstDay | Set the first week of the year's week number to be started from 1. Then it followed as 1, 2, 3 ...

FirstFullWeek | Set the first week of the year's week number to be started from 52 or 53 (i.e December last week's week Number). Then it followed as 53, 1, 2 ...

FirstFourDayWeek | Set the week number based on the majority of dates present in the week for the respected months. If January dates are presented in the week more than December, the first week of the year's week number will be started from 1. If December dates are presented in the week more than January, the first week of the year's week number will be started from 52 or 53.

The output will be as follows.

WeekRule: First Day

1/1/2021 - 1/10/2021

Jan 01, 2021 → Jan 10, 2021
10 Days

January 2021							February 2021						
S	M	T	W	T	F	S	S	M	T	W	T	F	S
1	2	3	4	5	6	7	1	2	3	4	5	6	
8	9	10	11	12	13	14	7	8	9	10	11	12	13
15	16	17	18	19	20	21	14	15	16	17	18	19	20
22	23	24	25	26	27	28	21	22	23	24	25	26	27
29	30	31					28						

CANCEL APPLY

WeekRule: First Full Week

1/1/2021 - 1/10/2021

Jan 01, 2021 → Jan 10, 2021
10 Days

January 2021							February 2021						
S	M	T	W	T	F	S	S	M	T	W	T	F	S
1	2	3	4	5	6	7	1	2	3	4	5	6	
8	9	10	11	12	13	14	7	8	9	10	11	12	13
15	16	17	18	19	20	21	14	15	16	17	18	19	20
22	23	24	25	26	27	28	21	22	23	24	25	26	27
29	30	31					28						

CANCEL APPLY

WeekRule: First Four Day Week

1/1/2021 - 1/10/2021

Jan 01, 2021 → Jan 10, 2021
10 Days

January 2021							February 2021						
S	M	T	W	T	F	S	S	M	T	W	T	F	S
1	2	3	4	5	6	7	1	2	3	4	5	6	
8	9	10	11	12	13	14	7	8	9	10	11	12	13
15	16	17	18	19	20	21	14	15	16	17	18	19	20
22	23	24	25	26	27	28	21	22	23	24	25	26	27
29	30	31					28						

CANCEL APPLY

How To

Disable the Blazor DateRangePicker Component

DateRangePicker can be inactivated on a page. By setting [Enabled](#) value to false will disable the component completely from all the user interactions including in the form post. The following code demonstrates the disabled component.

ASPX-CS

```
@using Syncfusion.Blazor.Calendars
<SfDateRangePicker TValue="DateTime?" Enabled=false StartDate="@Start"
EndDate="@End"></SfDateRangePicker>
@code {
public DateTime? Start { get; set; } = new DateTime(DateTime.Now.Year,
DateTime.Now.Month, 20);
public DateTime? End { get; set; } = new DateTime(DateTime.Now.Year,
DateTime.Now.Month + 1, 25);
}
```

The output will be as follows.



You can refer to our [Blazor Date Range Picker](#) feature tour page for its groundbreaking feature representations. You can also explore our [Blazor Date Range Picker example](#) to understand how to present and manipulate data.

Set the Placeholder in Blazor DateRangePicker Component

The following code demonstrates how to set [Placeholder](#) in the DateRangePicker component.

Using **Placeholder**, you can display a short hint in the input element.

ASPX-CS

```
@using Syncfusion.Blazor.Calendars
<SfDateRangePicker TValue="DateTime?" Placeholder="Choose a date
range"></SfDateRangePicker>
```

The output will be as follows.



You can refer to our [Blazor Date Range Picker](#) feature tour page for its groundbreaking feature representations. You can also explore our [Blazor Date Range Picker example](#) to understand how to present and manipulate data.

Customization using CssClass in Blazor DateRangePicker Component

To customize UI, you can make use of [CssClass](#) that will be added to the DateRangePicker component as the root CSS class. With this CSS class, you can override existing styles of DateRangePicker.

Following is the list of classes that provides flexible way to customize the DateRangePicker component:

Class Name	Description
---	---
e-date-range-wrapper	Applied to DateRangePicker wrapper.
e-range-icon	Applied to DateRangePicker icon.
e-popup	Applied to DateRangePicker popup wrapper.
e-calendar	Applied to both Calendar element.

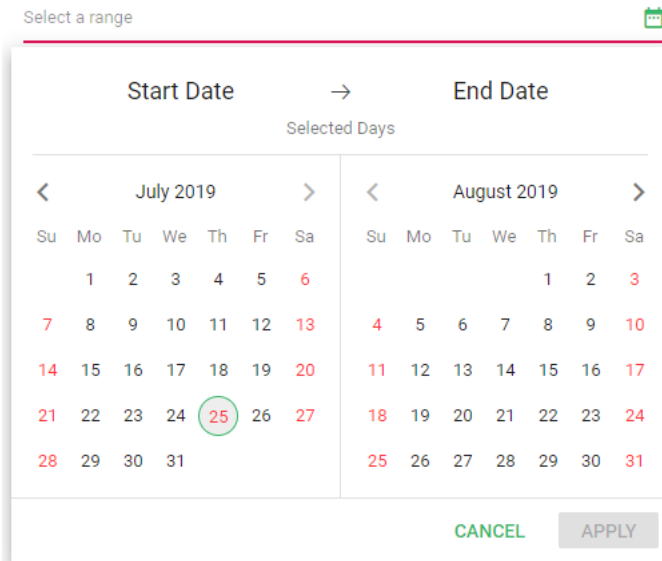
- | e-right-calendar | Applied to right Calendar element. |
- | e-left-calendar | Applied to left Calendar element. |
- | e-start-label | Applied to start label in a popup. |
- | e-end-calendar | Applied to end label in a popup. |
- | e-day-span | Applied to day span details label in a popup. |
- | e-footer | Applied to footer elements in a popup. |
- | e-apply | Applied to apply button in footer in a popup. |
- | e-cancel | Applied to cancel button in footer in a popup. |
- | e-header | Applied to Calendar header. |
- | e-title | Applied to Calendar title. |
- | e-icon-container | Applied to Calendar previous and next icon container. |
- | e-prev | Applied to Calendar previous icon. |
- | e-next | Applied to Calendar next icon. |
- | e-weekend | Applied to Calendar weekend dates. |
- | e-other-month | Applied to Calendar other month dates. |
- | e-day | Applied to each day cell of the Calendar. |
- | e-selected | Applied to Calendar selected dates. |
- | e-disabled | Applied to Calendar disabled dates. |

ASPX-CS

```
@using Syncfusion.Blazor.Calendars
<SfDateRangePicker TValue="DateTime?" Placeholder="Select a range"
  CssClass="CustomCSS" ></SfDateRangePicker>
<style>
.CustomCSS .e-calendar .e-content .e-selected span.e-day,
.CustomCSS .e-calendar .e-content .e-selected span.e-day:hover,
.CustomCSS .e-calendar .e-content .e-today.e-selected:hover span.e-day,
.CustomCSS .e-calendar .e-content .e-today.e-selected span.e-day,
.CustomCSS .e-calendar .e-content .e-selected:hover span.e-day {
background-color: #35b86b;
}
.CustomCSS .e-calendar .e-content .e-today span.e-day,
.CustomCSS .e-calendar .e-content .e-focused-date.e-today span.e-day {
border: 1px solid #35b86b;
color: #ff3337;
}
.CustomCSS .e-calendar .e-content .e-weekend span {
color: #ff3337;
}
.CustomCSS.e-date-range-wrapper .e-input-group-icon.e-icons.e-active,
.CustomCSS .e-btn.e-flat,
.CustomCSS .e-btn.e-flat:hover {
color: #35b86b;
}
```

```
</style>
```

The output will be as follows.



You can refer to our [Blazor Date Range Picker](#) feature tour page for its groundbreaking feature representations. You can also explore our [Blazor Date Range Picker example](#) to understand how to present and manipulate data.

Datetime Picker

Getting Started with Blazor DateTime Picker Component

This section briefly explains about how to include a [Blazor DateTimePicker](#) Component in your Blazor Server-Side and Client-Side application. You can refer to our Getting Started with [Blazor Server-Side DateTimePicker](#) and [Blazor WebAssembly DateTimePicker](#) documentation pages for configuration specifications.

To get start quickly with Blazor DateTimePicker component, you can check on this video.

{% youtube

"youtube:https://www.youtube.com/watch?v=BzcHdhd4o8I"%}

Importing Syncfusion Blazor component in the application

- Install `Syncfusion.Blazor.Calendars` NuGet package to the application by using the `NuGet` Package Manager.
- You can add the client-side resources through [CDN](#) or from [NuGet](#) package in the **HEAD** element of the `~/wwwroot/index.html` page.

HTML

```
<head>
<link href="_content/Syncfusion.Blazor.Themes/bootstrap4.css"
rel="stylesheet" />
```

```
<!-- <link  
href="https://cdn.syncfusion.com/blazor/{{version}}/styles/{{theme}}.css"  
rel="stylesheet" /> -->  
</head>
```

For Internet Explorer 11 kindly refer the polyfills. Refer the [documentation](#) for more information.

HTML

```
<head>  
<link href="_content/Syncfusion.Blazor.Themes/bootstrap4.css"  
rel="stylesheet" />  
<script  
src="https://github.com/Daddoon/Blazor.Polyfill/releases/download/3.0.1/blaz  
or.polyfill.min.js"></script>  
</head>
```

Adding component package to the application

Open `~/_Imports.razor` file and import the `Syncfusion.Blazor.Calendars` package.

ASPX-CS

```
@using Syncfusion.Blazor.Calendars
```

Add `SyncfusionBlazor` service in `Program.cs`

Open the `Program.cs` file and add services required by Syncfusion components using `builder.Services.AddSyncfusionBlazor()` method.

CSHARP

```
using Syncfusion.Blazor;  
namespace BlazorApplication  
{  
    public class Program  
    {  
        public static async Task Main(string[] args)  
        {  
            ....  
            ....  
            builder.Services.AddSyncfusionBlazor();  
            await builder.Build().RunAsync();  
        }  
    }  
}
```

To enable custom client side resource loading from CRG or CDN. You need to disable resource loading by `AddSyncfusionBlazor(true)` and load the scripts in the **HEAD** element of the `~/wwwroot/index.html` page.

HTML

```
<head>  
<script src="https://cdn.syncfusion.com/blazor/{{ site.blazorversion  
}}/syncfusion-blazor.min.js"></script>
```



```
</head>
```

Adding DateTimePicker component to the application

To initialize the DateTimePicker component add the below code to your `Index.razor` view page which is present under `~/Pages` folder.

The following code shows a basic DateTimePicker component.

ASPX-CS

```
<SfDateTimePicker TValue="DateTime?" Placeholder='Select a date and time'></SfDateTimePicker>
```

Run the application

After successful compilation of your application, press **F5** to run the application.

The output will be as follows.

Select a date and time  

Setting the Value, Min and Max

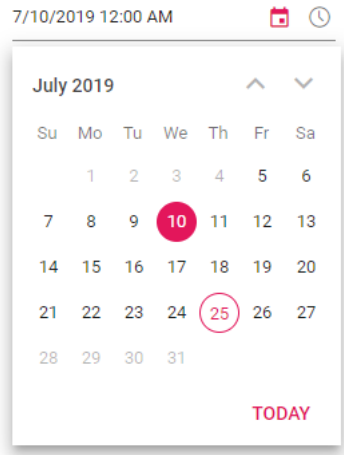
The minimum and maximum date time can be defined with the help of [Min](#) and [Max](#) properties.

The following code demonstrates how to set the `Min` and `Max` on initializing the DateTimePicker. `TValue` specifies the type of the DatePicker component.

ASPX-CS

```
<SfDateTimePicker TValue="DateTime?" Value="@DateValue" Min="@MinDate" Max="@MaxDate"></SfDateTimePicker>
@code{
public DateTime? DateValue {get;set;} = new
DateTime(DateTime.Now.Year,DateTime.Now.Month,10);
public DateTime MinDate {get;set;} = new
DateTime(DateTime.Now.Year,DateTime.Now.Month,05);
public DateTime MaxDate {get;set;} = new DateTime(DateTime.Now.Year,
DateTime.Now.Month, 27);
}
```

The output will be as follows.



See Also

- [Getting Started with Syncfusion Blazor for Client-Side in .NET Core CLI](#)
- [Getting Started with Syncfusion Blazor for Server-side in Visual Studio](#)
- [Getting Started with Syncfusion Blazor for Server-Side in .NET Core CLI](#)

Data Binding in Blazor Datetime Picker Component

This section briefly explains how to bind the value to the DateTimePicker component in the below different ways.

- One-Way Data Binding
- Two-Way Data Binding
- Dynamic Value Binding

One-Way Binding

We can bind the value to the DateTimePicker component directly for **Value** property as mentioned in the following code example. In one-way binding, we need to pass property or variable name along with **@** (For Ex: "@DateValue").

ASPX-CS

```
@using Syncfusion.Blazor.Calendars
<SfDateTimePicker TValue="DateTime?" Value="@DateValue"></SfDateTimePicker>
<button @onclick="@UpdateValue">Update Value</button>
@code {
    public DateTime? DateValue { get; set; } = new DateTime(DateTime.Now.Year,
    DateTime.Now.Month, 28);
    public void UpdateValue()
    {
        DateValue = DateTime.Now;
    }
}
```

Two-Way Data Binding

Two-way binding can be achieved by using `bind-Value` attribute and it supports string, int, Enum, DateTime, bool types. If component value has been changed, it will affect the all places where we bind the variable for the `bind-value` attribute.

ASPX-CS

```
@using Syncfusion.Blazor.Calendars
<p>DateTimePicker value is: @DateValue</p>
<SfDateTimePicker TValue="DateTime?" @bind-
Value="@DateValue"></SfDateTimePicker>
@code {
    public DateTime? DateValue { get; set; } = DateTime.Now;
}
```

Dynamic Value Binding

We can change the property value dynamically by manually calling the `StateHasChanged()` method inside public event of **Blazor DateTimePicker component** only. This method notifies the component that its state has changed and queues a re-render.

There is no need to call this method for native events since it's called after any lifecycle method has been called and can also be invoked manually to trigger a re-render. Please refer the below mentioned code example.

ASPX-CS

```
@using Syncfusion.Blazor.Calendars
<p>DateTimePicker value is: @DateValue</p>
<SfDateTimePicker TValue="DateTime?" Value="@DateValue">
    <DatePickerEvents TValue="DateTime?"
    ValueChange="@onChange"></DatePickerEvents>
</SfDateTimePicker>
@code {
    public DateTime? DateValue { get; set; } = DateTime.Now;
    private void
    onChange(Syncfusion.Blazor.Calendars.ChangedEventArgs<DateTime?> args)
    {
        DateValue = args.Value;
        StateHasChanged();
    }
}
```

DateTime Range in Blazor Datetime Picker Component

DateTimePicker provides an option to select a date and time value within a specified range by using the [Min](#) and [Max](#) properties. Always the Min value has to be lesser than the Max value.

The `Value` property depends on the Min/Max with respect to [StrictMode](#) property. For more information about StrictMode, refer to the [Strict Mode](#) section from the documentation.

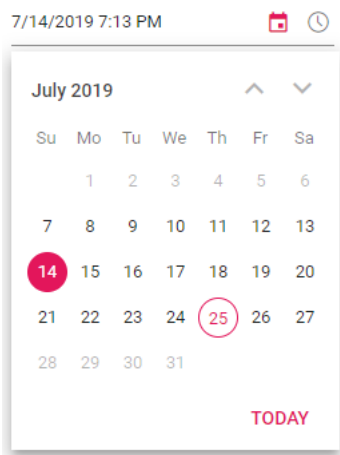
The following code allows selecting a date within the range from 7th to 27th day in a month.

ASPX-CS

```
@using Syncfusion.Blazor.Calendars
```

```
<SfDateTimePicker TValue="DateTime?" Min="@MinDateTime" Max="@MaxDateTime"
Value="@DateTimeValue"></SfDateTimePicker>
@code {
public DateTime MinDateTime {get;set;} = new
DateTime(DateTime.Now.Year,DateTime.Now.Month, 7, 0, 0, 0);
public DateTime MaxDateTime {get;set;} = new DateTime(DateTime.Now.Year,
DateTime.Now.Month, 27, DateTime.Now.Hour, DateTime.Now.Minute,
DateTime.Now.Second);
public DateTime? DateTimeValue {get;set;} = new DateTime(DateTime.Now.Year,
DateTime.Now.Month, 14, DateTime.Now.Hour, DateTime.Now.Minute,
DateTime.Now.Second);
}
```

The output will be as follows.

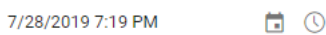


When the Min and Max properties are configured and the selected datetime value is out-of-range or invalid, then the model value will be set to **out of range datetime value** or **null** respectively with highlighted **error** class to indicates the datetime is out of range or invalid.

ASPX-CS

```
@using Syncfusion.Blazor.Calendars
<SfDateTimePicker TValue="DateTime?" Min="@MinDateTime" Max="@MaxDateTime"
Value="@DateTimeValue"></SfDateTimePicker>
@code {
public DateTime MinDateTime {get;set;} = new
DateTime(DateTime.Now.Year,DateTime.Now.Month, 7, 0, 0, 0);
public DateTime MaxDateTime {get;set;} = new DateTime(DateTime.Now.Year,
DateTime.Now.Month, 27, DateTime.Now.Hour, DateTime.Now.Minute,
DateTime.Now.Second);
public DateTime? DateTimeValue {get;set;} = new DateTime(DateTime.Now.Year,
DateTime.Now.Month, 28, DateTime.Now.Hour, DateTime.Now.Minute,
DateTime.Now.Second);
}
```

The output will be as follows.



If the value of **Min** or **Max** properties changed through code behind, you have to update the **Value** property to set within the range.

Globalization in Blazor Datetime Picker Component

Globalization is the combination of adapting the control to various languages by means of parsing and formatting the date or number **Internationalization** and also by adding cultural specific customizations and translating the text **localization**.

Blazor server side

Add **UseRequestLocalization** middle-ware in Configure method in **Startup.cs** file to get browser Culture Info.

Refer the following code to add configuration in Startup.cs file

CSHARP

```
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Localization;
namespace BlazorApplication
{
    public class Startup
    {
        ....
        ....
        public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
        {
            app.UseRequestLocalization();
            ....
            ....
        }
    }
}
```

The **Localization** library allows you to localize default text content. The **DateTimePicker** component has static text that can be changed to other cultures (Arabic, Deutsch, French, etc.).

In the following examples, demonstrate how to enable **Localization** for **DateTimePicker** in server side Blazor samples. Here, we have used Resource file to translate the static text.

The Resource file is an XML file which contains the strings(key and value pairs) that you want to translate into different language. You can also refer [Localization link](#) to know more about how to configure and use localization in the ASP.NET Core application framework.

- Open the **Startup.cs** file and add the below configuration in the **ConfigureServices** function as follows.

CSHARP

```
using Syncfusion.Blazor;
using System.Globalization;
using Microsoft.AspNetCore.Localization;
namespace BlazorApplication
{
    public class Startup
```

```

{
    ....
    ....
    public void ConfigureServices(IServiceCollection services)
    {
        ....
        ....
        services.AddSyncfusionBlazor();
        services.AddLocalization(options => options.ResourcesPath = "Resources");
        services.Configure<RequestLocalizationOptions>(options =>
        {
            // define the list of cultures your app will support
            var supportedCultures = new List<CultureInfo>()
            {
                new CultureInfo("de")
            };
            // set the default culture
            options.DefaultRequestCulture = new RequestCulture("de");
            options.SupportedCultures = supportedCultures;
            options.SupportedUICultures = supportedCultures;
            options.RequestCultureProviders = new List<IRequestCultureProvider>() {
                new QueryStringRequestCultureProvider() // Here, You can also use other
                localization provider
            };
        });
        services.AddSingleton(typeof(ISyncfusionStringLocalizer),
            typeof(SampleLocalizer));
    }
}

```

- Then, write a **class** by inheriting **ISyncfusionStringLocalizer** interface and override the **Manager** property to get the resource file details from the application end.

CSHARP

```

using Syncfusion.Blazor;
namespace blazorCalendars
{
    public class SampleLocalizer: ISyncfusionStringLocalizer
    {
        public string Get(string key)
        {
            return this.Manager.GetString(key);
        }
        public System.Resources.ResourceManager Manager
        {
            get
            {
                return blazorCalendars.Resources.SyncfusionBlazorLocale.ResourceManager;
            }
        }
    }
}

```

- Add **.resx** file to Resource folder and enter the key value (Locale Keywords) in the **Name** column and the translated string in the Value column as follows.

| **Name** | **Value (in Deutsch culture)** |

| --- | --- |

| DateTimePicker_Placeholder | Wählen Sie ein Datum und eine Uhrzeit |

| DateTimePicker_Today | Heute |

- Finally, Specify the culture for DateTimePicker using **locale** property.

ASPX-CS

```
@using Syncfusion.Blazor.Calendars
<SfDateTimePicker TValue="DateTime?" Locale="de"></SfDateTimePicker>
```

Blazor WebAssembly

By default, the DateTimePicker week and month names are specific to the **American English** culture. It utilizes the **Blazor Internationalization** package to parse and format the date object based on the culture by using the official [UNICODE CLDR](#) JSON data.

The following steps explain how to render the DateTimePicker in German culture ('de-DE') in Blazor Web Assembly application.

- Open the **program.cs** file and add the below configuration in the **Builder ConfigureServices** function as follows.

CSHARP

```
using Syncfusion.Blazor;
using Microsoft.AspNetCore.Builder;
namespace WebAssemblyLocale
{
    public class Program
    {
        public static async Task Main(string[] args)
        {
            ....
            ....
            builder.Services.Configure<RequestLocalizationOptions>(options =>
            {
                // Define the list of cultures your app will support
                var supportedCultures = new List<System.Globalization.CultureInfo>()
                {
                    new System.Globalization.CultureInfo("en-US"),
                    new System.Globalization.CultureInfo("de"),
                };
                // Set the default culture
                options.DefaultRequestCulture = new
                Microsoft.AspNetCore.Localization.RequestCulture("de");
                options.SupportedCultures = supportedCultures;
                options.SupportedUICultures = supportedCultures;
            });
        }
    }
}
```

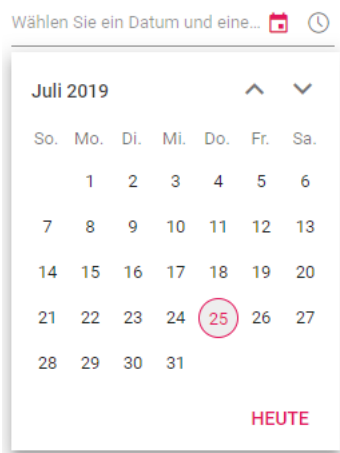
```
options.RequestCultureProviders = new
List<Microsoft.AspNetCore.Localization.IRequestCultureProvider>() {
new Microsoft.AspNetCore.Localization.QueryStringRequestCultureProvider()
};
});
....
....
}
}
}
```

- Download the required locale packages to render the Blazor DateTimePicker component with specified locale.
- To download the locale definition of Blazor components, use this [link](#).
- After downloading the `blazor-locale` package, copy the `blazor-locale` folder with required local definition file into `wwwroot` folder.
- By default, the `blazor-locale` package contains the localized text for static text present in components like button text, placeholder, tooltip, and more.
- Set the culture by using the `SetCulture` method.

ASPX-CS

```
@using Syncfusion.Blazor.Calendars
@inject HttpClient Http;
<SfDateTimePicker TValue="DateTime?" Locale="de"></SfDateTimePicker>
@code {
[Inject]
protected IJSRuntime JsRuntime { get; set; }
protected override async Task OnInitializedAsync()
{
this.JsRuntime.Sf().LoadLocaleData(await Http.GetJsonAsync<object>("blazor-
locale/src/de.json")).SetCulture("de");
}
}
```

The output will be as follows.



Customize the localized text

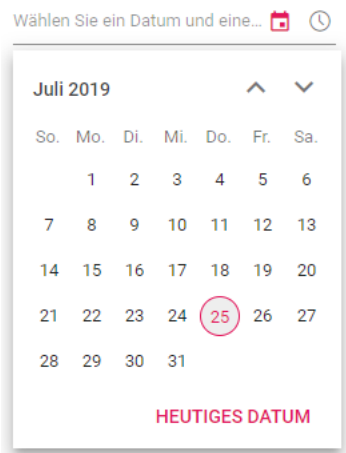
- You can change the localized text of particular component by editing the `wwwroot/blazor-locale/src/{{locale name}}.json` file.
- In the following code, modified the localized text of `today button` and `placeholder` in `de` culture.

[`wwwroot/blazor-locale/src/de.json`]

CSHARP

```
{
  "de": {
    "datetimepicker": {
      "today": "Heutiges Datum",
      "placeholder": "Wählen Sie ein Datum und eine Uhrzeit aus"
    }
  }
}
```

The output will be as follows.



Right-To-Left

The `DateTimePicker` supports RTL (right-to-left) functionality for languages like Arabic and Hebrew to displays the text in the right-to-left direction. Use [EnableRtl](#) property to set the RTL direction.

The following code example initialize the `DateTimePicker` component in `Arabic` culture.

ASPX-CS

```
@using Syncfusion.Blazor.Calendars
@inject HttpClient Http;
<SfDateTimePicker TValue="DateTime?" Locale="ar"
EnableRtl=true></SfDateTimePicker>
@code {
    [Inject]
    protected IJSRuntime JsRuntime { get; set; }
    protected override async Task OnInitializedAsync()
    {

```

```
this.JsRuntime.Sf().LoadLocaleData(await Http.GetJsonAsync<object>("blazor-locale/src/ar.json")).SetCulture("ar");
}
```

The output will be as follows.



Native Events in Blazor Datetime Picker Component

The following section explains steps to include native events and pass data to event handler in DateTimePicker component.

Bind native events to DateTimePicker

You can access any native event by using on `<event>` attribute with a component. The attribute's value is treated as an event handler.

In the following example, the KeyPressed method is called every time the key is pressed on input.

ASPX-CS

```
@using Syncfusion.Blazor.Calendars
<SfDateTimePicker TValue="DateTime?"
@onkeypress="@KeyPressed"></SfDateTimePicker>
@code {
public void KeyPressed() {
Console.WriteLine("Key Pressed!");
}
}
```

Also, you can rewrite the previous example code as follows using Lambda expressions.

ASPX-CS

```
@using Syncfusion.Blazor.Calendars
```

```
<SfDateTimePicker TValue="DateTime?" @onkeypress="@(() =>
Console.WriteLine("Key Pressed!"))"></SfDateTimePicker>
```

Pass event data to event handler

Blazor provides set of argument types for map to native events. The following is the list of event types and event arguments:

- Focus Events - FocusEventArgs
- Mouse Events - MouseEventArgs
- Keyboard Events - KeyboardEventArgs
- Input Events - ChangeEventArgs/EventArgs
- Touch Events – TouchEventArgs
- Pointer Events – PointerEventArgs

In the following example, the KeyPressed method is called every time any key is pressed inside input. But the message will be printed when you press "5" key.

ASPX-CS

```
@using Syncfusion.Blazor.Calendars
<SfDateTimePicker TValue="DateTime?" @onkeypress='@(e =>
KeyPressed(e))'></SfDateTimePicker>
@code {
public void KeyPressed(KeyboardEventArgs args)
{
if (args.Key == "5")
{
Console.WriteLine("5 was pressed");
}
}
}
```

Using Lambda expression also, you can pass the event data to the event handler.

List of Native events supported

| List of Native events | | |

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

| onclick | onblur | onfocus | onfocusout |

| onmousemove | onmouseover | onmouseout | onmousedown | onmouseup |

| ondblclick | onkeydown | onkeyup | onkeypress |

| ontouchend | onfocusin | onmouseup | ontouchstart |

Strict Mode in Blazor Datetime Picker Component

The [StrictMode](#) is an act, that allows the users to enter only the valid date and time within the specified **Min/Max** range in text box. If the input entered is invalid, then the component will stay with the previous value. Else, if the datetime is out of range, then the component will set the datetime to the Min/Max value.

The following example demonstrates the DateTimePicker in **StrictMode** with Min/Max range of 5/5/2019 2:00 AM to 5/25/2019 2:00 AM. Here, it allows to enter only the valid date and time within the specified range.

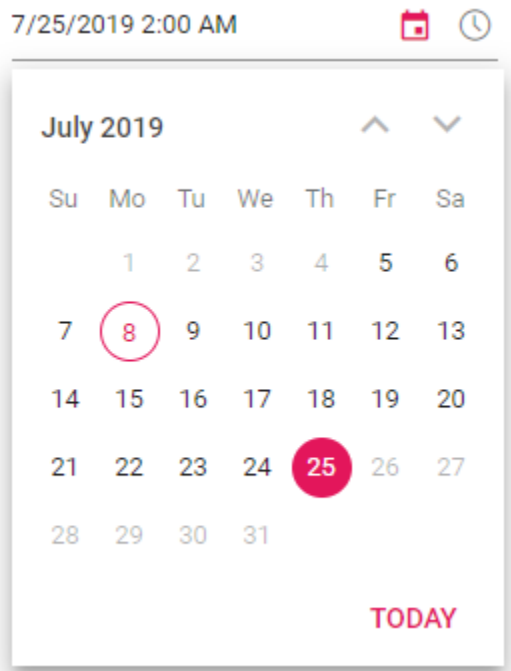
- If you are trying to enter the out-of-range value like 5/28/2019, then the value will set to the **Max** value as 5/25/2019 2:00 AM since the value 28 is greater than the **Max** value of 25.
- If you are trying to enter the invalid date, then the Value will stay with the previous value.

The following code demonstrates the DateTimePicker with **StrictMode** **true**.

ASPX-CS

```
@using Syncfusion.Blazor.Calendar
<SfDateTimePicker TValue="DateTime?" Min="@MinDate" Max="@MaxDate"
StrictMode=true Value="@DateValue"></SfDateTimePicker>
@code {
public DateTime MinDate {get;set;} = new
DateTime(DateTime.Now.Year,DateTime.Now.Month, 05, 02, 00, 00);
public DateTime MaxDate {get;set;} = new DateTime(DateTime.Now.Year,
DateTime.Now.Month, 25, 02, 00, 00);
public DateTime? DateValue {get;set;} = new DateTime(DateTime.Now.Year,
DateTime.Now.Month, 28, 02, 00, 00);
}
```

The output will be as follows.



By default, the DateTimePicker act in **StrictMode** as **false** state, that allows you to enter the invalid or out-of-range datetime in text box.

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

The following code demonstrates the `StrictMode` as `false`. Here, it allows to enter the valid or invalid value in textbox.

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

ASPX-CS

```
@using Syncfusion.Blazor.Calendars
<SfDateTimePicker TValue="DateTime?" Min="@MinDate" Max="@MaxDate"
StrictMode=false Value="@DateValue"></SfDateTimePicker>
@code {
public DateTime MinDate {get;set;} = new
DateTime(DateTime.Now.Year,DateTime.Now.Month, 05, 02, 00, 00);
public DateTime MaxDate {get;set;} = new DateTime(DateTime.Now.Year,
DateTime.Now.Month, 25, 02, 00, 00);
public DateTime? DateValue {get;set;} = new DateTime(DateTime.Now.Year,
DateTime.Now.Month, 28, 02, 00, 00);
}
```

The output will be as follows.

7/28/2019 2:00 AM



If the value of `Min` or `Max` properties changed through code behind, you have to update the `Value` property to set within the range.

Accessibility in Blazor Datetime Picker Component

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

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

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

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

- **aria-expanded:** Attributes indicates the state of a collapsible element.
- **aria-disabled:** Attribute indicates the disabled state of this DateTimePicker component.
- **aria-activedescendent:** Attribute helps in managing the current active child of the DateTimePicker component.

Keyboard Interaction

You can use the following keys to interact with the DateTimePicker. The component implements the keyboard navigation support by following the [WAI-ARIA practices](#).

DateTimePicker supports the below list of shortcut keys:

Input Navigation

Before opening the popup, use the following keys to control the popup element.

Keys	Description
------	-------------

--- ---	
-----------	--

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

Alt + Down Arrow + Alt + Down Arrow	Toggles between two popups
-------------------------------------	----------------------------

Calendar Navigation

Use the following keys to interact with the Calendar after the DatePicker popup has opened:

Keys	Description
------	-------------

--- ---	
-----------	--

Upper Arrow	Focuses the previous week date.
-------------	---------------------------------

Down Arrow	Focuses the next week date.
------------	-----------------------------

Left Arrow	Focuses the previous date.
------------	----------------------------

Right Arrow	Focuses the next date.
-------------	------------------------

Home	Focuses the first date in the month.
------	--------------------------------------

End	Focuses the last date in the month.
-----	-------------------------------------

Page Up	Focuses the same date in the previous month.
---------	--

Page Down	Focuses the same date in the next month.
-----------	--

Enter	Selects the currently focused date.
-------	-------------------------------------

Shift + Page Up	Focuses the same date in the previous year.
-----------------	---

Shift + Page Down	Focuses the same date in the previous year.
-------------------	---

Control + Upper Arrow	Moves into the inner level of view like month-year, year-decade
-----------------------	---

Control + Down Arrow	Moves out from the depth level view like decade-year, year-month
----------------------	--

Control +Home	Focuses the starting date in the current year.
---------------	--

Control +End	Focuses the ending date in the current year.
--------------	--

Use the following shortcut keys to interact with the TimePicker after the TimePicker Popup has opened:

Keys	Description
------	-------------

--- ---	
-----------	--

Upper Arrow	Navigates and selects the previous item.
-------------	--

Down Arrow	Navigates and selects the next item.
------------	--------------------------------------

Left Arrow	Moves the cursor towards arrow key pressed direction.
------------	---

Right Arrow	Moves the cursor towards arrow key pressed direction.
-------------	---

- | Home | Navigates and selects the first item. |
- | End | Navigates and selects the last item. |
- | Enter | Selects the currently focused item and close the popup. |
- | Alt + Upper Arrow | Closes the popup. |
- | Alt + Down Arrow | Opens the popup. |
- | Esc | Closes the popup. |

To focusout the DateTimePicker component, use the **t** keys. For additional information about native event, [click](#) here.

ASPX-CS

```
@using Syncfusion.Blazor.Calendars
<SfDateTimePicker TValue="DateTime?" @onkeypress="@ (e => KeyPressed(e)) "
@ref="DateTimeObj"></SfDateTimePicker>
@code {
public SfDateTimePicker<DateTime?> DateTimeObj;
public void KeyPressed(KeyboardEventArgs args)
{
if (args.Key == "t")
{
this.DateTimeObj.FocusOutAsync();
}
}
}
```

Events in Blazor Datetime Picker Component

This section explains the list of events of the DateTimePicker component which will be triggered for appropriate DateTimePicker actions.

Event Name(v17.1.) /Event Name(v17.2.)

change | [ValueChange](#)

close | [OnClose](#)

open | [OnOpen](#)

renderDayCell | [OnRenderDayCell](#)

Blur

Blur event triggers when the input loses the focus.

ASPX-CS

```
@using Syncfusion.Blazor.Calendars
<SfDateTimePicker TValue="DateTime?" >
<DateTimePickerEvents TValue="DateTime?"
Blur="BlurHandler"></DateTimePickerEvents>
</SfDateTimePicker>
@code{
public void BlurHandler(Syncfusion.Blazor.Calendars.BlurEventArgs args)
{
}
```

```
// Here you can customize your code
}
```

ValueChanged

ValueChanged event triggers when the Calendar value is changed.

ASPX-CS

```
@using Syncfusion.Blazor.Calendar
<SfDateTimePicker TValue="DateTime?" >
  <DateTimePickerEvents TValue="DateTime?"
  ValueChange="ValueChangedHandler"></DateTimePickerEvents>
</SfDateTimePicker>
@code{
public void ValueChangeHandler(ChangedEventArgs<DateTime?> args)
{
  // Here you can customize your code
}
```

OnClose

OnClose event triggers when popup is closed.

ASPX-CS

```
@using Syncfusion.Blazor.Calendar
<SfDateTimePicker TValue="DateTime?" >
  <DateTimePickerEvents TValue="DateTime?"
  OnClose="OnCloseHandler"></DateTimePickerEvents>
</SfDateTimePicker>
@code{
public void OnCloseHandler(object args)
{
  // Here you can customize your code
}
```

Created

Created event triggers when DateTimePicker is created.

ASPX-CS

```
@using Syncfusion.Blazor.Calendar
<SfDateTimePicker TValue="DateTime?" >
  <DateTimePickerEvents TValue="DateTime?"
  Created="CreatedHandler"></DateTimePickerEvents>
</SfDateTimePicker>
@code{
public void CreatedHandler(object args)
{
  // Here you can customize your code
}
```


Destroyed

Destroyed event triggers when DateTimePicker is destroyed.

ASPX-CS

```
@using Syncfusion.Blazor.Calendar
<SfDateTimePicker TValue="DateTime?" >
  <DateTimePickerEvents TValue="DateTime?"
  Destroyed="DestroyHandler"></DateTimePickerEvents>
</SfDateTimePicker>
@code{
public void DestroyHandler(object args)
{
  // Here you can customize your code
}
```

Focus

Focus event triggers when the input gets focus.

ASPX-CS

```
@using Syncfusion.Blazor.Calendar
<SfDateTimePicker TValue="DateTime?" >
  <DateTimePickerEvents TValue="DateTime?"
  Focus="FocusHandler"></DateTimePickerEvents>
</SfDateTimePicker>
@code{
public void FocusHandler(object args)
{
  // Here you can customize your code
}
```

Navigated

Navigated event triggers when the Calendar is navigated to another level or within the same level of view

ASPX-CS

```
@using Syncfusion.Blazor.Calendar
<SfDateTimePicker TValue="DateTime?" >
  <DateTimePickerEvents TValue="DateTime?"
  Navigated="NavigateHandler"></DateTimePickerEvents>
</SfDateTimePicker>
@code{
public void NavigateHandler(NavigatedEventArgs args)
{
  // Here you can customize your code
}
```

OnOpen

OnOpen event triggers when popup is opened.

ASPX-CS

```
@using Syncfusion.Blazor.Calendars
<SfDateTimePicker TValue="DateTime?" >
  <DateTimePickerEvents TValue="DateTime?"
    OnOpen="OpenHandler"></DateTimePickerEvents>
</SfDateTimePicker>
@code{
public void OpenHandler(object args)
{
    // Here you can customize your code
}
}
```

OnRenderDayCell

OnRenderDayCell event triggers when each day cell of the Calendar is rendered.

ASPX-CS

```
@using Syncfusion.Blazor.Calendars
<SfDateTimePicker TValue="DateTime?" >
  <DateTimePickerEvents TValue="DateTime?"
    OnRenderDayCell="onRenderDayCellHandler"></DateTimePickerEvents>
</SfDateTimePicker>
@code{
public void onRenderDayCellHandler(RenderDayCellEventArgs args)
{
    // Here you can customize your code
}
}
```

DateTimePicker will be limited with these events and new events will be added in future based on the user requests. If the event you are looking for is not in the list, then please request [here](#).

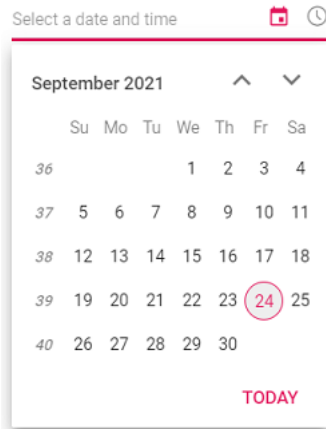
Week Number in Blazor DateTimePicker Component

You can enable WeekNumber in the DateTimePicker by using the [WeekNumber](#) property.

ASPX-CS

```
@using Syncfusion.Blazor.Calendars
<SfDateTimePicker TValue="DateTime?" Width="250px"
  WeekNumber="true"></SfDateTimePicker>
```

The output will be as follows.



Week Rule

You can enable **WeekRule** in the DateTimePicker by using the [WeekRule](#) property. This property provide an option to specify the rule for defining the first week of the year. Please find the possible values of **WeekRule** property.

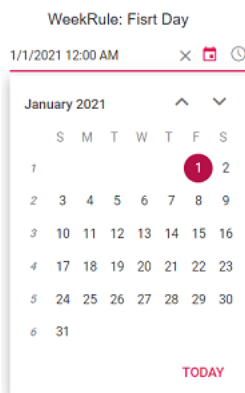
Types | Description

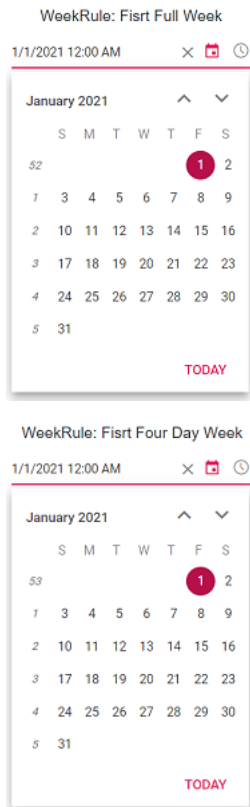
FirstDay | Set the first week of the year's week number to be started from 1. Then it followed as 1, 2, 3 ...

FirstFullWeek | Set the first week of the year's week number to be started from 52 or 53 (i.e December last week's week Number). Then it followed as 53, 1, 2 ...

FirstFourDayWeek | Set the week number based on the majority of dates present in the week for the respected months. If January dates are presented in the week more than December, the first week of the year's week number will be started from 1. If December dates are presented in the week more than January, the first week of the year's week number will be started from 52 or 53.

The output will be as follows.





Special Dates in Blazor DateTimePicker Component

You can customize specific dates in a DateTimePicker by using the [OnRenderDayCell](#) event. This event gets triggered on each day cell element creation that allows you to customize or disable the specific dates in the DateTimePicker. Here, list of dates in the current month are customized with custom styles by adding the personal-appointment and official-appointment class.

ASPX-CS

```
@using Syncfusion.Blazor.Calendars
<div class="control-wrapper">
<SfDateTimePicker TValue="DateTime?" Placeholder="Select a date and time"
ShowClearButton="true" @bind-Value="@SelectedDate">
<DateTimePickerEvents TValue="DateTime?" OnRenderDayCell="CustomDates"
ValueChange="OnChange"></DateTimePickerEvents>
</SfDateTimePicker>
</div>
<div id="display-date">
<span>Selected Day : @SelectedValue</span>
</div>
@code {
public DateTime? SelectedDate { get; set; }
public string SelectedValue { get; set; } =
DateTime.Now.ToString("M/d/yyyy");
public DateTime? CurrentDate { get; set; } = DateTime.Now;
public void CustomDates(RenderDayCellEventArgs args)
{
var CurrentMonth = CurrentDate.Value.Month;
if (args.Date.Month == CurrentMonth && (args.Date.Day == 7 || args.Date.Day
== 14 || args.Date.Day == 24 || args.Date.Day == 29)) {
```

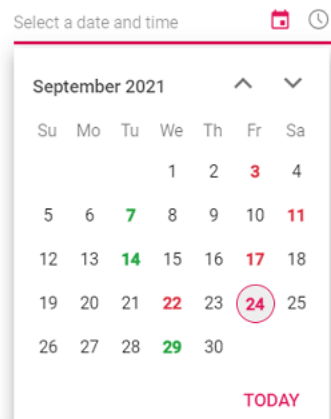
```

args.CellData.ClassList += " personal-appointment";
}
if (args.Date.Month == CurrentMonth && (args.Date.Day == 3 || args.Date.Day
== 11 || args.Date.Day == 17 || args.Date.Day == 22))
{
args.CellData.ClassList += " official-appointment";
}
}
public void OnChange(ChangedEventArgs<DateTime?> args)
{
var Count = 0;
var CurrentMonth = CurrentDate.Value.Month;
if (args.Value.Value.Month == CurrentMonth && (args.Value.Value.Day == 7 ||
args.Value.Value.Day == 14 || args.Value.Value.Day == 24 ||
args.Value.Value.Day == 29))
{
this.SelectedValue = this.SelectedDate?.ToString("M/d/yyyy") + " (Personal
appointment)";
Count++;
}
if (args.Value.Value.Month == CurrentMonth && (args.Value.Value.Day == 3 ||
args.Value.Value.Day == 11 || args.Value.Value.Day == 17 ||
args.Value.Value.Day == 22))
{
this.SelectedValue = this.SelectedDate?.ToString("M/d/yyyy") + " (Official
appointment)";
Count++;
}
if (Count == 0)
{
this.SelectedValue = this.SelectedDate?.ToString("M/d/yyyy");
}
}
}
<style>
#display-date {
max-width: 270px;
padding: 15px 0;
font-size: 13px;
}
.control-wrapper {
width: 300px;
margin: 0 auto;
padding-top: 50px;
}
.e-calendar .e-content .e-cell.personal-appointment span.e-day,
.e-calendar .e-content td:hover.e-cell.personal-appointment span.e-day,
.e-calendar .e-content td.e-selected.e-focused-date.e-cell.personal-
appointment span.e-day {
color: #28a745;
font-weight: 800;
}
.e-calendar .e-content .e-cell.official-appointment span.e-day,
.e-calendar .e-content td:hover.e-cell.official-appointment span.e-day,
.e-calendar .e-content td.e-selected.e-focused-date.e-cell.official-
appointment span.e-day {
color: #dc3545;

```

```
font-weight: 800;
}
.e-calendar .e-content td.e-selected.e-focused-date.e-cell.personal-
appointment span.e-day,
.e-calendar .e-content td.e-selected.e-focused-date.e-cell.official-
appointment span.e-day {
background-color: #b511485e;
}
</style>
```

The output will be as follows.



How To

Disable the Blazor DateTimePicker Component

To disable the DateTimePicker, set its [Enabled](#) property to `false`.

The following code demonstrates the disabled component.

ASPX-CS

```
@using Syncfusion.Blazor.Calendars
<SfDateTimePicker TValue="DateTime?" Enabled=false
Value="@DateTimeValue"></SfDateTimePicker>
@code {
public DateTime? DateTimeValue { get; set; } = DateTime.Now;
}
```

The output will be as follows.



Set the Placeholder in Blazor Datetime Picker Component

The following code demonstrates how to set `Placeholder` in the DateTimePicker component.

Using [Placeholder](#), you can display a short hint in the input element.

ASPX-CS

```
@using Syncfusion.Blazor.Calendars
<SfDateTimePicker TValue="DateTime?" Placeholder="Choose a
datetime"></SfDateTimePicker>
```

The output will be as follows.

